Null Messages, Information and Coordination

Raïssa Nataf
Technion, Haifa, Israel

Guy Goren
Protocol Labs, Haifa, Israel

Yoram Moses
Technion, Haifa, Israel

Abstract

This paper investigates the role that null messages play in synchronous systems with and without failures, and provides necessary and sufficient conditions on the structure of protocols for information transfer and coordination there. We start by introducing a new and more refined definition of null messages. A generalization of message chains that allow these null messages is provided, and is shown to be necessary and sufficient for information transfer in reliable systems. Coping with crash failures requires a much richer structure, since not receiving a message may be the result of the sender's failure. We introduce a class of communication patterns called *resilient message blocks*, which impose a stricter condition on protocols than the *silent choirs* of Goren and Moses (2020). Such blocks are shown to be necessary for information transfer in crash-prone systems. Moreover, they are sufficient in several cases of interest, in which silent choirs are not. Finally, a particular combination of resilient message blocks is shown to be necessary and sufficient for solving the Ordered Response coordination problem.

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, knowledge analysis

Digital Object Identifier 10.4230/LIPIcs.DISC.2023.30

Related Version *Full Version*: `https://arxiv.org/abs/2208.10866` [17]

Funding *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

Communication and coordination in distributed systems depend crucially on properties of the model at hand. In synchronous systems in which processes have clocks and message transmission times are bounded, sending explicit messages is not the only way to transmit information. Suppose that a sender s needs to transmit its (binary) initial value v_s to a destination process d, in a system in which messages are delivered in 1 time step. If s follows a protocol by which it sends d a message at time 0 in case $v_s = 0$ and does not send anything if $v_s = 1$, then d can learn that $v_s = 1$ at time 1 without receiving any messages. Lamport called this "*sending a message by not sending a message*" in [13], and he referred to not sending a message over a communication channel at a given time t as sending a "*null message*." In this paper we provide a new and more precise definition of null messages, and investigate the general role that null messages play in information transfer and in coordination in synchronous systems with and without failures.

In particular, our results extend and generalize those of Goren and Moses in [7], who were the first to explicitly consider how silence can be used in systems with crash failures.

The possibility of failures makes information transfer a rather subtle issue. Denote by f an *a priori* upper bound on the number of failures per execution. Consider the following protocol, which we denote P_1: In the first round process s sends a message to p reporting whether its initial value v_s is 1 or 0. In round 2, if process p has received a message stating that $v_s = 1$ it keeps silent, and it sends an actual message to d otherwise. A run r_1 of P_1 in which $v_s = 1$ and no process fails is depicted in Figure 1. (In our figures, solid arrows represent actual messages and dashed arrows represent null messages.) Note that if $f = 0$, then process p receives the first round message from s in every run of P_1. Consequently, if $v_s = 1$ then following the second round, d learns that $v_s = 1$ since it did not hear from p. Now assume that one process may crash ($f = 1$). In this case r_1, where none fails, is a legal execution of P_1 but d is not informed that $v_s = 1$ in r_1. This is because d cannot distinguish r_1 from a run in which $v_s = 0$ and p crashes before sending its message to d.

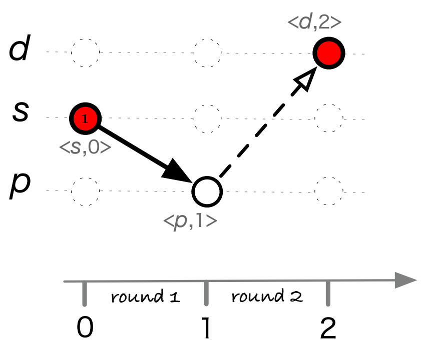

Figure 1 The run r_1 of P_1 in which d is informed that $v_s = 1$ when $f = 0$ but **not** when $f = 1$.

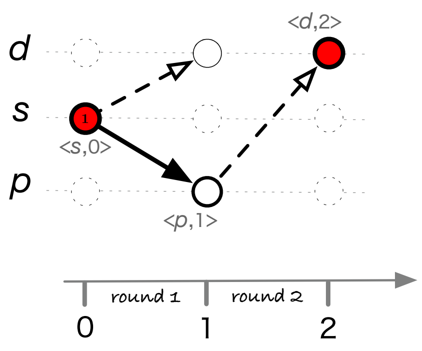

Figure 2 The run r_2 of P_2 in which a silent choir informs d that $v_s = 1$ when $f = 1$.

Consider now a protocol P_2 that differs from P_1 only in that according to P_2 process s should send d a message at time 0 if and only if $v_s = 0$. (Process s remains silent if $v_s = 1$.) Figure 2 depicts the run r_2 of P_2 where $v_s = 1$ and no failures occur. Observe that in case $f = 1$, process d is informed in r_2 that $v_s = 1$. As before, d cannot observe in r_2 whether p has crashed. However, since $f = 1$, at most one of the missing messages to d can be explained by a process crash. The other missing message must be caused by the fact that $v_s = 1$. Note that exactly the same messages are sent in r_1 and r_2. Process d obtains different information in the two cases because the protocols are different: In particular, s keeps silent toward d only under certain conditions of interest according to P_2 while it always keeps silent according to P_1. This is what provides d genuine information.

The above discussion motivates a new and more refined definition of null messages. While [13] considers not receiving a message as the receipt of a null message, we define a null message to be sent by a process i to its neighbor j at time t in a given execution if process i does not send an actual message at time t, and there is at least one execution of the protocol in which i *does* send j an actual message at time t. (A formal definition appears in Section 2.) With such a definition, a null message is guaranteed to carry some nontrivial information.

Goren and Moses showed in [7] that information can be transmitted in silence even when crashes may occur. Their *Silent Choir* Theorem states a necessary condition for d to learn the initial value of s in a crash-prone system without a message chain from s. For failure-free executions, their necessary condition becomes the following: If d knows the value of v_s at time $m > 0$ without an actual message chain (i.e., a message chain exclusively composed of actual messages) from s having reached it, there are at least $f+1$ processes that receive an actual message chain from s by time $m-1$ and send no message to d at time $m-1$. These processes are called a *silent choir*. In r_2, process d learns the value of s at time 2, and we can see in Figure 2 that the set $\{s,p\}$ constitutes a silent choir. However, as we shall see, while being necessary for information transfer in crash-prone synchronous systems, the Silent Choir Theorem's conditions are not sufficient, even for failure-free executions.

Consider again the run r_1 of Figure 1 and assume $f = 1$. The set $\{s,p\}$ forms a silent choir, i.e., the conditions of the Silent Choir Theorem hold. However, as we described above, d does not learn that $v_s = 1$ in r_1. Process s, who belongs to the silent choir here, *never* sends d a message under P_1, and so its silence does not form a null message according to our new definition. Naturally, strengthening the Silent Choir condition by requiring that the processes of the silent choir actually send null messages at time $m-1$, i.e., one time unit before d gains the knowledge that $v_s = 1$, could make it sufficient. However, the condition would then **not** be necessary. For example, as depicted in Figure 2, s does not send d a null message in the second round of r_2 (which is at time $m-1$ in the language of the Silent Choir Theorem). Notice that, in addition, members of the silent choir do not necessarily send null messages to d. Indeed, they do not even need to be neighbors of d.

Beyond the fundamental value of studying null messages to understand the differences between synchronous and asynchronous models of distributed systems, judicious use of null messages can lead to considerable savings. For a concrete example, consider a network structured as depicted in Figure 3, in which there are different costs for sending over different channels. This can arise, for example, from the three intermediate processes residing at the same site or belonging to the same organization as s, while d is across the ocean, or just connected via expensive channels. Assume, in addition, that d needs to know the value of v_s, where normally $v_s = 1$ and only very rarely $v_s \neq 1$. Finally, we wish to be able to overcome up to $\boldsymbol{f = 2}$ process crashes. While sending an actual message chain from s to d would cost \$1001, null messages can be used to inform d that $v_s = 1$ at a cost of \$2 if $v_s = 1$ and no failures occur (see Figure 4). With such a solution, if $v_s = 0$ then the cost may in the worst case be as high as \$3003. But if the latter is rare and the former is very common, such use of null messages can provide a clear advantage. E.g., if for every 100 runs in which $v_s = 1$ and no failures occur we expect to see 2 runs where this is not the case, then using null messages for the 102 runs will cost at most \$6206, compared to \$102102 spent by a protocol that uses only actual messages in the network of Figure 3.

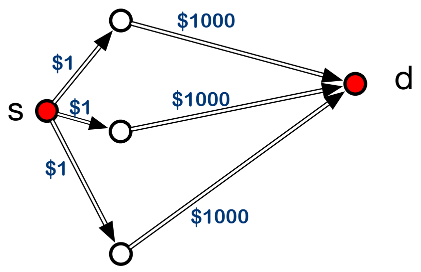

Figure 3 A network with different communication costs. Sending an actual message chain from s to d costs \$1001.

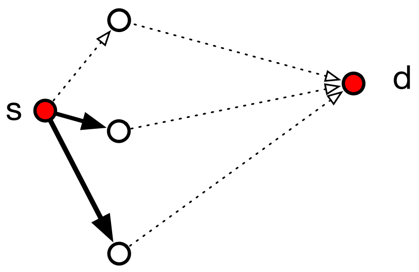

Figure 4 The sender s can inform d that $v_s = 1$ at a cost of \$2 in a failure free run assuming a bound of $f = 2$ failures.

