Basilic: Resilient-Optimal Consensus Protocols With Benign and Deceitful Faults

Alejandro Ranchal-Pedrosa†
Protocol Labs and University of Sydney
Sydney, Australia
alejandro.ranchalpedrosa@sydney.edu.au

Vincent Gramoli
University of Sydney and Redbelly Network
Sydney, Australia
vincent.gramoli@sydney.edu.au

***Abstract*—The problem of Byzantine consensus has been key to designing secure distributed systems. However, it is particularly difficult, mainly due to the presence of Byzantine processes that act arbitrarily and the unknown message delays in general networks. Although it is well known that both safety and liveness are at risk as soon as $n/3$ Byzantine processes fail, very few works attempted to characterize precisely the faults that produce safety violations from the faults that produce termination violations.**

In this paper, we present a new lower bound on the solvability of the consensus problem by distinguishing deceitful faults violating safety and benign faults violating termination from the more general Byzantine faults, in what we call the Byzantine-deceitful-benign fault model. We show that one cannot solve consensus if $n \leq 3t+d+2q$ with t Byzantine processes, d deceitful processes, and q benign processes.

In addition, we show that this bound is tight by presenting the Basilic class of consensus protocols that solve consensus when $n > 3t + d + 2q$. These protocols differ in the number of processes from which they wait to receive messages before progressing. Each of these protocols is thus better suited for some applications depending on the predominance of benign or deceitful faults.

***Index Terms*—agreement, consensus, fault tolerance, accountability, Byzantine-deceitful-benign,**

I. Introduction

The problem of Byzantine consensus has been key to designing secure distributed systems [1], [2], [3], [4], [5]. This problem is particularly difficult to solve because a Byzantine participant acts arbitrarily [6] and message delays are generally unpredictable [7]. Any consensus protocol would fail in this general setting if the number of Byzantine participants is $t \geq n/3$ [7], where n is the total number of participants. In some executions, $\lceil n/3 \rceil$ Byzantine participants can either prevent the termination of the consensus protocol by stopping or by sending unintelligible messages. In other executions, $\lceil n/3 \rceil$ can violate the agreement property of the consensus protocol by sending conflicting messages.

Interestingly, various research efforts were devoted to increase the fault tolerance of consensus protocols in closed networks (e.g., datacenters) by distinguishing the type of failures [2], [3], [4], [8]. Some works overcome the $t < n/3$ bound by tolerating a greater number of omission than commission faults [1], [2]. These works are naturally well-suited for closed networks where processes are protected from intrusions by a firewall: their processes are supposedly more likely to crash than to be corrupted by a malicious adversary. In this sense, these protocols favor tolerating a greater number of faults for liveness than for safety.

Unfortunately, fewer research efforts were devoted to explore the fault tolerance of consensus protocols in open networks (e.g., blockchains). In such settings, participants are likely to cause a disagreement if they can steal valuable assets. This is surprising given that attacks are commonplace in blockchain systems as illustrated by the recent losses of $\$70,000$[1] and $\$18$ million[2] in Bitcoin Gold, and of $\$5.6$ million in Ethereum Classic[3]. Comparatively, some blockchain participants, called miners, are typically monitored continuously so as to ensure they provide some rewards to their owners, hence making it less likely to prevent termination. To our knowledge, only alive-but-corrupt (abc) processes [9] characterize the processes that violate consensus safety. Unfortunately, abc processes are restricted to only try to cause a disagreement if the coalition size is sufficiently large to succeed at the attempt, which is impossible to predict in blockchain systems.

A. Our Results

In this paper, we present a new lower bound on the solvability of the Byzantine consensus problem by precisely exploring these two additional types of faults (that either prevent termination or agreement when $t \geq n/3$). Our lower bound states that there is no protocol solving consensus in the partially synchronous model [7] if $n \leq 3t + d + 2q$ with t Byzantine processes, d deceitful processes, and q benign processes. These different types of processes define the *Byzantine-deceitful-benign (BDB) failure model* and are characterized by the faults they commit. First, a *deceitful* process is a process that sends some conflicting messages (messages that contribute to a violation of agreement) during its execution. Second, a *benign* process is a faulty process that never sends any conflicting messages, contributing to non-termination. For example, a benign process can crash or send stale messages, or even equivocate as long as its messages have no effect on

† The author was with the University of Sydney, Sydney, Australia. He is now with Protocol Labs.

[1] https://news.bitcoin.com/bitcoin-gold-51-attacked-network-loses-70000-in-double-spends/
[2] https://news.bitcoin.com/bitcoin-gold-hacked-for-18-million/
[3] https://news.bitcoin.com/5-6-million-stolen-as-etc-team-finally-acknowledge-the-51-attack-on-network/

2023 IEEE 36th Computer Security Foundations Symposium (CSF) | 979-8-3503-2192-0/23/$31.00 ©2023 IEEE | DOI: 10.1109/CSF57540.2023.00002

DOI 10.1109/CSF57540.2023.00002

the agreement property. These two faults lie at the core of the consensus problem, as the property of validity can be locally checked for correctness, while termination and agreement can be violated in the presence of enough malicious processes. Compared to abc faults, we do not impose the restriction on deceitful processes to know whether their attack will succeed. Hence, while a protocol might tolerate $d < n/3$ abc faults along with $q < n/3$ benign faults, it would not necessarily tolerate $d < n/3$ deceitful faults along with $q < n/3$ benign faults. The contrary direction always holds.

Furthermore, we show that this lower bound is tight, in that we present the Basilic[4] class of protocols that solves consensus with $n > 3t + d + 2q$. Basilic builds upon recent advances in the context of accountability [10] by taking into account key messages only if they are cryptographically signed by their sender. If they are properly signed, the recipient stores these messages and progresses in the consensus protocol execution. Recipients also cross-check the messages they received with other recipients, based on the assumption that signatures cannot be forged. Once conflicting messages are detected, they constitute an undeniable proof of fraud to exclude the faulty sender before continuing the protocol execution. Thanks to this exclusion, Basilic satisfies a new property, *active accountability*, which guarantees that deceitful processes can not prevent termination.