This paper investigates the role of null messages for information flow and coordination in synchronous systems with crash failures. Its main contributions are:

1. We provide a new definition of null messages, whereby not sending a message is considered to be a null message only if it conveys nontrivial information. Moreover, we formalize an essential aspect of the synchronous model, by proving that *enhanced message chains*, which can contain both actual and null messages, are necessary for information transfer in synchronous systems.
2. We strengthen this result by proving that in order for there to be information transfer from a process s to d, process d must **know** that an enhanced message chain from s has reached it. This result plays an important role in the analysis of information transfer in the presence of failures.
3. We identify communication patterns called *resilient message blocks*, which are necessary for information transfer in crash-prone synchronous systems, proving a stronger and more general theorem than the Silent Choir Theorem of [7]. Based on this theorem, we provide necessary and sufficient conditions that characterize protocols for nice-run information transfer, in which the transfer should succeed in failure-free executions, and for Robust information transfer in which it should succeed more generally.
4. Finally, we provide an analysis of communication requirements from protocols solving the Ordered Response coordination problem, based on resilient blocks.

This paper is structured as follows. In Section 2, we define the model and present preliminary definitions and results regarding null messages, knowledge, and communication graphs, which are used throughout the analysis. We show in Section 2.2 that it is impossible to achieve information transfer from a process s to another process d in synchronous systems without constructing an enhanced message chain from s to d. We then prove the stronger fact (in Theorem 8) that in order for d to know the value of v_s, it must *know* that an enhanced message chain from s to d exists. This is followed in Section 3 by an analysis that identifies the communication patterns that must be created by a protocol in runs where d learns the value of v_s Theorem 11. These structures, called *resilient message blocks*, can involve multiple enhanced message chains that must be arranged in a particular manner. The analysis is applied to characterize necessary and sufficient conditions on the communication patterns solving Nice-run Information Transfer in Section 4. Based on Section 3 and the results of application of Section 4, necessary and sufficient conditions for solving the Ordered Response coordination problem are presented in Section 5. Finally, patterns solving Robust Information Transfer are characterized in Section 6. A brief conclusion is presented in Section 7. Due to space restrictions, the proofs for Sections 5 and 6 appear in the paper's Arxiv version [17].

1.1 Related Work

Lamport's seminal paper [12] focuses on the role of message chains in asynchronous message passing systems. Indeed, Chandy and Misra showed in [4] that the only way in which knowledge about the state of the system at remote sites can be gained in asynchronous systems is via the construction of message chains. As mentioned above, in his later paper [13], Lamport points out that in synchronous systems information can also be conveyed using null messages. In a more recent paper [3], Ben Zvi and Moses analyzed knowledge gain and coordination in a model in which processes are reliable (no process ever crashes) and share a global clock, and there are upper bounds (possibly greater than 1) on message transmission times along each of the channels in the network. They extend the notion of a message chain to so-called *syncausal* message chains, which are sequences consisting of a combination of time intervals that correspond to the upper bounds and actual messages. They show that syncausal chains are necessary and sufficient for point-to-point information transfer when

$f = 0$. Moreover, they define a coordination problem called Ordered Response (which we revisit in Section 5) and show that a communication pattern they call a *centipede*, which generalizes message chains for their model, is necessary and sufficient for solving this problem. As mentioned in the Introduction and as will be defined in Section 2, not sending a message does not always count as a null message, even if message delays are bounded. The notion of an enhanced message chain thus refines that of a syncausal message chain: Every enhanced message chain is a syncausal message chain, but the converse is not true.

Our paper extends the work [7]. Their Silent Choir Theorem, discussed in the previous section, gives necessary conditions that are not sufficient even for failure-free executions. In the current paper we take the further step of characterizing necessary and sufficient properties of communication patterns that solve this problem, and investigate the role of silence in the more general coordination problem of Ordered Response coordination problem. None of the previous works (e.g., [13, 3, 7]) requires not sending a message to be informative in order to count as a null message. Making this requirement plays a technically significant role in the analyses performed in the current paper.

In Sections 4–6 we consider the design of protocols that are required to behave in a good way in the common case, which in this case is when the initial values are appropriate and no failures occur. Focusing on the design of protocols that are optimized for the common case has a long tradition in distributed computing (see, e.g., [15, 11, 14]). In our synchronous model Amdur, Weber, Hadzilacos and Halpern use them in order to design efficient protocols for Byzantine agreement [1, 10]. Guerraoui and Wang and others use them for Atomic Commitment [9, 7]. Solutions for Consensus in a synchronous Byzantine model optimized for the common case appeared in [8], also making explicit use of null messages. However, how null messages can be used for information transfer and coordination has not been characterized in a formal way.

2 Model and Preliminary Results

We follow the modeling of [7]. We consider a standard synchronous message-passing model with a set $\mathbb{P}$ of $N > 2$ processes and benign crash failures. For convenience, one of the processes will be denoted by s and called the *source*, while another, d can be considered as the *destination*. Processes are connected via a communication network defined by a directed graph $(\mathbb{P}, \mathsf{ch})$ where an edge from process i to process j is called a *channel*, and denoted by $\mathsf{ch}_{i,j}$. We assume that the receiver of a message detects the channel over which it was delivered, and thus knows the identity of the sender. The model is *synchronous*: All processes share a discrete global clock that starts at time 0 and advances by increments of one. Communication in the system proceeds in a sequence of *rounds*, with round $m+1$ taking place between time m and time $m+1$, for $m \geq 0$. A message sent at time m (i.e., in round $m+1$) from a process i to j will reach j at time $m+1$, i.e., at the end of round $m+1$. In every round, each process performs local computations, sends a set of messages to other processes, and finally receives messages sent to it by other processes during the same round. At any given time $m \geq 0$, a process is in a well-defined ***local state***. We denote by $r_i(m)$ the local state of process i at time m in the run r. For simplicity, we assume that the local state of a process i consists of its initial value v_i, the current time m, and the sequence of the events that i has observed (including the messages it has sent and received) up to that time. In particular, its local state at time 0 has the form $(v_i, 0, \{\})$. We focus on deterministic protocols, so a ***protocol*** Q describes what messages a process should send and what decisions it should take, as a function of its local state.

Processes in our model are prone to crash failures. A faulty process in a given execution fails by *crashing* at a given time. A process that crashes at time t is completely inactive from time $t+1$ on, and so it performs no actions and in particular sends no messages in round $t+2$ and in all later rounds. It behaves correctly up to and including round t. Finally, in round $t+1$ (which takes place between time t and time $t+1$) this process sends a (possibly strict) subset of the messages prescribed by the protocol. For ease of exposition we assume that the process does not perform any additional local actions (e.g., decisions) in round $t+1$.

For ease of exposition, we say that a process that has not failed up to and including time t is *active* at time t. We will consider the design of protocols that are required to tolerate up to f crashes. We denote by γ^f the model described above in which no more than f processes crash in any given run. We assume that a protocol has access to the values of N and f as well as to the communication network $(\mathbb{P}, \mathsf{ch})$. A ***run*** is a description of a (possibly infinite) execution of the system. We call a set of runs a ***system***. We will be interested in systems of the form $R_Q = R(Q, \gamma^f)$ consisting of all runs of a given protocol Q in which no more than f processes fail. A failure pattern determines who fails in the run, and what messages it succeeds in sending when it fails. Formally, we define:

▶ **Definition 1** (Failure patterns). *A* failure pattern *for a model γ^f is a set*

$$FP \triangleq \{\langle q_1, t_1, Bl(q_1)\rangle, \dots \langle q_k, t_k, Bl(q_k)\rangle\}$$

of $k \leq f$ triples, where $q_i \in \mathbb{P}$, $t_i \geq 0$, and $Bl(q_i) \subseteq \mathbb{P}$ for $i = 1, \dots, k$. We consider a run r to be compatible with *a failure pattern $FP = \{\langle q_1, t_1, Bl(q_1)\rangle, \dots \langle q_k, t_k, Bl(q_k)\rangle\}$ if*

1. *each process q_i fails in r at time t_i and only the processes $q_1, \dots, q_k$ fail in r, while*
2. *for every process p to whom q_i should send a message in round $t_i + 1$ according to Q (based on q_i's local state at time t_i in r), a message from q_i to p is sent at time t_i in r iff $p \notin Bl(q_i)$.*

If a process q_i is specified as failing in a pattern FP, then for every $p \in Bl(q_i)$, we consider the channel $\mathsf{ch}_{q_i,p}$ to be *blocked* from round $t_i + 1$ on.

For ease of exposition in this paper we will restrict our attention to the case in which the source s has a binary initial value $v_s \in \{0, 1\}$, while the initial values of all other processes $p \neq s$ are fixed. Thus, there are only two distinct initial global states in a system R_Q. Moreover, the deterministic protocol Q, a given initial global state (in our case the value of v_s) and a failure pattern uniquely determine a run. Relaxing these assumptions would not modify our results in a significant way; it would only make proofs quite a bit more cumbersome.

2.1 Defining Knowledge

Our analysis makes use of a formal theory of knowledge in distributed systems. We sketch the theory here; see [6] for more details and a general introduction to the topic. In general, a process i can be in the same local state in different runs of the same protocol. We shall say that two runs r and r' are *indistinguishable* to process i at time m if $r_i(m) = r'_i(m)$. The current time m is represented in the local state $r_i(m)$, and so, $r_i(m) = r'_i(m')$ can hold only if $m = m'$. Notice that since we assume that processes follow deterministic protocols, if $r_i(m) = r'_i(m)$ then process i is guaranteed to perform the same actions at time m in both r and r' if it is active at time m.

▶ **Definition 2** (Knowledge). *Fix a system R, a run $r \in R$, a process i and a fact φ. We say that $K_i\varphi$ (which we read as "process i* ***knows*** *φ") holds at time m in r iff φ is true at time m at all runs $r' \in R$ such that $r_i(m) = r'_i(m)$.*

Definition 2 immediately implies the so-called *Knowledge property*: If $K_i\varphi$ holds at time m in r, then so does φ. The logical notation for "the fact φ holds at time m in the run r with respect to the system R" is $(R, r, m) \vDash \varphi$. Often, the system is clear from context and is not stated explicitly. In this paper, the system will typically consist of all the runs of a given protocol Q in the current model of computation, which we denote by R_Q. Observe that knowledge can change over time. Thus, for example, $K_j(v_i = 1)$ may be false at time m in a run r and true at time $m + 1$, based perhaps on messages that j does or does not receive in round $m + 1$.

An essential connection between knowledge and action in distributed protocols, called the ***knowledge of preconditions principle*** (KoP), is provided in [16]. It states that whatever must be true whenever a particular action is performed by a process i must be known by i when the action is performed. More formally, we say that a fact φ is a ***necessary condition*** for an action α in a system R if for all runs $r \in R$ and times m, if α is performed at time m in r then φ must be true at time m in r. In our model the KoP can be stated as follows:

▶ **Theorem 3** (KoP [16]). *Fix a protocol Q for γ^f and let α be an action of process i in R_Q. If φ is a necessary condition for α in R_Q then $K_i\varphi$ is a necessary condition for α in R_Q.*

As observed in [2], in synchronous systems the passage of time can provide a process information about events at remote sites. (E.g., p can know that q performs an action at some specific time, based purely on the protocol.) In order to focus on genuine flow of information between processes, we make the following definition:

▶ **Definition 4.** Information transfer *(*IT*) between s and d is achieved when $K_d(v_s = b)$ holds, for some value $b \in \{0, 1\}$.*

Since the initial value v_s is independent of the protocol, for d to learn this value requires genuine flow of information from s to d.

2.2 Null Messages and Enhanced Message Chains

As discussed in the Introduction, if no message is ever sent over a given channel at time t under the protocol Q, then the absence of such a message in a given execution is not informative. We now define not sending to be a null message only if it is informative:

▶ **Definition 5.** *Let r be a run of some protocol Q. Process* ***i* sends *j* a null message** *at (r, t) if*

- $\mathsf{ch}_{i,j}$ *is not blocked at (r, t),*
- *i does not send an actual message over $\mathsf{ch}_{i,j}$ at (r, t), and*
- *there is a run r' of Q in which i sends an actual message over $\mathsf{ch}_{i,j}$ at (r', t).*

We can now generalize message chains to allow for null messages as well as actual ones:

▶ **Notation 1.** *We denote by $\theta = \langle p, t\rangle$ the* **process-time** *pair consisting of a process p and time t. Such a pair is used to refer to the point at time t on p's timeline.*

▶ **Definition 6.** *Let r be a run of a protocol Q. We say that there is* **an enhanced message chain** *from $\theta = \langle p, t\rangle$ to $\theta' = \langle q, t'\rangle$ in r, and write $\boldsymbol{\theta \rightsquigarrow_{Q,r} \theta'}$, if there exist processes $p = i_1, i_2 \ldots, i_k = q$ and times $t \le t_1 < t_2 < \cdots < t_k = t'$ such that for all $1 \le h < k$ process i_h sends either an actual message or a null message to i_{h+1} at (r, t_h). (We omit the subscript and write simply $\boldsymbol{\theta \rightsquigarrow \theta'}$ when Q and r are clear from the context.)*

Observe that Figure 4 contains three enhanced message chains between process s and d. Two of them contain a single actual message each, and one does not contain any actual message.

We are now ready to show that information transfer in synchronous systems requires the existence of an enhanced message chain.

▶ **Theorem 7.** *Let $f \geq 0$ and let Q be a protocol and $r \in R(Q, \gamma^f)$. Then $K_d(v_s = 1)$ holds at (r, m) only if $\langle s, 0\rangle \rightsquigarrow \langle d, m\rangle$ in r.*

Proof. Assume, by way of contradiction, that $K_d(v_s = 1)$ at (r, m), and that $\langle s, 0\rangle \not\rightsquigarrow \langle d, m\rangle$ in r. By Definition 2 it suffices to show a run $r' \in R(Q, \gamma^f)$ in which $v_s \neq 1$ such that $r_d(m) = r'_d(m)$. Denote

$$T \triangleq \{\theta \in \mathbb{V} \colon \langle s, 0\rangle \not\rightsquigarrow \theta \text{ in } r\}.$$

I.e., T is the set of nodes to which there is no enhanced message chain from $\langle s, 0\rangle$ in the run r. Observe that, by assumption, $\langle d, m\rangle \in T$. We construct a run r' as follows: The initial global state $r'(0)$ differs from $r(0)$ only in the value of the variable v_s (thus, $v_s = 0$ in r'), which appears in s's local state. All other initial local states are the same in $r'(0)$ and in $r(0)$. Finally, all processes have the same failure patterns in both runs. We now prove by induction on time t that $r_i(t) = r'_i(t)$ holds for all nodes $\langle i, t\rangle \in T$.

Base: $t = 0$. Assume that $\langle i, 0\rangle \in T$. By definition of T, it follows that $i \neq s$, and by construction of r' we immediately have that $r_i(0) = r'_i(0)$, as required.

Step. Let $t > 0$ and assume that the claim holds for all nodes $\langle j, t'\rangle$ with $t' < t$. Fix a node $\langle i, t\rangle \in T$. Clearly, $\langle i, t-1\rangle \in T$, and so by the inductive hypothesis $r_i(t-1) = r'_i(t-1)$. To establish our claim regarding $\langle i, t\rangle$, it suffices to show that i receives exactly the same messages at time t in both runs. Recall that the synchrony of the model implies that the only messages that i can receive at time t are ones sent at time $t-1$. Hence, we reason by cases, showing that every process $z \neq i$ sends i the same messages at time $t-1$ in both runs.

- Suppose that $\langle z, t-1\rangle \in T$. Then by the inductive assumption $r_z(t-1) = r'_z(t-1)$, i.e., process z has the same local state at time $t-1$ in both runs. Since Q is deterministic and since the runs r and r' have identical failure patterns, z sends i a message in r at time $t-1$ in r' iff it does so in r. Moreover, if it sends a message, it sends the same message in both cases.
- Suppose that $\langle z, t-1\rangle \notin T$, i.e., there is an enhanced message chain from $\langle s, 0\rangle$ to $\langle z, t-1\rangle$ in r. Since $\langle i, t\rangle \in T$ we have that z does not send a message to i at time $t-1$ in r. Assume by way of contradiction that i receives such a message in r'. In particular, this implies that the channel $\mathsf{ch}_{z,i}$ is not blocked in r', and since r and r' have the same failure pattern, $\mathsf{ch}_{z,i}$ is not blocked in r. Hence, by definition, there is a null message from $\langle z, t-1\rangle$ to $\langle i, t\rangle$ in r. This contradicts the fact that, by assumption, $\langle i, t\rangle \in T$. It follows that, in both r and r', process i does not receive any message from z at time t.

Since $r_i(t-1) = r'_i(t-1)$ process i performs the same actions at time $t-1$ in both runs. Since, in addition, i receives exactly the same messages at (r', t) as it does in (r, t) as we have shown, it follows that $r_i(t) = r'_i(t)$.

The inductive argument above showed that, for all processes i and all times $t \leq m$, if i is has not failed by time t and $\langle i, t\rangle \in T$, then $r_i(t) = r'_i(t)$. Since $\langle d, m\rangle \in T$ by assumption, it follows that, in particular, $r_d(m) = r'_d(m)$. Since $v_s \neq 1$ in r' we obtain that $\neg K_d(v_s = 1)$ at time m in r by Definition 2. This contradicts the assumption that $K_d(v_s = 1)$ holds at time m in r, completing the proof. ◀

Theorem 7 establishes that enhanced message chains are necessary for information transfer for all values of $f \geq 0$. In fact, when $f = 0$, enhanced message chains are also sufficient. (We omit a proof of this particular claim since it will follow from the more general Theorem 16). This demonstrates that enhanced message chains play an analogous role in reliable synchronous settings to the one that standard message chains play in asynchronous systems (cf. [4]).

In reliable systems, null messages are detected as reliably as actual messages are. As discussed in the Introduction, however, this is no longer true in the presence of failures. The knowledge formalism allows us to crisply capture a stronger requirement than the one in Theorem 7, which lies at the heart of the issue.

▶ **Theorem 8.** *Let $f \geq 0$, let r be a run of $R_Q = R(Q, \gamma^f)$ and denote $\theta_s = \langle s, 0\rangle$ and $\theta_d = \langle d, m\rangle$. Then $K_d(v_s = 1)$ holds at (r, m) only if $K_d(\theta_s \rightsquigarrow \theta_d)$ holds at (r, m).*

Proof. Suppose that $(R_Q, r, m) \vDash K_d(v_s = 1)$. Definition 2 implies that $(R_Q, r', m) \vDash K_d(v_s = 1)$ holds for every run r' such that $r'_d(m) = r_d(m)$. By Theorem 7 it follows that $(R_Q, r', m) \vDash (\theta_s \rightsquigarrow \theta_d)$ for every such r', and so by Definition 2 we obtain that $(R_Q, r, m) \vDash K_d(\theta_s \rightsquigarrow \theta_d)$, as claimed. ◀

3 Dealing with Failures

The need to know that an enhanced chain has reached d, established in Theorem 8, is not the same as the mere existence of such a chain. This difference matters when processes may fail, because then silence can be ambiguous, and null messages can be confused with process crashes. How, then, can d come to know that an enhanced chain has reached it, in a setting where $f > 0$ processes can crash? One possibility would be to have the protocol construct $f + 1$ enhanced chains from $\langle s, 0\rangle$ to $\langle d, m\rangle$ whose sets of participating processes are pairwise disjoint.[1] While such an assumption may be needed in a protocol that in a precise sense guarantees information transfer (we revisit this point in Section 6), in many instances the destination process can learn the sender's value even if the protocol does not employ such a scheme.