Basilic is a class of consensus protocols, each parameterized by a different *voting threshold* or the number of distinct processes from which a process receives messages in order to progress. For a voting threshold of $h \in (n/2, n]$, Basilic satisfies termination if $h \leq n - q - t$, and agreement if $h > \frac{d+t+n}{2}$. This means that for just one threshold, say $h = 2n/3$, Basilic tolerates multiple combinations of faulty processes: it can tolerate $t < n/3$, $q = 0$ and $d = 0$; but also $t = 0$, $q < n/3$ and $d < n/3$; or even $t < n/6$, $q < n/6$ and $d < n/6$. This voting threshold can be modified by an application in order to tolerate any combination of t Byzantine, d deceitful and q benign processes satisfying $n > 3t + d + 2q$. The generalization of Basilic to any voting threshold h thus allows us to pick the best suited protocol depending on the application requirements. If, on the one hand, the application runs in a closed network (e.g., datacenter) dominated by benign processes, then the threshold will be lowered to ensure termination. If, on the other hand, the application runs in an open network (e.g., blockchain) dominated by deceitful processes, then the threshold will be raised to ensure agreement.

We illustrate in Figure 1 the new resilient-optimal bounds that Basilic tolerates if there are only deceitful and benign processes (i.e., for $t = 0$), compared to the classic Byzantine fault-tolerant (BFT) bound [7]. We prove that these bounds are resilient-optimal in the Byzantine-deceitful-benign failure model. We observe that compared to state-of-the-art accountable consensus protocols, Basilic satisfies active accountability and tolerates a greater number of faults, while maintaining the same time, message and bit complexities in synchronous periods.

[4]The name "Basilic" is inspired from the Basilic cannon that Ottomans used to break through the walls of Constantinople. Much like the cannon, our Basilic protocol provides a tool to break through the classical bounds of Byzantine fault tolerance.

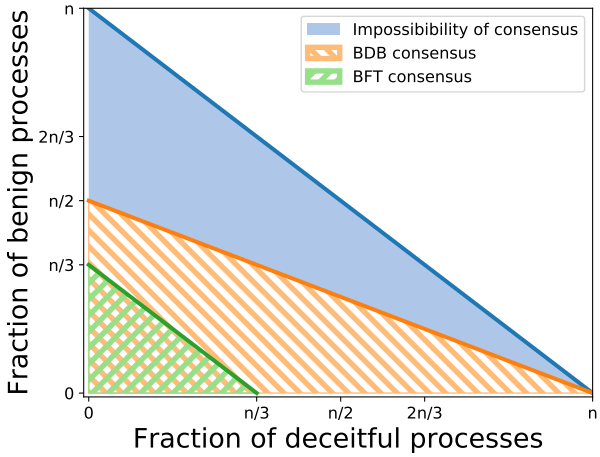

Fig. 1: The green area represents the bound for BFT consensus, where $t < n/3$ and thus the same for d, q, i.e., $d + q < n/3$. The orange area is the new fault tolerance in the Byzantine-deceitful-benign (BDB) failure model, where $d < n - 2q$ and $q < n/2$ (for $t = 0$). In blue, the area where it is impossible to solve consensus in the BDB model.

B. Roadmap

The rest of the paper is structured as follows. In Section II, we present the model and define the problem. In Section III, we present our impossibility result in the Byzantine-deceitful-benign model while we prove that Basilic solves the consensus problem in Section IV and analyze its complexities in Section V. Finally, we present the related work in Section VI, and we finally conclude in Section VII.

II. Model & Problem

We consider a committee as a set $N = \{p_0, ..., p_{n-1}\}$ of $|N| = n$ processes. These processes communicate in a partially synchronous network, meaning there is a known bound Δ on the communication delay that will hold after an unknown Global Stabilization Time (GST) [7]. Processes communicate through standard all-to-all reliable and authenticated communication channels [11], meaning that messages can not be duplicated, forged or lost, but they can be reordered.

a) Cryptography: We assume a public-key infrastructure (PKI) in that each party has a public key and a private key, and any party's public key is known to all [12]. As with other protocols that use this standard assumption [12], [13], we do not require the use of revocation lists (we will remove processes from the committee, but not their keys from the PKI). We refer to λ as the security parameter, i.e., the number of bits of the keys. As our claims and proofs require cryptography, they hold except with $\epsilon(\lambda)$ negligible probability [14]. We formalize negligible functions measured in the security parameter λ, which are those functions that decrease asymptotically faster than the inverse of any polynomial. Formally, a function $\epsilon(\kappa)$

is negligible if for all $c > 0$ there exists a κ_0 such that $\epsilon(\kappa) < 1/\kappa^c$ for all $\kappa > \kappa_0$ [14].

b) Consensus: A protocol executed by a committee of processes solves the consensus problem if the following three properties are satisfied by the protocol:

- **Termination.** Every non-faulty process eventually decides on a value.
- **Agreement.** No two non-faulty processes decide on different values.
- **Validity.** If all non-faulty processes propose the same value, no other value can be decided.

c) Conflicting messages: In order to detect faulty processes, these have to send distinct messages to different processes where they were expected to broadcast the same message to different processes [15], we refer to these messages as conflicting. Given a protocol σ, we say that a message, or set of messages, m sent by process p *conforms* to an execution σ_E of the protocol σ, if σ_E belongs to the set of all possible executions where p sent m and p is a non-faulty process. Also, a faulty process p sending two messages m, m' *contributes* to a disagreement if there is an execution σ_E of σ such that (i) sufficiently many faulty processes sending m, m' (and possibly more messages) to a disjoint subset of non-faulty processes, one to each, leads to a disagreement, and (ii) σ_E does not lead to a disagreement without p sending m, m'. Two messages m, m' are *conflicting* with respect to σ if:

1) m, m' individually conform to algorithm σ for some execution σ_E, $\sigma_{E'}$, respectively, $\sigma_E \neq \sigma_{E'}$,
2) there is no execution $\sigma_{E''}$ of σ such that both messages together conform to $\sigma_{E''}$, and
3) if p sending m, m' to a disjoint subset of non-faulty processes, one to each, contributes to a disagreement.

When combined in one message and signed by the sender, conflicting messages constitute a proof of a process being faulty with the purpose of causing a disagreement. We speak of this proof as a *proof-of-fraud* (PoF). An example of two conflicting messages is a faulty process sending two different proposals for the same round (the proposer should only propose one value per round).