Our purpose is to investigate the communication patterns under which d can learn the value of v_s. For this purpose, we will find it convenient to associate a "communication graph" with every run, which we define as follows. The nodes of the graph are process-time pairs. Edges correspond to messages sent among processes, to null messages, and a local tick of the clock at a process. More formally:

▶ **Definition 9** (Communication Graphs). *The* communication graph *of a run r of protocol Q is $\mathsf{CG}_Q(r) \triangleq (\mathbb{V}, E)$, with nodes $\mathbb{V} = \mathbb{P} \times \mathbb{N}$ and edges $E = E_{\mathsf{l}} \cup E_{\mathsf{a}}(r) \cup E_{\mathsf{n}}(r)$, where*

- $E_{\mathsf{l}} = \{(\langle i, t\rangle, \langle i, t + 1\rangle) : i \in \mathbb{P}, t \in \mathbb{N}\}$,
- $E_{\mathsf{a}}(r) = \{(\langle i, t\rangle, \langle j, t + 1\rangle) : i$ *sends an actual message to* j *at time* t *in* $r\}$,
- $E_{\mathsf{n}}(r) = \{(\langle i, t\rangle, \langle j, t + 1\rangle) : i$ *sends a null message to* j *at time* t *in* $r\}$

Notice that both the set of nodes $\mathbb{V}$ and the set E_{l} of local edges are the same in all communication graphs. Observe that the communication graph directly represents enhanced message chains: $\boldsymbol{\theta \rightsquigarrow_{Q,r} \theta'}$ holds if and only if $\mathsf{CG}_Q(r)$ contains a path from θ to θ'.

[1] A similar issue arises in the Byzantine Agreement literature (cf. [5]) where many process-disjoint chains are used to overcome the possibility of failures.

3.1 Resilient Message Blocks

The Silent Choir Theorem of [7] states that a necessary condition for $K_d(v_s = 1)$ to hold at time m without an actual chain from s to d, is for there to be actual message chains to $f + 1$ members of the silent choir, after which they are all silent to d at time $m - 1$. As discussed in the Introduction, however, these members need not send d a *null message* at $m - 1$. Indeed, members of the "choir" need not even be neighbors of d. In this section we will present a strictly stronger condition on the communication pattern than the one in their theorem, called a resilient message block. The new condition will also be more informative as it is explicitly formulated in terms of null messages. Moreover, for the interesting case of optimizing communication for failure-free executions, our resilient message blocks will be both necessary *and* sufficient for information transfer.

Very roughly speaking, a process that knows about failures might detect the existence of an enhanced chain more easily than one who is unaware of failures. E.g., if d has detected all f faulty processes, then it can readily detect null messages sent by correct processes. Correctly coping with such issues requires a somewhat subtle definition and theorem statement.

► **Notation 2** (B null free paths). *Fix a protocol Q, let r be a run of Q, and let B be a set of processes. A path π in $\mathsf{CG}_Q(r)$ that does not contain null messages sent by processes in the set B is called B* null free *(we write that "π is $B_{\not\eta}$" for brevity).*

A $B_{\not\eta}$ path can contain null messages, but not ones "sent" by members of B. Roughly speaking, if the members of B crash, this path can remain a legal enhanced message chain. In light of Theorem 8, we are now ready to characterize the properties of communication graphs of protocols that enable information transfer. Using $B_{\not\eta}$ paths we can now define the central communication patterns that will play a role in our analysis:

► **Definition 10** ((f/failed)-resilient message block). *Let r be a run of a protocol Q and denote by $\mathbb{F}^r$ the set of processes that fail in r. Let $\theta, \theta' \in \mathbb{P} \times \mathbb{N}$ be two nodes. An (f/failed)*-resilient message block *from θ to θ' in $\mathsf{CG}_Q(r)$ is a set Γ of paths between θ and θ' such that for every set of processes B such that $|B \cup \mathbb{F}^r| \le f$, there is a $B_{\not\eta}$ path in $\mathsf{CG}_Q(r)$ from θ to θ' in Γ.*

Recall that an actual message chain contains no null messages, and is thus a $B_{\not\eta}$ path for every set B of processes. As a result, an actual message chain between two nodes is, in particular, an f/failed-resilient message block, for all runs and *all* values of $f \ge 0$.

Our next theorem states that in order for a process d to know in some run r at time m that there is an enhanced message chain from some node to itself, there must be a resilient message block between them. Roughly speaking, the claim is proved by way of contradiction. We assume a set B of processes contradicting the assumption and construct a run r' that is indistinguishable to d from r in which nodes involving processes of B cut all paths from $\theta_s = \langle s, 0\rangle$ to $\theta_d = \langle d, m\rangle$. I.e., there is no enhanced message chain from θ_s to θ_d in r'. More precisely, given a set B of processes we will define the set T_B to be the set of nodes to which there is *no* $B_{\not\eta}$ path from θ_s. We give an example of such a set in Figure 5. The highlighted nodes are in T_B while the others are not in T_B.[2] As detailed in the complete proof, when constructing r', we make the processes of B fail at times that make these failures unnoticeable by processes appearing in T_B (and hence by the contradiction assumption neither by d at

[2] For the sake of clarity we do not draw all of the nodes and edges of the communication graph. Thus, for example, we do not represent all the nodes along local time lines and the local edges in E_l that connect them.

time m). As a result, process d does not know at (r, m) that an enhanced message chain has reached it. Since by Theorem 7, this is a necessary condition, we conclude that $\neg K_d(v_s = 1)$ at (r, m), as claimed. We can now show:

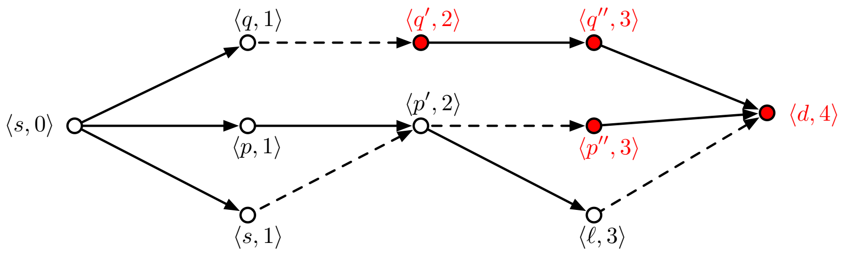

Figure 5 A communication graph and its corresponding set T_B (highlighted) for $B = \{q, p', \ell\}$.

▶ **Theorem 11.** *Let r be a run of a given protocol Q and let $\mathbb{F}^r$ denote the set of faulty processes in r. If $K_d(v_s = 1)$ holds at (r, m), there is an (f/failed)-resilient message block from $\theta_s \triangleq \langle s, 0\rangle$ to $\theta_d \triangleq \langle d, m\rangle$ in $\mathsf{CG}_Q(r)$.*

The complete proof appears in the Appendix.[3] We point out that Theorem 11 is stronger than the Silent Choir Theorem of [7]. Namely, as we claim in Lemma 12 the existence of the depicted resilient message block implies that a silent choir exists. However, the converse is not true. Remember the example of Figure 1. The set of processes $\{s, p\}$ is a silent choir for $f = 1$ but clearly, there is no f-resilient message block – take for instance $B = \{p\}$, there is no $B_{\not{d}}$ path from $\langle s, 0\rangle$ to $\langle d, 2\rangle$.

As the following lemma establishes, Theorem 11 implies the Silent Choir Theorem:

▶ **Lemma 12.** *If there is an f/failed-resilient message block from θ to θ' in $CG_Q(r)$, then there exists a silent choir from θ to θ' in r.*

We are now ready to characterize the communication structures needed to solve information transfer and coordination in several interesting cases.

4 Application: Information Transfer in Nice Runs

Theorem 11 is at the heart of information transfer in fault-prone synchronous systems. f-resilient message blocks constitute a necessary pattern for information transfer in our model since without them, one cannot know that an enhanced message chain reaches it. Clearly, in a system prone to crash failures, it is natural to require for information to be conveyed in failure-free runs. Focusing on the design of protocols that are optimized for the common case has a long tradition in distributed computing and can be useful in applications including Consensus, Atomic Commitment, and blockchain protocols (see, e.g., [1, 10, 15, 11, 14, 8]).

Clearly, the information that a null message conveys depends crucially on the protocol. More precisely, it depends on the conditions under which i would not send an *actual* message. To account for this, we make the following definition:

[3] Every claim is completely proved either in the main text,in the Appendix or in the full version of the paper [17].

▶ **Definition 13.** *We say that process i* **sends a null message** *over* $\mathsf{ch}_{i,j}$ *at time t* **in case $\boldsymbol{\varphi}$ in** *a given protocol* $\boldsymbol{Q}$ *if for every run $r \in R_Q$ in which* $\mathsf{ch}_{i,j}$ *is not blocked at (r,t), process i sends a null message over* $\mathsf{ch}_{i,j}$ *at (r,t) iff $(R_Q, r, t) \models \varphi$.*

Roughly speaking, a process p "sends a null message in case φ" at time t if whenever it is active at time t and φ does not hold, then p sends an actual message. If φ does hold at time t, then p keeps silent. By definition, a null message is sent in case φ only if φ is true. Therefore, in a reliable system (i.e., when $f = 0$), sending a null message in case φ informs the recipient that φ holds.

We focus on solving the IT problem in *nice runs*, which are formally defined as follows:

▶ **Definition 14.** *Let Q be a protocol. The run of Q in which $v_s = 1$ and no process fails is called* $\boldsymbol{Q}$*'s* **nice run**. *We denote Q's nice run by* $\boldsymbol{\hat{r}(Q)}$ *and the communication graph of $\hat{r}(Q)$ by* $\mathsf{nG}(\boldsymbol{Q})$. *When Q is clear from the context, we simply write* $\boldsymbol{\hat{r}}$.

Every protocol Q has a unique nice run. We show in this section that the conditions of Theorem 11 are also sufficient for information transfer in *nice* runs. Clearly, in a failure-free execution r, it holds that $\mathbb{F}^r = \emptyset$. An f/failed-resilient message block with $\mathbb{F}^r = \emptyset$ is a central structure for information transfer in nice runs and will be used in Section 5. It can be defined in slightly simpler terms as follows:

▶ **Definition 15** (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 $\mathsf{CG}_Q(r)$ *is a set Γ of paths between θ and θ' such that for every set of processes B of size $|B| \le f$, there is a $B_{\not\in}$ path in $\mathsf{CG}_Q(r)$ from θ to θ' in Γ.*

Observe that the presence of an f-resilient message block in the communication graph of a run r suffices to ensure that in any run r' that "looks the same" to d at (r', m) i.e., $r_d(m) = r'_d(m)$, there is an enhanced message chain reaching d.

▶ **Theorem 16** (Nice-run IT). *f-resilient message blocks are necessary and sufficient for solving* IT *in the nice run. Namely,*

- *(Necessity) If $K_d(v_s = 1)$ at $(\hat{r}, m)$ then there is an f-resilient message block from θ_s to θ_d in* nG.
- *(Sufficiency) If a communication graph* CG *contains an f-resilient message block between θ_s and θ_d, then there exists a protocol Q such that $\mathsf{nG}(Q) = \mathsf{CG}$ that solves* IT *between θ_s and θ_d.*

Theorem 16 gives a *tight* characterization of the communication patterns needed to solve IT in nice runs: Every solution must construct an f-resilient message block, and for every f-resilient message block, there exists an IT protocol that uses only the paths in this block in its nice run.

The necessity part of Theorem 16 results from Theorem 11 applied to $\hat{r}$ where the set of faulty processes is $\mathbb{F}^r = \emptyset$. To show sufficiency, we need to describe a protocol Q as claimed. Before sketching the proof, we discuss and define a class of protocols used in the proof.

In protocols solving IT in the nice run, messages only need to convey whether the sender has detected that the run is not nice. To consider this more formally, for a given protocol Q we denote by ψ_{nice} the fact "the current run is $\hat{r}$." Typically, if $f > 0$, it is impossible for a process to know that ψ_{nice} is true: Even if a process has observed no failures, or indeed, even if no failures have occurred by a given time t, there may be run indistinguishable to the process in which one or more processes fail after time t. Nevertheless, a process may readily know that $\neg\psi_{nice}$, if it knows of a failure or detects that $v_s = 0$. Of course, because of the Knowledge property, in the nice run $\hat{r}$ itself, no process will ever know $\neg\psi_{nice}$.

▶ **Definition 17** (Nice-based Message Protocols). *A protocol Q is a Nice-based Message protocol (*NbM *protocol) if (i) All actual messages sent are single bit messages and whenever a process p sends an actual message, it sends a '0' if $K_p\neg\psi_{nice}$ and sends a '1' otherwise (i.e., if $\neg K_p\neg\psi_{nice}$) and (ii) for all processes p, each null message sent by p over any channel is a null message in case $\neg K_p\neg\psi_{nice}$.*

We remark that a process p can efficiently check whether $K_p(\neg\psi_{nice})$ by simply comparing p's local state at (r,t) to its local state at $(\hat{r},t)$. If the two states are identical, then the predicate $K_p(\neg\psi_{nice})$ is false. Otherwise, it is true.

Proof sketch (of sufficiency in Theorem 16). Suppose that CG contains an f-resilient message block between θ_s and θ_d. The desired protocol Q is defined to be a Nice-based Message protocol such that $\mathsf{nG}(Q)=\mathsf{CG}$. I.e., for every edge $e\triangleq(\langle u,t\rangle,\langle v,t+1\rangle)$ in CG: If $e\in E_{\mathsf{n}}$, then u keeps silent if $\neg K(\neg\psi_{nice})$ and should send a '0' to v otherwise. If $e\in E_{\mathsf{a}}$, then u should send '1' to v if $\neg K_u(\neg\psi_{nice})$ and '0' otherwise. We show that the assumptions guarantee that for every run r of Q there is (at least one) path in $\mathsf{nG}(Q)$ from θ_s to θ_d along which no silent process fails in r. We show by induction on time that in every run r in which $v_s=0$, for each node $\langle p,t\rangle$ along this path, p knows at time (r,t) that the run is not nice. Hence, d also knows at (r,m) that the run is not nice. Since ψ_{nice} holds throughout $\hat{r}$, so does $\neg K_d\neg\psi_{nice}$. It follows that $K_d(v_s=1)$ at $(\hat{r},m)$, as claimed. ◀

5 Application: Coordination

f-resilient message blocks and Nice-run IT are useful tools for solving more complex problems. We now show how they can be used to characterize solutions to the Ordered Response (O-R) coordination problem. This problem was originally defined in [3], and it requires a sequence of actions to be performed in linear temporal order, in response to a triggering signal from the environment.[4] For simplicity, we identify the signal to be received iff $v_s=1$. We assume that each process $i_h\in\{i_1,i_2,\ldots,i_k\}$ has a specific action a_h to perform, and that the actions should be performed in order, provided that initially $v_s=1$.

▶ **Definition 18** (Ordered Response). *We say that a protocol Q is* ***consistent*** *with the instance* OR $=\langle v_s=1,a_1,\ldots,a_k\rangle$ *of the* Ordered Response (O-R) *problem if it guarantees that a_h is performed in a run only if $v_s=1$ and $a_1,\ldots,a_{h-1}$ are performed. In particular, if both a_{h-1} and a_h are performed at times t_{h-1} and t_h respectively, then $t_{h-1}\le t_h$. Protocol Q* ***solves*** *this instance* OR *if, in addition, all of the actions a_h are performed in Q's nice run.*

Let us denote by $\underline{a}_h$ the fact that the action a_h has (already) been performed. Since, by definition of O-R, both $v_s=1$ and $\underline{a}_{h-1}$ are necessary conditions for performing a_h, the Knowledge of Preconditions principle (Theorem 3) implies that these facts must be known when $\underline{a}_h$ is performed:

▶ **Lemma 19.** *Suppose that Q solves the instance* OR $=\langle v_s=1,a_1,\ldots,a_k\rangle$ *of* O-R. *For every run r of Q and action a_h performed (at time t_h) in r, we have*

1. $(R_Q,r,t_h)\vDash K_{i_h}(v_s=1)$, *and*
2. $(R_Q,r,t_h)\vDash K_{i_h}(\underline{a}_{h-1})$ *if $h>1$.*

[4] In [3] Ordered Response was studied in a reliable setting with no crashes, and upper bounds on message delivery times; a very different set of assumptions than here.

For a protocol Q solving an instance of Ordered Response, every action a_h, is performed at some specific time t_h in the nice run $\hat{r} = \hat{r}(Q)$. For ease of exposition we denote by by $\theta_h \triangleq \langle i_h, t_h \rangle$ the node of $\mathsf{nG}(Q)$ where the action is taken, and by $\theta_h^+ \triangleq \langle i_h, t_h + 1 \rangle$ the node of i_h one time step after the node θ_h.

We can use Lemma 19 to provide necessary conditions on the nice communication graph of protocols that solve Ordered Response. Lemma 19(1) implies that Q must perform Nice-run IT to θ_h, for all actions a_h. Lemma 19(2), in turn, implies that i_h needs to learn that a_{h-1} has been performed in order to perform its action. A straightforward way to do this is by performing Nice-run IT directly between consecutive actions, i.e., by creating an f-resilient block between θ_{h-1} and θ_h. While this is a possible solution, it is *not* the only way that i_h can obtain this knowledge. Process i_h can also learn about the previous action *indirectly*. This requires i_h to know that i_{h-1} couldn't have failed (since otherwise i_h would have an indication that it failed) and that it received the information needed to perform a_{h-1}. The possibility of acting on indirect information is where solving the Ordered Response problem goes beyond Nice-run IT. We can show the following:[5]

▶ **Theorem 20** (O-R Necessity). *Let Q be a protocol solving* $\mathsf{OR} = \langle v_s = 1, a_1, \ldots, a_k \rangle$ *for some sequence of times $t_1 \leq t_2 \leq \ldots \leq t_k$. Then* $\mathsf{nG}(Q)$ *must contain the following blocks:*

1. *An f-resilient message block between θ_s and θ_x, for each $x \leq k$; and*
2. *An $(f-1)$-resilient message block between θ_{x-1}^+ and θ_x that does not contain null messages sent by i_{x-1}, for each $1 < x \leq k$.*

Observe that Item 2 of Theorem 20 implies the existence of an $(f-1)$-resilient message block in which i_h does not send null messages. This requirement is weaker than the existence of an f-resilient message block. While the conditions for O-R stated in Theorem 20 are necessary, they are not sufficient in general. Indeed, it is unclear what conditions on $\mathsf{nG}(Q)$ might be sufficient to solve O-R for general protocols Q. In the Appendix, we present a natural class of protocols for which conditions that are both necessary and sufficient can be stated and proven (see Theorems 31 and 32).