Our definition of conflicting messages differs from previous similar concepts in that conflicting messages allow for any process p to verify if two messages are conflicting: a non-faulty process can always construct a PoF from two conflicting messages alone, but it cannot do so with all mutant messages [16], as p would need to also learn the entire execution, or with messages sent from an equivocating process [17], as these do not necessarily contribute to disagreeing.

d) Send, receive and deliver: Messages can be sent and received, but we also consider broadcast primitives that contain two functions: a broadcast function that allows process p_i to send messages through multiple channels accross the network, and a deliver function that is invoked at the very end of the broadcast primitive to indicate that the recipient of the message has received and processed the message. There could be however multiple message exchanges before the delivery can happen. When we specify the broadcast primitives, we attach the name of the protocol as a prefix to the broadcast and deliver function to refer to a message broadcast or delivered using that protocol, such as AARB-broadcast, AARB-deliver, ABV-broadcast and ABV-deliver, as we detail later in this paper.

e) Fault model: There are three mutually exclusive classes of faulty processes: Byzantine, deceitful and benign [18], in what we refer to as the *Byzantine-deceitful-benign* (BDB) failure model. Each faulty process belongs to only one of these classes. Byzantine, deceitful and benign processes are characterized by the faults they can commit. A fault is *deceitful* if it contributes to breaking agreement, in that it sends conflicting messages violating the protocol in order to lead two or more partitions of processes to a disagreement. We allow deceitful processes to constantly keep sending conflicting messages, even if they do not succeed at causing a disagreement, but instead their deceitful behavior prevents termination. As deceitful processes model processes that try to break agreement, we assume also that a deceitful fault does not send conflicting messages for rounds or phases of the protocol that it has already terminated at the time that it sends the messages. Deceitful processes can alternate between sending conflicting messages and following the protocol, but cannot deviate in any other way. A *benign* fault is any fault that does not ever send conflicting messages. Hence, benign faults cover only faults that can break termination, e.g. by crashing, sending stale messages, etc.

As usual, Byzantine processes can act arbitrarily. Thus, Byzantine processes can commit benign or deceitful faults, but they can also commit faults that are neither deceitful nor benign. A fault that sends conflicting messages and crashes afterwards is, by these definitions, neither benign nor deceitful. We denote t, d, and q as the number of Byzantine, deceitful, and benign processes, respectively. We assume that the adversary is static, in that the adversary can choose up to t Byzantine, d deceitful and q benign processes at the start of the protocol, known only to the adversary.

In order to distinguish benign (resp. deceitful) processes from Byzantine processes that commit a benign (resp. deceitful) fault during a particular execution of a protocol, we formalize fault tolerance in the BDB model. Let $E_\sigma(t, d, q)$ denote the set of all possible executions of a protocol σ given that there are up to t Byzantine, d deceitful and q benign processes. We say that a protocol σ for a particular problem P is (t, d, q)-*fault-tolerant* if σ solves P for all executions $\sigma_E \in E_\sigma(t, d, q)$. We abuse notation by speaking of a (t, d, q)-fault-tolerant protocol σ as a protocol that tolerates t, d and q Byzantine, deceitful and benign processes, respectively.

Note that, given a protocol σ, then $E_\sigma(0, d + k, q) \subset E_\sigma(k, d, q)$ by definition. Thus, if σ is (k, d, q)-fault-tolerant then σ is $(0, d + k, q)$-fault tolerant, and also $(0, d, q + k)$-fault-tolerant. However, the contrary is not necessarily true: a protocol σ that is $(0, d + k, q)$-fault-tolerant is not necessarily (k, d, q)-fault tolerant, as $E_\sigma(k, d, q) \nsubseteq E_\sigma(0, d + k, q)$, because Byzantine participants can commit more faults than

deceitful or benign. Finally, a process is *non-faulty* if it is neither Byzantine, nor deceitful, nor benign. Non-faulty processes follow the protocol.

Compared to commission and omission faults, notice that not all commission faults contribute to causing disagreements. For example, some commission faults broadcast an invalid message that can be discarded. In our BDB model, this type of fault would categorize as benign, and not deceitful, since invalid messages never contribute to a disagreement, but can instead prevent termination (by only sending invalid messages that are discarded). All omission faults are however benign faults, while the contrary is also not true (as per the same aforementioned example). Compared to the alive-but-corrupt failure model, deceitful faults are not restricted to only contribute to a disagreement if they know the disagreement will succeed, but instead we let them try forever, even if they do not succeed. Also, the alive-but-corrupt failure model does not define benign faults.

We believe thus the BDB model to be better-suited for consensus, as it establishes a clear difference in the types of faults depending on the type of property that the fault jeopardizes (agreement for deceitful, termination for benign), without restricting the behavior of these faults to the cases where they are certain that they will cause a disagreement. We restate that the property of validity is defined only to rule out trivial solutions of consensus in which all processes decide a constant, and this property can be locally checked for correctness.

III. Impossibility Results

In this section, we extend Dwork et al.'s impossibility results [7] on the number of non-faulty processes necessary to solve the Byzantine consensus problem in partial synchrony by adding deceitful and benign processes. First, we prove in Section III-A lower bounds on the size of the committee of any consensus protocol. Then, we prove in Section III-B some lower bounds depending on the voting threshold of that protocol, which we define in the same section.

A. Impossibility of consensus in the BDB model

First, we consider the case where $t = 0$, i.e., there are only deceitful and benign processes. In particular, we show in Lemma III.1 that if a protocol solves consensus then it tolerates at most $d < n - 2q$ deceitful processes and $q < n/2$ benign processes. The intuition for the proof is analogous to the classical impossibility proof of consensus in partial synchrony in the presence of $t_0 + 1$ Byzantine processes. Lemma III.1 extends the classical lower bound for the BFT model [7], by tolerating a stronger adversary than the classical bound (e.g. an adversary causing $d = \lceil n/3 \rceil - 1$ deceitful faults and $q = \lceil n/3 \rceil - 1$ benign faults). By contradiction, we show that in the presence of a greater number of faulty processes than bounded by Lemma III.1, in some executions all processes would either not terminate, or not satisfy agreement, if maintaining validity.

Lemma III.1. *Let a protocol σ and let σ solve consensus for all executions $\sigma_E \in E_\sigma(t, d, q)$ for some $t, d, q > 0$. Then, $d + t < n - 2(q + t)$.*

Proof. First, we show $q < n/2$ by contradiction, as done by previous work for omission faults [7]. Suppose $q \geq n/2$, $d = 0$, $t = 0$ and consider processes are divided into a disjoint partition P, Q such that P contains between 1 and q processes and Q contains $n - |P|$. First, consider scenario A: all processes in P are benign and the rest correct, and all processes in Q propose value 0. Then, by validity all processes in Q decide 0. Then, consider scenario B: all processes in Q are benign and the rest correct, and all processes in P propose value 1. Then, by validity all processes in P decide 1. Now consider scenario C: no process is benign, and processes in P propose all 1 while processes in Q propose all value 0. For processes in P scenario C is indistinguishable from scenario B, while for processes in Q scenario C is indistinguishable from scenario A. This yields a contradiction.