6 Robust Information Transfer

We now turn to consider protocols that convey information in a "robust" way. Namely, they ensure that in every run in which $v_s = 1$ and the source process s does not fail, the destination eventually knows that $v_s = 1$.

▶ **Definition 21** (Robust Information Transfer). *A protocol Q is said to solve* Robust Information Transfer *between processes s and d if, for every run r of Q in which $v_s = 1$ and s does not fail, there is a time m such that $K_d(v_s = 1)$ holds at (r, m).*

Clearly, a protocol that solves the Robust Information Transfer problem also solves, in particular, IT in its nice run. However, Robust Information Transfer is a strictly harder problem and so, as shown in this section, its solutions require more communication than is allowed by f-resilient message blocks.

▶ **Theorem 22** (Robust IT Necessity). *Let Q be a protocol that solves Robust* IT *between s and d. Then, there exists $m \geq 0$ such that*

- $\mathsf{nG}(Q)$ *contains an actual message sent from s to d no later than at time $m-1$, or*
- $\mathsf{nG}(Q)$ *contains $f+1$ paths from $\theta_s = \langle s, 0 \rangle$ to $\theta_d = \langle d, m \rangle$ that are disjoint in message senders (except for s and d) such that in at least one of these paths, s does not send null messages.*

[5] Recall that all proofs for Sections 5 and 6 appear in [17].

Proof sketch. The claim is proved by way of contradiction assuming there exists no m as described in the Theorem. We then consider different cases according to the way the Theorem's assumptions are violated. For each case, we construct a run $r \in R_Q$ in which $v_s = 1$ and s does not fail as well as a corresponding run r' in which there is no enhanced message chain from θ_s to θ_d and that is indistinguishable by d. By Theorem 7, it results that $\neg K_d(v_s = 1)$ at (r, m), completing the proof. ◀

An interesting difference between the necessary conditions for solving the Nice-run IT and the ones for Robust IT is in that for the former, process disjointness is required only for the processes sending null messages, while for the latter it is required for all processes. This resembles the conditions of [5] where in order to solve Byzantine Agreement in the synchronous byzantine model, the communication network must have a connectivity of at least $2f + 1$. Thus, the paths from the source process s must be process disjoint the stronger sense.

We now show that the conditions of Theorem 22 are not only necessary, but also sufficient.

▶ **Theorem 23** (Robust IT Sufficiency). *Let* CG *be a communication graph satisfying the conditions of Theorem 22. Then there is a protocol Q with* $\mathsf{nG}(Q) = \mathsf{CG}$ *that solves the Robust* IT *problem.*

In analogy to how we defined Nice-based Message protocols in Section 4, we define in the Appendix (Definition 29) what we call Robust-based Message protocols – protocols in which whether messages are sent and the contents of the messages sent depend on whether a process knows that $(v_s = 0 \vee s \text{ failed})$. Taken together, Theorems 22 and 23 provide a tight characterization of the communication patterns of protocols solving Robust Information Transfer.

7 Conclusions

Every model of distributed computing provides particular means by which processes can communicate, and these can have a profound impact on the problems that can be solved in the model and on the form that protocols solving them will have. Synchronous systems with global clocks, for example, allow nontrivial use of null messages, which are completely meaningless in the asynchronous model, for example. Since a message not sent can be informative only if there are alternative conditions under which it would be sent, null messages are especially useful as a means of shifting communication costs to optimize for the common case. As illustrated in Figures 3 and 4 an demonstrated in the Atomic Commitment protocols of [7], shifting these costs in a careful way can result in significant savings.

By refining the definitions of null messages, we were able to investigate fundamental aspects of information transfer and coordination in synchronous systems with crash failures. In particular, we obtained characterizations of protocols that solve information transfer and coordination problems in nice, failure-free executions. A central tool in our analysis is the notion of an f-resilient message block, which is significantly more refined than the silent choirs of [7]. Indeed, while constructing silent choirs is a necessary condition on protocols solving information transfer in nice runs, constructing f-resilient blocks is both necessary *and* sufficient. For the Ordered Response coordination problem, where liveness needs to be guaranteed in nice runs, we obtain a condition based on resilient message blocks which, again, is both necessary and sufficient. No similar analysis of Ordered Response has been attempted in the literature.

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. On interactive knowledge with bounded communication. *Journal of Applied Non-Classical Logics*, 21(3-4):323–354, 2011. URL: `http://jancl.e-revues.com/article.jsp?articleId=17078`.

3 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.

4 K. M. Chandy and J. Misra. How processes learn. *Distributed Computing*, 1(1):40–52, 1986.

5 Danny Dolev. The byzantine generals strike again. *Journal of algorithms*, 3(1):14–30, 1982.

6 Ronald Fagin, Joseph Y Halpern, Yoram Moses, and Moshe Y Vardi. *Reasoning About Knowledge*. MIT Press, 1995. `doi:10.7551/mitpress/5803.001.0001`.

7 Guy Goren and Yoram Moses. Silence. *J. ACM*, 67:3:1–3:26, 2020. `doi:10.1145/3377883`.

8 Guy Goren and Yoram Moses. Optimistically tuning synchronous byzantine consensus: another win for null messages. *Distributed Comput.*, 34(5):395–410, 2021. `doi:10.1007/s00446-021-00393-8`.

9 Rachid Guerraoui and Jingjing Wang. How fast can a distributed transaction commit? In Emanuel Sallinger, Jan Van den Bussche, and Floris Geerts, editors, *Proceedings of the 36th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems, PODS 2017, Chicago, IL, USA, May 14-19, 2017*, pages 107–122. ACM, 2017. `doi:10.1145/3034786.3034799`.

10 Vassos Hadzilacos and Joseph Y. Halpern. Message-optimal protocols for byzantine agreement. *Mathematical Systems Theory*, 26(1):41–102, 1993.

11 Alex Kogan and Erez Petrank. A methodology for creating fast wait-free data structures. In *ACM SIGPLAN Notices*, volume 47, pages 141–150. ACM, 2012.

12 L. Lamport. Time, clocks, and the ordering of events in a distributed system. *Communications of the ACM*, 21(7):558–565, 1978.

13 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`.

14 Kfir Lev-Ari, Alexander Spiegelman, Idit Keidar, and Dahlia Malkhi. Fairledger: A fair blockchain protocol for financial institutions. In Pascal Felber, Roy Friedman, Seth Gilbert, and Avery Miller, editors, *23rd International Conference on Principles of Distributed Systems, OPODIS 2019, December 17-19, 2019, Neuchâtel, Switzerland*, volume 153 of *LIPIcs*, pages 4:1–4:17. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019. `doi:10.4230/LIPIcs.OPODIS.2019.4`.

15 Barbara Liskov. Practical uses of synchronized clocks in distributed systems. *Distributed Computing*, 6(4):211–219, 1993.

16 Yoram Moses. Relating knowledge and coordinated action: The knowledge of preconditions principle. In *Proceedings of TARK,*, pages 231–245, 2015. `doi:10.48550/arXiv.1606.07525`.

17 Raïssa Nataf, Guy Goren, and Yoram Moses. Null messages, information and coordination, 2023. `arXiv:2208.10866`.

Appendix

Additional details regarding Failure Patterns defined in Section 2

Our technical analysis is facilitated by defining a strictness ordering among failure patterns, and associating a minimal failure pattern with each run.

Notice that there may be several failure patterns that are compatible with a given run. In particular, blocking a channel along which no message should be sent does not affect processes' local states.

▶ **Definition 24** (Failure patterns comparison). *Let $FP = \{\langle q_i, t_i, Bl(q_i)\rangle\}_{i\le k}$ and $FP' = \{\langle q'_j, t'_j, Bl'(h'_j)\rangle\}_{j\le k'}$ be two failure patterns with $k, k' \le f$, and denote by F and F' the sets of processes that appear in these patterns, respectively. We say that FP' is* harsher *than FP (denoted by $FP' \le FP$) if $F \subseteq F'$ and for every $q_i \in F$:*
- *$t'_i < t_i$, or*
- *$t_i = t'_i$ and $Bl(q_i) \subseteq Bl'(q_i)$*

Note that processes that don't fail in r' might fail in r. We now define what we call a *minimal* failure pattern wrt. a run.