It follows that $q < n/2$. Hence, for $n = 2$, and since $q < 1$, it is immediate that for $d + t \geq 2$ it is impossible to solve consensus. As such, we have left to consider $d+t \geq n-2(q+t)$ with $n \geq 3$. We will prove this by contradiction.

Consider processes are divided into three disjoint partitions P, Q, R, such that P and Q contain between 1 and $q + t$ processes each, and R contains between 1 and $d + t$. First consider the following scenario A: processes in P and R are non-faulty and propose value 0, and processes in Q are benign. It follows that $P \cup R$ must decide value 0 at some time T_A, for if they decided 1 there would be a scenario in which processes in Q are non-faulty and also propose 0, but messages sent from processes in Q are delivered at a time greater than T_A, having processes in $P \cup R$ already decided 1. This would break the validity property. Also, they must decide some value to satisfy termination tolerating $q + t$ benign faults.

Consider now scenario B: processes in P are benign, and processes in R and Q are non-faulty and propose value 1. By the same approach, $R \cup Q$ decide 1 at a time T_B.

Now consider scenario C: processes in P and Q are non-faulty, and processes in R are deceitful, the messages sent from processes in Q are delivered by processes in P at a time greater than $\max(T_A, T_B)$, and the same for messages sent from processes in P to processes in Q. Then, for processes in P this scenario is identical to scenario A, deciding 0, while for processes in Q this is identical to scenario B, deciding 1, which leads to a disagreement. This yields a contradiction. □

Corollary III.2 (Impossibility of consensus with $t = 0$)**.** *It is impossible for a consensus protocol σ to tolerate d deceitful and q benign processes if $d \geq n - 2q$ or $q \geq n/2$.*

Proof. This is immediate from Lemma III.1 since σ is $(0, d, q)$-fault-tolerant if σ solves P for all executions $\sigma_E \in E_\sigma(0, d, q)$. □

We prove the impossibility result of Theorem III.3 by extending the result of Corollary III.2: it is impossible to solve

consensus in the presence of t Byzantine, q benign and d deceitful processes unless $n > 3t + d + 2q$.

Theorem III.3 (Impossibility of consensus)**.** *It is impossible for a consensus protocol to tolerate t Byzantine, d deceitful and q benign processes if $n \leq 3t + d + 2q$.*

Proof. This is immediate from Lemma III.1 since σ is (t, d, q)-fault-tolerant if σ solves P for all executions $\sigma_E \in E_\sigma(t, d, q)$. □

B. Impossibility bounds per voting threshold

The proofs for the impossibility results of Section III-A (and for the classical impossibility results [7]) derive a trade-off between agreement and termination. In some scenarios, processes must be able to terminate without delivering messages from a number of processes that may commit benign faults. In other scenarios, processes must be able to deliver messages from enough processes before terminating in order to make sure that no disagreement caused by deceitful faults is possible. We prove in this section the impossibility results depending on this trade-off.

A protocol that satisfies both agreement and termination in partial synchrony must thus state a threshold that represents the number of processes from which to deliver messages in order to be able to terminate without compromising agreement. If this threshold is either too small to satisfy agreement, or too large to satisfy termination, then the protocol does not solve consensus. We refer to this threshold as the *voting threshold*, and denote it with h. Typically, this threshold is $h = n - t_0 = \lceil \frac{2n}{3} \rceil$ to tolerate $t_0 = \lceil \frac{n}{3} \rceil - 1$ Byzantine faults [19], [10], [20], [21], [18]. We prove however in Lemma III.4 and Corollary III.5 that $h > \frac{d+t+n}{2}$ with $h \in (n/2, n]$ for safety.

Lemma III.4 (Impossibility of Agreement ($t = 0$))**.** *Let σ be a protocol with voting threshold $h \in (n/2, n]$ that satisfies agreement. Then σ tolerates at most $d < 2h - n$ deceitful processes.*

Proof. The bound $h \in (n/2, n]$ derives trivially: if $h \leq n/2$ then two subsets without any faulty processes can reach the threshold for different values (Lemma III.1). We calculate for which cases it is possible to cause a disagreement. Hence, we have two disjoint partitions of non-faulty processes such that $|A| + |B| \leq n - d$. Suppose that processes in A and in B decide each a different decision v_A, v_B, $v_A \neq v_B$. This means that both $|A| + d \geq h$ and $|B| + d \geq h$ must hold. Thus, $|A| + |B| + 2d \geq 2h$ and since $|A| + |B| \leq n - d$ this means that $n + d \geq 2h$ for a disagreement to occur. This means that if $h > \frac{n+d}{2}$ then it is impossible for d deceitful processes to cause a disagreement. □

The proof of Lemma III.4 can be straightforwardly extended to include Byzantine processes, resulting in Corollary III.5.

Corollary III.5. *Let σ be a protocol with voting threshold $h \in (n/2, n]$ that satisfies agreement. Then σ tolerates at most $d + t < 2h - n$ deceitful and Byzantine processes.*

Next, in Lemma III.6 and Corollary III.7 we show the analogous results for the termination property. That is, we show that if a protocol solves termination while $t = 0$, then it tolerates at most $q \leq n - h$ benign processes, or $q + t \leq n - h$ benign and Byzantine processes.

Lemma III.6 (Impossibility of Termination ($t = 0$))**.** *Let σ be a protocol with voting threshold h that satisfies termination. Then σ tolerates at most $q \leq n - h$ benign processes.*

Proof. If $n - q < h$, then termination is not guaranteed, since in this case termination would require the votes from some benign processes. This is impossible if $h \leq n - q$, as it guarantees that the threshold is lower than all processes minus the q benign processes. □

Corollary III.7. *Let σ be a protocol with voting threshold h that satisfies termination. Then, σ tolerates at most $q + t \leq n - h$ benign and Byzantine processes.*

Combining the results of corollaries III.5 and III.7, one can derive an impossibility bound for a consensus protocol given its voting threshold. We show this result in Corollary III.8.

Corollary III.8. *Let σ be a protocol that solves the consensus problem with voting threshold $h \in (n/2, n]$. Then, σ tolerates at most $d + t < 2h - n$ and $q + t \leq n - h$ Byzantine, deceitful and benign processes.*

We show in Figure 2 the threshold h to tolerate a number d of deceitful and q of benign processes. For example, for a threshold $h = \lceil \frac{5n}{9} \rceil - 1$, we have that $d < \frac{n}{9}$ for safety and $q < \frac{4n}{9}$ for liveness, with $t = 0$. The maximum number of Byzantine processes tolerated with $d = q = 0$ is the minimum of both bounds, being for example $t < \frac{n}{9}$ for $h = \lceil \frac{5n}{9} \rceil - 1$. In the remainder of this paper, we assume the adversary satisfies the resilient-optimal bounds of $h < n - q - t$ and $h > \frac{d+t+n}{2}$, given a particular voting threshold h. The result of Theorem III.3 holds regardless of the voting threshold. Thus, a protocol that satisfies both $h < n - q - t$ and $h > \frac{d+t+n}{2}$ can set its voting threshold $h \in (n/2, n]$ in order to solve consensus for any combination of t Byzantine, q benign and d deceitful processes, as long as $n > 3t + d + 2q$ holds.

IV. The Basilic Protocol

In this section, we introduce the Basilic class of protocols, a class of resilient-optimal protocols that solve, for different voting thresholds, the actively accountable consensus problem in the BDB model. In particular, all protocols within the Basilic class tolerate t Byzantine, d deceitful and q benign processes satisfying $n > 3t + d + 2q$, and, given a particular protocol $\sigma(h)$ of the class uniquely defined by a voting threshold $h \in (n/2, n]$, then $\sigma(h)$ tolerates a number n of processes satisfying $d + t < 2h - n$ and $q + t \leq n - h$. In this section, we first need to introduce few assumptions and definitions in Section IV-A. Second, we present the overview of the Basilic protocol in Section IV-B, and show its components in sections IV-D, IV-E, and IV-C.

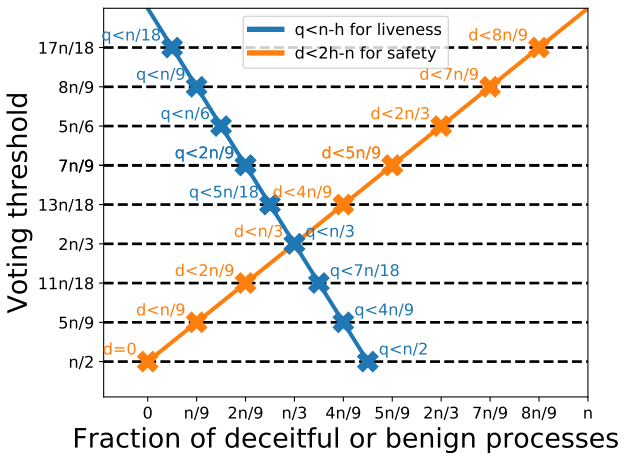

Fig. 2: Number of deceitful processes d and benign processes q tolerated for safety and liveness, respectively, per voting threshold h and with $t = 0$ Byzantine processes.

A. Additional Assumptions

a) Adversary: In order to limit the computational power of processes to prevent the adversary from forging keys, we model processes as probabilistic polynomial-time interactive Turing machines (ITMs) [22], [23], [24]. A process is an ITM defined by the following protocol: it is activated upon receiving an incoming message to carry out some computations, update its states, possibly generate some outgoing messages, and wait for the next activation. The adversary $\mathcal{A}$ is a probabilistic ITM that runs in polynomial time (in the number of message bits generated by non-faulty processes).

b) Actively accountable consensus problem: The accountable consensus problem [10] includes the property of accountability in order to provide guarantees in the event that deceitful and Byzantine processes manage to cause a disagreement. This property is however insufficient for the purpose of Basilic. We need an additional property that identifies and removes all deceitful behaviors that prevent termination. Faulty processes can break agreement in a finite number of conflicting messages, but once they send a pair of these conflicting messages, they leave a trace that can result in their exclusion from the system. Our goal is to exploit this trace to make sure that deceitful processes cannot contribute to breaking liveness. As a result, we include the property of active accountability, stating that deceitful faults do not prevent termination of the protocol.

Definition 1 (Actively accountable consensus problem)**.** *A protocol σ with voting threshold h solves the actively accountable consensus problem if the following properties are satisfied:*

- **Termination.** *Every non-faulty process eventually decides on a value.*
- **Validity.** *If all non-faulty processes propose the same value, no other value can be decided.*
- **Agreement.** *If $d + t < 2h - n$ then no two non-faulty processes decide on different values.*
- **Accountability.** *If two non-faulty processes output disagreeing decision values, then all non-faulty processes eventually identify at least $2h - n$ faulty processes responsible for that disagreement.*
- **Active accountability.** *Deceitful behavior does not prevent liveness.*

We generalise the previous definition of accountability [10] by including the voting threshold h. That is, the previous definition of accountability is the one we present in this work for the standard voting threshold of $h = 2n/3$.

B. Basilic Internals

Basilic is a class of consensus protocols, all these protocols follow the same pseudocode (Algorithms 1–2) but differ by their voting threshold $h \in (n/2, n]$. The structures of these protocols follow the classic reduction [25] from the consensus problem, which accepts any ordered set of input values, to the binary consensus problem, which accepts binary input values.

a) Basilic Overview: More specifically, Basilic has at its core the binary consensus protocol called *actively accountable binary consensus* or AABC for short (Alg. 2–3) and presented in Section IV-D. We show in Figure 3 an example execution with $n = 4$ processes in the committee. First, each process p_i selects their input value v_i, which they share with everyone executing an instance of a reliable broadcast protocol called *actively accountable reliable broadcast* or AARB for short. Then, processes execute one instance AABC$_i$ of the binary consensus protocol for each process p_i to decide whether to select their associated input value from process p_i. Finally, processes locally process the minimum input value from the values whose associated AABC instance output 1.

This AABC protocol shares similarities with Polygraph [10], as it also detects guilty processes, but goes further, by excluding these detected processes and adjusting its voting threshold at runtime to solve consensus even in cases where Polygraph cannot ($n/3 \leq t + q + d < n$). We summarize the comparison of Basilic with the state of the art in Table II. Finally, the rest of the reduction is depicted in Alg. 1 and invokes n actively accountable reliable broadcast instances or AARB (Alg. 4) described in Section IV-E, followed by n of the aforementioned AABC instances.

b) Certificates and transferable authentication.: Basilic uses certificates in order to validate or discard a message, and also to detect deceitful processes by cross-checking certificates. A certificate is a list of previously delivered and signed messages that justifies the content of the message on which the certificate is piggybacked. Thus, non-faulty processes perform transferable authentication [17]. That is, process p_i can deliver msg from p_j by verifying the signature of msg, even if msg was received from p_k, for $k \neq i \neq j$.

c) Detected deceitful processes: A key novelty of Basilic is to remove detected deceitful processes from the committee at runtime. For this reason, we refer to d_r as the number