▶ **Definition 25.** *Let r be a run and let FP be a failure pattern compatible with r. We say that FP is minimal wrt. r if for every failure pattern FP' that is compatible with r and such that FP is harsher than FP' (i.e., $FP \le FP'$), it holds that $FP' = FP$.*

Clearly, for a given run r there is only one minimal compatible failure pattern. We denote it by $FP(r)$. In the proofs appearing in this section, we will be interested in comparing communication graphs of two runs of the same protocol. We compare the communication graphs regardless of the edges category. Formally:

▶ **Definition 26** (unlabeled-edge subgraph). *Let $\mathsf{CG} = (\mathbb{V}, E)$ and $\mathsf{CG}' = (\mathbb{V}, E')$ be two communication graphs. We say that CG is an "unlabeled-edge" subgraph of CG', and write $\mathsf{CG} \subseteq_u \mathsf{CG}'$ if for every edge $e \in E$ it is the case that $e \in E'$. (Although e can be an actual message edge in one graph and a null-message edge in the other.) In the rest of paper, we often write "subgraph" to stand in for "unlabeled-edge subgraph".*

We can now show:

▶ **Lemma 27.** *If $FP(r') \le FP(r)$ for two runs of a protocol Q, then $\mathsf{CG}_Q(r') \subseteq_u \mathsf{CG}_Q(r)$.*

Proof. Both graphs have the same set of nodes. We now prove that for each $e \in E_\mathsf{l} \cup E_\mathsf{a}(r') \cup E_\mathsf{n}(r')$, it holds that $e \in E_\mathsf{l} \cup E_\mathsf{a}(r) \cup E_\mathsf{n}(r)$.
- $E_\mathsf{l}(r) = E_\mathsf{l}(r')$.
- Let $e \triangleq (\langle p, t\rangle, \langle p', t+1\rangle) \in E_\mathsf{a}(r')$, i.e., p sends p' an actual message at (r', t). If p sends p' an actual message at (r, t) then $e \in E_\mathsf{a}(r)$. Otherwise, since $FP(r') \le FP(r)$, it holds that the channel $\mathsf{ch}_{p,p'}$ is not blocked at (r, t). Hence, by the definitions of communication graphs and of null messages, we have that $e \in E_\mathsf{n}(r)$. So $e \in E$.
- Finally, let $e \triangleq (\langle p, t\rangle, \langle p', t+1\rangle) \in E_\mathsf{n}(r')$. In particular, the channel $\mathsf{ch}_{p,p'}$ is not blocked at (r', t). Hence by null messages definition and the fact that $FP(r) \le FP(r')$ it holds that $e \in E_\mathsf{a}(r) \cup E_\mathsf{n}(r)$. ◀

Proof of Theorem 11

In the proof of Theorem 11, we will construct a run r' that is similar to r but in which we make fail additional processes. In order to ensure that these additional failures are not noticed by d (and hence to ensure that d does not distinguish r from r'), we make the processes fail at a specific *critical* time that we define as follows:

▶ **Definition 28** (Critical Time). *Let r be a run of Q, let B be a set of processes and let p be a process. For every pair θ and θ' of nodes of $\mathsf{CG}(Q)$, the* critical time *$t_p = t_p(\theta, \theta')$ wrt. $(\mathsf{CG}_Q(r), B)$ is defined to be the minimal time m_p such that $\mathsf{CG}_Q(r)$ contains a $B_{\not\gamma}$ path from θ to $\langle p, m_p\rangle$ as well as a path from $\langle p, m_p\rangle$ to θ'. If no such time m_p exists, then $t_p = \infty$.*

Informally, the critical time of a process p represents the first time at which p can learn about an event local to s and may be able to inform d about this event. Making relevant processes fail at their critical times ensures that d does not notice these failures and hence that d does not distinguish the constructed run r' from the nice run. We can now prove Theorem 11:

Proof. Assume by way of contradiction that no such block exists in $\mathsf{CG}_Q(r)$. I.e., there exists a set B such that $|B \cup \mathbb{F}^r| \leq f$ and every path from θ_s to θ_d in $\mathsf{CG}_Q(r)$ contains a null message from a process in B. Let B be a minimal set (by set inclusion) with this property. Define the set T_B to be:

$$T_B \triangleq \{\langle p, t\rangle \in \mathbb{V} : \text{There is no } B_{\not\emptyset} \text{ path from } \theta_s \text{ to } \langle p, t\rangle \text{ in } \mathsf{CG}_Q(r)\}$$

Notice that our assumption about θ_d implies that $\theta_d \in T_B$. Moreover, observe that if $\langle i, t\rangle \in T_B$, then $\langle i, t'\rangle \in T_B$ for all earlier times $0 \leq t' < t$.

We show that there exists a run r' of Q such that $r'_d(m) = r_d(m)$ and there is no enhanced message chain from θ_s to θ_d in r'. This will contradict the fact that $K_d(\theta_s \rightsquigarrow \theta_d)$ holds at time m in r. We construct r' as follows: The initial global state is $r'(0) = r(0)$. Each process $b \in B$ crashes in r' at its critical time $t_b \triangleq t_b(\theta_s, \theta_d)$ wrt. $(\mathsf{CG}_Q(r), B)$ without sending any messages from time t_b on. Moreover, every process in $\mathbb{F}^r \backslash B$ crashes in precisely the same manner in r' as it does in r. By definition, the critical time of a process $p \in \mathbb{F}^r$ is necessarily smaller or equal to the actual time at which p fails in r. We hence have that $FP(r') \leq FP(r)$. Clearly, there is no path from θ_s to θ_d in $\mathsf{CG}_Q(r')$. Notice that by minimality of B, each process $p \in B$ has a finite critical time $t_p = t_p(\theta_s, \theta_d)$ wrt. $(\mathsf{CG}_Q(r), B)$. We now prove by induction on t that for all $\langle i, t\rangle \in T_B$, if i has not crashed by time t in r', then $r_i(t) = r'_i(t)$.

Base: $t = 0$. By assumption, $r'(0) = r(0)$. Thus, $r'_i(0) = r_i(0)$ for every process i and in particular for those satisfying $\langle i, 0\rangle \in T_B$.

Step. Let $t > 0$ and assume that the claim holds for all nodes $\langle l, t'\rangle$ with $t' < t$. Fix a node $\langle i, t\rangle \in T_B$. Clearly, $\langle i, t-1\rangle \in T_B$, and so by the inductive hypothesis $r_i(t-1) = r'_i(t-1)$. To establish our claim regarding $\langle i, t\rangle$, it suffices to show that i receives exactly the same messages at time t in both runs. Since messages are delivered in one time step in our model, the only messages that i can receive at time t are ones sent at time $t - 1$. Hence, we reason by cases, showing that every process $z \neq i$ sends i the same messages at time $t - 1$ in both runs.

- Suppose that $\langle z, t - 1\rangle \in T_B$.
  - If $z \in \mathbb{F}^r \backslash B$ then it is active at time $t - 1$ in r' iff it is active at this time in r. We have by the inductive assumption that it has the same local state in r. Since Q is deterministic, z sends i a message at time $t - 1$ in r' iff it does so in r. Moreover, if it sends a message, it sends the same message in both cases.
  - We show that if $z \in B$, then the channel $\mathsf{ch}_{z,i}$ is not blocked at time $t-1$ in r'. Assume by way of contradiction that $\mathsf{ch}_{z,i}$ is blocked at time $t - 1$. This means that z has failed in r' by time $t - 1$. Since $z \in B$ and z has failed in r' by time $t - 1$, we have that $t_z \leq t - 1$. By definition of z's critical time t_z, there is a $B_{\not\emptyset}$ path π from θ_s to $\langle z, t_z\rangle$ in $\mathsf{CG}_Q(r)$. There also is a path in $\mathsf{CG}_Q(r)$ from $\langle z, t_z\rangle$ to $\langle z, t - 1\rangle$ consisting of locality edges. Together, these two paths form a $B_{\not\emptyset}$ path from θ_s to $\langle z, t - 1\rangle$ in $\mathsf{CG}_Q(r)$, contradicting the assumption that $\langle z, t - 1\rangle \in T_B$.

    - Assume that $z \in B$ and $\mathsf{ch}_{z,i}$ is not blocked at time $t-1$ in r'. Then, as in the previous case, the inductive assumption and the fact that Q is deterministic imply that exactly the same communication occurs between $\langle z, t-1\rangle$ and $\langle i, t\rangle$ in both runs.
    - Finally, assume that $z \notin B \cup \mathbb{F}^r$. Then z is active at time $t-1$ in both r' and r. We have by the inductive assumption that it has the same local state in r. Since Q is deterministic, z sends i a message at time $t-1$ in r' iff it does so in r. Moreover, if it sends a message, it sends the same message in both cases.
- Now suppose that $\langle z, t-1\rangle \notin T_B$, i.e., there is a $B_{\not\!\!\mathsf{f}}$ path from θ_s to $\langle z, t-1\rangle$ in $CG_Q(r)$. Since $\langle i, t\rangle \in T_B$ we have that z does not send a message to i at time $t-1$ in r. There is no edge from $\langle z, t-1\rangle$ to $\langle i, t\rangle$ in $\mathsf{CG}_Q(r)$. Since $FP(r') \leq FP(r)$, it holds by Lemma 27 that $\mathsf{CG}_Q(r') \subseteq_u \mathsf{CG}_Q(r)$ and hence, there is no edge from $\langle z, t-1\rangle$ to $\langle i, t\rangle$ in $\mathsf{CG}_Q(r')$ either. Meaning that i does not receive a message from z neither at (r, t) nor at (r', t).

Since $r_i(t-1) = r'_i(t-1)$ process i performs the same actions at time $t-1$ in both runs. Since, in addition, i receives exactly the same messages at time t in r' as it does in $\hat{r}$ as we have shown, it follows that $r_i(t) = r'_i(t)$.

The inductive argument above showed that, for all processes i and all times $t \leq m$, if i is active at time t and $\langle i, t\rangle \in T_B$, then $r_i(t) = r'_i(t)$. Since, $\theta_d \in T_B$ by assumption, it follows that, in particular, $r_d(m) = r'_d(m)$. Since there is no enhanced message chain from θ_s to θ_d in r', we obtain that $\neg K_d(\theta_s \rightsquigarrow \theta_d)$ at time m in r by the Definition 2 of the knowledge operator. This contradicts the assumption that $K_d(\theta_s \rightsquigarrow \theta_d)$ holds at time m in r, completing the proof. ◀

Proof of Lemma 12

Proof. Denote by $\mathbb{F}^r$ the set of faulty processes in r, and assume that the conditions of the Silent Choir Theorem do not hold. I.e., neither a silent choir nor an actual message chain from θ to θ' exist in r. Let $\theta' = \langle q, m\rangle$. Let S be the set of processes such that for each $p \in S$ there is an actual message chain from θ to $\langle p, m-1\rangle$. Since there is no silent choir, the following holds: $|S \cup \mathbb{F}^r| \leq f$.