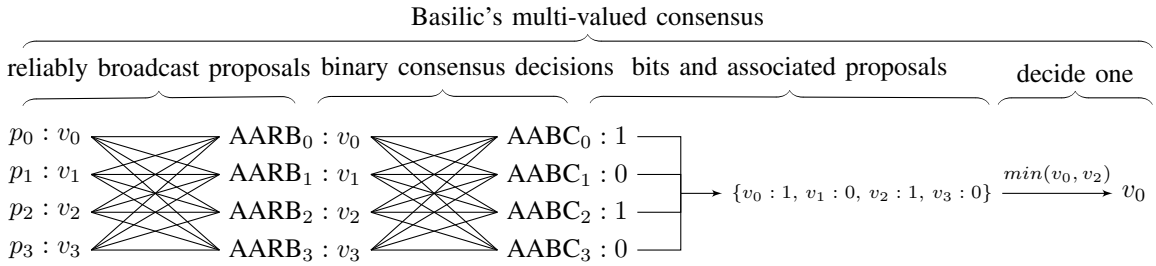

Fig. 3: Basilic execution example for a committee of $n = 4$. First, each process p_i selects their input value v_i, which they share with everyone executing their respective instance $AARB_i$ of $AARB$. Then, processes execute one instance $AABC_i$ of the binary consensus protocol to decide whether to select their associated input value from process p_i. Finally, processes locally process the minimum input value from the values whose associated AABC instance output 1.

of detected deceitful processes, and define a voting threshold $h(d_r)$ that varies with the number of detected deceitful processes. Therefore, processes start Basilic with an initial voting threshold $h(d_r = 0) = h_0$, e.g., $h_0 = \lceil \frac{2n}{3} \rceil$, but then update the threshold by removing detected deceitful processes, i.e. $h(d_r) = h_0 - d_r$. This way, detected deceitful processes break neither liveness nor safety, as we will show. Certificates must always contain $h(d_r)$ signatures from distinct processes justifying the message (after filtering out up to d_r signatures from detected deceitful processes), or else they will be discarded. Recall that the adversary is thus constrained to the bounds from Corollary III.8 depending on the voting threshold. As Basilic uses a threshold that updates at runtime starting from an initial threshold $h(d_r) = h_0 - d_r$, we restate these bounds applied to the initial threshold $h_0 \leq n-q-t$ and $h_0 > \frac{d+t+n}{2}$, or to the updated threshold $h(d_r) < n-q-t-d_r$ and $h(d_r) > \frac{d+t+n}{2} - d_r$.

C. The General Basilic Protocol

We bring together the n instances of the AABC binary consensus protocol with the n instances of the AARB reliable broadcast protocol in Algorithm 1, where we show the general Basilic protocol. The protocol derives from Polygraph's general protocol [10], which in turn derives from DBFT's multi-valued consensus protocol [19].

Non-faulty processes first start their respective AARB protocol (for which they are the proposing process) by proposing a value in line 2. Delivered proposals are stored in $msgs$ with the index corresponding to the source of the proposal. A binary consensus at index k is started with input value 1 for each index k where a proposal has been recorded (line 6). Notice that we can guarantee to decide 1 on at most $h(d_r)$ proposals (line 7), where d_r can be up to d and is set by update-committee in Algorithm 3, meaning that, for the standard threshold $h(d_r) = \lceil \frac{2n}{3} \rceil - d_r$, the maximum number of decided proposals is $\lceil \frac{n}{3} \rceil$, since $d_r < \frac{n}{3}$. Once non-faulty processes decide 1 on at least $h(d_r)$ AABC instances, non-faulty processes start the remaining AABC instances with input value 0 (line 9), without having to wait to AARB-deliver their respective values.

Finally, once all AABC instances have terminated (line 10), non-faulty processes can output a decision. As such, processes take as input a list of AARB-delivered values and their associated index and output a decision selecting the AARB-delivered value with the lowest associated index whose binary consensus with the same index output 1 (line 13).

Algorithm 1 The general Basilic with initial threshold h_0.

```
1:  Basilic-gen-propose_{h_0}(v_i):
2:    msgs ← AARB-broadcast(EST, ⟨v_i, i⟩)                    ▷ Algorithm 4
3:    repeat:
4:      if (∃v, k : (EST, ⟨v, k⟩) ∈ msgs) then                ▷ proposal AARB-delivered
5:        if (BIN-CONSENSUS[k] not yet invoked) then          ▷ Algorithm 2
6:          bin-decisions[k] ← BIN-CONSENSUS[k].AABC-prop(1)
7:    until |bin-decisions[k] = 1| ≥ h(d_r)                   ▷ decide 1 on at least h(d_r)
8:    for all k such that BIN-CONSENSUS[k] not yet invoked do
9:      bin-decisions[k] ← BIN-CONSENSUS[k].AABC-prop(0)
10:   wait until for all k, bin-decisions[k] ≠ ⊥
11:   j ← min{k : bin-decisions[k] = 1})
12:   wait until ∃v : (EST, ⟨v, j⟩) ∈ msgs
13:   decide v
```

D. Actively accountable Binary Consensus

We show in Algorithm 2 the Basilic *actively accountable binary consensus* (AABC) protocol with initial threshold $h_0 \in (n/2, n]$, along with some additional components and functions in Algorithm 3. First, note that all delivered messages are correctly signed (as wrongly signed messages are discarded) and stored in sig_msgs, along with all sent messages (as we detail in Rule 3 of Alg. 2).

The Basilic's AABC protocol is divided in two phases, after which a decision is taken. A key difference with Polygraph is that when a timer for one of the two phases reaches its timeout, if a process cannot terminate that phase yet, then it broadcasts its set of signed messages for that phase and resets the timer, as detailed in Rule 4. This allows Basilic to prevent deceitful processes from breaking termination by trying to cause a disagreement and never succeeding. For example, for $n = 4$ and $h = \lceil 2n/3 \rceil = 3$, if $q = 1$ and $d = 1$, the deceitful process can prevent the 2 non-faulty processes from terminating by constantly sending them conflicting messages, even if none of these non-faulty processes will reach the

Algorithm 2 Basilic's AABC with initial threshold h_0 for p_i.