Let $B \triangleq S \cup \mathbb{F}^r$ and let π be a path from θ to θ' in $CG_Q(r)$. Since there is no actual message chain from θ to θ' in $CG_Q(r)$ we get that there is an edge from a process of B in π that is in E_{n}, i.e., corresponds to a null message sent by a process in B. This holds for every path from θ to θ'. Hence there exists a set of processes B such that $|B \cup \mathbb{F}^r| \leq f$ and such that there is no $B_{\not\!\!\mathsf{f}}$ path from θ to θ' in $CG_Q(r)$, i.e., the conditions of Theorem 11 do not hold, completing the proof. ◀

Proof of Theorem 16

Proof. The assumptions guarantee that there will always be at least one path from θ_s to θ_d in CG along which no "silent" process fails. Let Q' be an NbM protocol such that $\mathsf{nG}(Q') = \mathsf{CG}$. We show by induction that in all runs in which $v_s = 0$ each process along the path will detect that the run is not nice. In particular, j will be able to distinguish the run from the nice one by time m. It follows that $K_d(v_s = 1)$ holds in $\hat{r}$ at time m.

Let r' be a run in which $v_s = 0$ and denote by B the set of processes that fail in this run. Clearly, $|B| \leq f$. Let π be a $B_{\not\!\!\mathsf{f}}$ path in $\mathsf{nG}(Q')$, which is guaranteed to exist by the assumption. Let r' be a run in which $v_s = 0$. We now prove by induction on time that for each node $\langle p, t\rangle$ in π it holds that $K_p(\neg\psi_{nice})$ holds at (r', t). .

Base: $t = 0$. In this case, $p = s$. Since v_s appears in s's local state and its value differs to its value in the nice run, $K_s(\neg\psi_{nice})$ holds at time 0.

Step: $t > 0$. We consider the nodes $\langle q, t-1\rangle$ and $\langle p, t\rangle$ in π. By the induction hypothesis $K_q(\neg\psi_{nice})$ holds at $t-1$ in r'. We now reason by cases according to the class of the edge $(\langle q, t-1\rangle, \langle p, t\rangle)$ in $\mathsf{nG}(Q')$.

- Case 1: $(\langle q, t-1\rangle, \langle p, t\rangle) \in E_{\mathsf{l}}$ then $p = q$ and since the fact $\neg\psi_{nice}$ is a stable property we have by the induction hypothesis that $K_p(\neg\psi_{nice})$ holds at (r', t).
- Case 2: $(\langle q, t-1\rangle, \langle p, t\rangle) \in E_{\mathsf{a}}(\hat{r})$:
  - Case 2a: in the run r' process q does not send p a message, then p detects that the run is not $\hat{r}$ (in which, by assumption, it would receive a message from q).
  - Case 2b: q does send a message to p in r' then, by the induction assumption and the fact that q sends 0 if $K_q(\neg\psi_{nice})$ it follows that p receives a different message in r' and in $\hat{r}$, and so $K_p(\neg\psi_{nice})$ holds at time t.
- Case 3: $(\langle q, t-1\rangle, \langle p, t\rangle) \in E_{\mathsf{n}}(\hat{r})$ we have by the choice of π that q does not fail in r' and by the induction assumption $K_q(\neg\psi_{nice})$ holds at time $t-1$. Recall that, by assumption, in Q' process q can send a null message only in case $\neg(K_q\neg\psi_{nice})$. Since, by the inductive assumption on time $t-1$ this is not the case, q must send p a '0'-message. Since such messages are never sent in $\hat{r}$, we again conclude that $K_p(\neg\psi_{nice})$ holds at time t in r', as desired.

We have shown that for all runs r' in which $v_s \neq 1$ it is the case that $r'_d(m) \neq \hat{r}_d(m)$. Consequently, $v_s = 1$ for all runs r such that $r_d(m) = \hat{r}_d(m)$ and so, by Definition 2, we obtain that $K_d(v_s = 1)$ holds at $(\hat{r}, m)$, as claimed. ◀

Definition of Robust-based Message protocols

▶ **Definition 29** (Robust-based Message protocols). *We say that Q is a Robust-based Message (*RbM*) protocol if*

- *All actual messages sent in Q are single-bit messages, and whenever a process p sends an actual message, it sends a '0' if $K_p(v_s \neq 1 \vee s$ is faulty$)$ and sends a '1' otherwise,*
- *for all processes p, each null message sent by p over any channel is a null message in case $\varphi = \neg K_p[(v_s \neq 1) \vee (s$ is faulty$)]$ and*
- *for every run r and process p it holds that if p sends q an actual message at $(\hat{r}, t)$, then if the channel $\mathsf{ch}_{p,q}$ is not blocked at (r, t), process p also sends q an actual message at (r, t).*

Ordered Response

▶ **Definition 30** (Conservative O-R protocols). *Let Q be a deterministic protocol that solves* $\mathsf{OR} = \langle v_s = 1, a_1, \ldots, a_k\rangle$. *We say that Q is conservative for* OR *if for every run r of Q and all $x \leq k$ the following is true: Process i_x performs a_x at t_x only if $\neg K_{i_x}(\neg\psi_{nice})$ holds at (r, t_x).*

In a conservative protocol, if a process i_x knows at $\theta_x = \langle i_x, t_x\rangle$ that a failure has occurred, then it is not allowed to perform its action. Concretely, suppose that process i_x is prevented from action at $\theta_x = \langle i_x, t_x\rangle$ because it observes there that a process b has failed at $\rho_b = \langle b, m_b\rangle$. Since by Theorem 20, only $(f-1)$-resilient message blocks are required between two consecutive processes in the OR instance, the failure of $f-1$ other processes might disconnect θ_x from a node θ_h, for an index $h > x$ in the instance of O-R being solved.

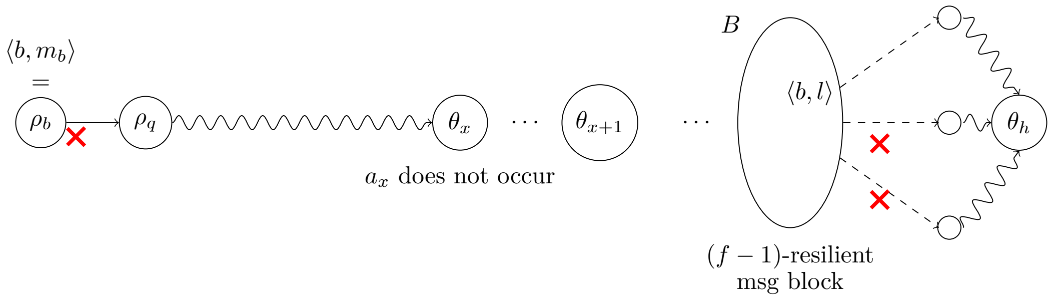

Figure 6 The problematic scenario that Theorem 31 solves. Squiggly arrows represent any kind of message chain or sets of message chains. Red crosses represent process failures. As in previous figures, dashed arrows represent null messages and full arrows represent real messages. If the depicted scenario occurs, then b's failure at ρ_b can cause i_x not to perform a_x. The protocol must therefore provide an $(f-1)$-resilient message block to θ_h from one of θ_x or $\langle b, m_b + 1 \rangle$.

Moreover, this might also disconnect ρ_b from θ_h. We would then obtain that i_h does not distinguish the current run from the nice run, resulting in a_h being performed. This is clearly a violation of O-R. We illustrate this scenario in Figure 6.

We can show that in order to prevent such a scenario there must be a $B_{\not\nu}$ path from θ_x or from b after its potential failing node ρ_b, to θ_h. This way, if b fails at ρ_b and prevents i_x from acting, then i_h will distinguish the current run from the nice run at θ_h. Acting conservatively, i_h will also refrain from acting, and thus avoid causing a violation of the O-R specification. Formally:[6]

▶ **Theorem 31.** *Let Q be a* conservative *protocol solving* $\mathsf{OR} = \langle v_s = 1, a_1, \ldots, a_k \rangle$.
For all nodes $\rho_b = \langle b, m_b \rangle$, indices $x < h \le k$ and sets $B \subseteq \mathbb{P}$, if

1. *there is a path π from ρ_b to θ_x in $\mathsf{nG}(Q)$ that starts with an edge $(\rho_b, \rho_q) \in E_{\mathsf{a}}$ and contains no edges corresponding to null message by b, and in addition*
2. *$b \in B$, $|B| \le f$ and there is no $B_{\not\nu}$ path from θ_x to θ_h in* CG,

then there is a $B_{\not\nu}$ path from $\langle b, m_b + 1 \rangle$ to θ_h in $\mathsf{nG}(Q)$.

In a precise sense, combining the conditions in this theorem with those of Theorem 20 we obtain a set of conditions that is not only necessary for conservative protocols Q (as already proved), but also sufficient. Indeed, as we now show, there exist protocols solving Ordered Response that satisfy precisely these conditions.

▶ **Theorem 32** (Sufficient conditions for O-R)**.** *The conditions stated in Theorem 20 and Theorem 31 are sufficient for solving an instance* $\mathsf{OR} = \langle v_s = 1, a_1, a_2, \ldots, a_k \rangle$ *of the Ordered Response problem. (For details, see [17].)*

Taken together, Theorems 20, 31, and 32 provide a characterization of the communication patterns that can solve Ordered Response using null messages. This characterization is tight for communication patterns of *conservative* protocols that solve O-R.

[6] The proofs of Theorems 31 and 32 appear in [17].