14: AABC-prop$_{h_0}(v_i)$:
15: $est \leftarrow v_i$
16: $r \leftarrow 0$
17: $timeout \leftarrow 0$
18: $cert[0] \leftarrow \emptyset$
19: $bin\_vals \leftarrow \emptyset$
20: **repeat:**
21: $r \leftarrow r + 1$
22: $timeout \leftarrow \Delta$ ▷ *set timer*
23: $coord \leftarrow ((r-1) mod\, n) + 1$ ▷ *rotate coordinator*
24: ▶ **Phase 1:**
25: $timer \leftarrow$ start-timer$(timeout)$ ▷ *start timer*
26: abv-broadcast(EST$[r], est, cert[r-1], i, bin_vals$)
27: **if** $(i = coord)$ **then**
28: **wait until** $bin\_vals[r] = \{w\}$
29: broadcast(COORD$[r], w$)
30: **wait until** $bin\_vals[r] \neq \emptyset \wedge timer$ expired
31: ▶ **Phase 2:**
32: $timer \leftarrow timeout$ ▷ *reset timer*
33: **if** $((\text{COORD}[r], w) \in sig\_msgs \wedge w \in bin\_vals[r])$ **then**
34: $aux \leftarrow \{w\}$ ▷ *prioritize coordinator's value*
35: **else** $aux \leftarrow bin\_vals[r]$ ▷ *else use any received value*
36: broadcast(ECHO$[r], aux$) ▷ *broadcast signed* ECHO *message*
37: **wait until** $(vals =$ comp-vals$(sig_msgs, bin_vals, aux)) \neq \emptyset \wedge$ $timer$ expired
38: ▶ **Decision phase:**
39: **if** $(|vals| = 1)$ **then** $est \leftarrow vals[0]$ ▷ *if only one, adopt as estimate*
40: **if** $(est = (r \bmod 2) \wedge p_i$ not decided before) **then**
41: decide(est); returnest ▷ *if parity matches, decide*
42: **else** $est \leftarrow (r \bmod 2)$ ▷ *otherwise, the estimate is the round's parity bit*
43: $cert[r] \leftarrow$ compute-cert$(vals, est, r, bin_vals, sig_msgs)$
44: **Upon receiving a signed message** s_msg**:**
45: $pofs \leftarrow$ check-conflicts$(\{s_msg\}, sig_msgs)$ ▷ *returns* $\emptyset$ *or PoFs*
46: update-committee$(pofs)$ ▷ *remove fraudsters*
47: **Upon receiving a certificate** $cert\_msg$**:**
48: $pofs \leftarrow$ check-conflicts$(cert_msg, sig_msgs)$ ▷ *returns* $\emptyset$ *or PoFs*
49: update-committee$(pofs)$ ▷ *remove fraudsters*
50: **Upon receiving a list of PoFs** $pofs\_msg$**:**
51: **if** (verify-pofs$(pofs_msg)$) **then** ▷ *if proofs are valid then*
52: update-committee$(pofs_msg)$ ▷ *remove fraudsters from committee*
53: **Rules:**

1) Every message that is not properly signed by the sender is discarded.
2) Every message that is sent by abv-broadcast without a valid certificate after Round 1, except for messages with value 1 in Round 2, are discarded.
3) Every signed message received is stored in sig_msgs, including messages within certificates.
4) Every time the timer reaches the timeout for a phase, and if that phase cannot be terminated, processes broadcast their current delivered signed messages for that phase (and all messages received for future phases and rounds) and reset the timer for that phase. These messages are added to the local set of messages and cross-checked for PoFs on arrival.

threshold for the disagreeing values. Thus, once the timer is reached, processes exchange their known set messages and can update the committee removing processes that sent conflicting messages. It is important that processes wait for this timer before taking a decision for the phase, or before exchanging signed messages, since only waiting for that timer guarantees that all sent messages will be received before the timer reaches its timeout, after GST. Each process maintains an estimate (line 15), initially given as input, and then proceeds in rounds executing the following phases:

1) In the first phase, each process broadcasts its estimate (given as input) via an accountable binary value reliable broadcast (ABV-broadcast) (line 26), which we present in Algorithm 3, lines 67–85 and discuss in Section IV-D. Decision and abv-broadcast messages are discarded unless they come with a certificate justifying them.

The protocol also uses a rotating coordinator (line 23) per round which carries a special COORD message (lines 27-29). All processes wait until they deliver at least one message from the call to abv-broadcast and until the timer, initially set to Δ, expires (line 30). (Note that the bound on the message delays remains unknown due to the unknown GST.) If a process delivers a message from the coordinator (line 33), then it broadcasts an ECHO message with the coordinator's value and signature in the second phase (line 36). Otherwise, it echoes all the values delivered in phase 1 as part of the call to abv-broadcast (line 35).

2) In the second phase, processes wait till they receive $h(d_r)$ ECHO messages, as shown in the call to comp-vals (line 37), which returns the set of values that contain these $h(d_r)$ signed ECHO messages. Function comp-vals is depicted in Algorithm 3 (lines 86–95). Processes then try to come to a decision in lines 39-43. As it was the case for phase 1, when the timer expires in phase 2, all processes broadcast their current set of ECHO messages. Then, they update their committee if they detect deceitful processes through PoFs (lines 44-52) and recheck if they reach the updated $h(d_r)$ threshold, after which they reset the timer.

3) During the decision phase, if there is just one value returned by comp-vals and that value's parity matches with the round's parity, process p_i decides it (line 41) and broadcasts the associated certificate in the call to compute-cert. If the parity does not match then process p_i simply adopts the value as the estimate for the next round (line 39). If instead there is more than one value returned by comp-vals then p_i adopts the round's parity as next round's estimate (line 42). Adopting the parity as next round's estimate helps with convergence in the next round, in this case where processes are hesitating between two values. The call to compute-cert (depicted at lines 96–105 of Algorithm 3) gathers the signatures justifying the current estimate and broadcasts the certificate if the estimate was decided in this round.

a) Detecting and removing deceitful processes: Upon receiving a signed message, non-faulty processes check if the received message conflicts with some previously delivered message in storage in sig_msgs by calling check-conflicts (line 45). This function returns $pofs = \emptyset$ if there are no conflicting messages, or a list $pofs$ of PoFs otherwise. Then, at line 46, non-faulty processes call update-committee (depicted at lines 54–66 of Algorithm 3) to remove the $|pofs|$ detected deceitful processes at runtime. In the call to update-committee, process p_i removes all processes that are proven deceitful via new PoFs, and updates the committee N, its size n, and the voting threshold $h(d_r)$. After that, p_i rechecks all delivered messages in that phase in case it can now terminate the phase with the new threshold $h(d_r)$ (and after

Algorithm 3 Helper Components.

54: update-committee(new_pofs): ▷ *function that removes fraudsters*
55: **if** ($new\_pofs \neq \emptyset \wedge new\_pofs \not\subseteq local\_pofs$) **then**
56: $new\_pofs \leftarrow new\_pofs \backslash local\_pofs$ ▷ *consider only new PoFs*
57: $local\_pofs \leftarrow local\_pofs \cup new\_pofs$ ▷ *store new PoFs*
58: broadcast(POF, new_pofs) ▷ *broadcast new PoFs*
59: $new\_deceitful \leftarrow new\_pofs$.get_processes() ▷ *extract deceitful*
60: $new\_deceitful \leftarrow new\_deceitful \backslash local\_deceitful$
61: $local\_deceitful \leftarrow local\_deceitful \cup new\_deceitful$
62: $N \leftarrow N \backslash \{new\_deceitful\}$; $n \leftarrow |N|$ ▷ *remove new deceitful*
63: $d_r \leftarrow |local\_deceitful|$ ▷ *update number of detected deceitful*
64: $h(d_r) \leftarrow$ recalculate-threshold(N, d_r)
65: recheck-certs-termination() ▷ *check termination of current phase*
66: reset-current-timer() ▷ *reset timer of current phase*

67: abv-broadcast(MSG, val, $cert$, i, bin_vals):
68: broadcast(BVECHO, $\langle val, cert, i \rangle$) ▷ *broadcast message*
69: **if** ($r = 3$ **or** ($r = 2$ **and** $val = 1$)) **then**
70: discard all messages received without a valid certificate
71: **Upon** receipt of (BVECHO, $\langle v, \cdot, j \rangle$)
72: **if** ((BVECHO, $\langle v, \cdot, \cdot \rangle$) received from $\lfloor \frac{n-q-t}{2} \rfloor - d_r + 1$
73: distinct processes **and** (BVECHO, $\langle v, \cdot, i \rangle$) not yet broadcast) **then**
74: Let $cert$ be any valid certificate $cert$ received in these messages
75: broadcast(BVECHO, $\langle v, cert, i \rangle$)
76: **if** ((BVECHO, $\langle v, \cdot, \cdot \rangle$) received from $h(d_r)$ distinct processes **and**
77: (BVREADY, $\langle v, \cdot, \cdot \rangle$) not yet broadcast) **then**
78: Let $cert$ be any valid certificate $cert$ received in these messages
79: Construct bv_cert a certificate with $h(d_r)$ signed BVECHO
80: $bin\_vals \leftarrow bin\_vals$.add(BVREADY, $\langle v, cert, j, bv\_cert \rangle$)
81: broadcast(BVREADY, $\langle v, cert, j, bv\_cert \rangle$)
82: **if** ((BVREADY, $\langle v, cert, j, bv\_cert \rangle$) received from 1 process) **then**
83: $bin\_vals \leftarrow bin\_vals$.add(BVREADY, $\langle v, cert, j, bv\_cert \rangle$)
84: **if** ((BVREADY, $\langle v, cert, j, bv\_cert \rangle$) not yet broadcast) **then**
85: broadcast(BVREADY, $\langle val, cert, i, bv\_cert \rangle$)

86: comp-vals($msgs$, b_set, aux_set): ▷ *check for termination of phase 2*
87: **If** $\exists S \subseteq msgs$ where the following conditions hold:
88: (i) $|S|$ contains $h(d_r)$ distinct ECHO$[r]$ messages
89: (ii) aux_set is equal to the set of values in S ▷ $h(d_r)$ *with same est*
90: **then return**(aux_set)
91: **Else If** $\exists S \subseteq msgs$ where the following conditions hold:
92: (i) $|S|$ contains $h(d_r)$ distinct ECHO$[r]$ messages
93: (ii) Every value in S is in b_set ▷ $h(d_r)$ *messages with different est*
94: **then return**(V = the set of values in S)
95: **Else return**($\emptyset$) ▷ *else not ready to terminate*

96: compute-cert($vals$, est, r, bin_vals, $msgs$): ▷ *compute and send cert*
97: **if** ($est = (r \bmod 2)$) **then**
98: **if** ($r > 1$) **then**
99: $to\_return \leftarrow (cert : (\text{EST}[r], \langle v, cert, \cdot \rangle) \in bin\_vals)$
100: **else** $to\_return \leftarrow (\emptyset)$
101: **else** $to\_return \leftarrow (h(d_r)$ signed msgs containing only $est)$
102: **if** ($vals = \{(r \bmod 2)\} \wedge$ no previous decision by p_i) **then**
103: $cert[r] \leftarrow h(d_r)$ signed messages containing only $r \bmod 2$
104: broadcast($est, r, i, cert[r]$) ▷ *broadcast decision*
105: **return**(to_return)

filtering out messages delivered by the d_r removed deceitful processes) by calling recheck-certs-termination() in line 65 of Algorithm 3. Finally, it resets the timer for the current phase by calling reset-current-timer() in line 66 of Algorithm 3.

b) Termination and agreement of Basilic's AABC: We show the detailed proofs of agreement and termination in Lemmas A.8 and A.11. The idea is that removing deceitful processes has no effect on agreement, while it facilitates termination, since the threshold $h(d_r) = h_0 - d_r$ decreases the initial threshold h_0 with the number of removed deceitful processes. Also, since all non-faulty processes broadcast their delivered PoFs and thanks to the property of accountability, eventually all non-faulty processes agree on the same set of removed deceitful processes.

Then, if a process p_i terminates broadcasting certificate $cert_i$ while another process p_j already removed newly detected deceitful processes new_d_r present in $cert_i$, then $|cert_i| - new\_d_r \geq h(d_r + new\_d_r)$ by construction. As such, either a non-faulty process terminates and then all subsequent non-faulty processes can terminate, even after removing more deceitful processes, or they all eventually reach a scenario where all deceitful processes are detected $d_r = d$ and removed, after which they all terminate.

Note that removing processes at runtime can result in rounds whose coordinator is already removed. For the sake of correctness, we do not change the coordinator for that round even if it has already been removed. This guarantees that all non-faulty processes eventually reach a round in which they all agree on the same coordinator, which is a non-faulty process. If this round is the first after GST and after all deceitful processes have been removed from the committee, then non-faulty processes will reach agreement.

c) Accountable Binary Value Broadcast: The ABV-broadcast that we present in Algorithm 3 is inspired from the E protocol presented by Malkhi et al. [26] and the binary broadcast presented in Polygraph [10]. If non-faulty processes add a value v to bin_vals (lines 80 and 83) as a result of the ABV-broadcast, we say that they *ABV-deliver* v. Processes exchange BVECHO and BVREADY messages during ABV-broadcast. BVECHO messages are signed and must come with a valid certificate $cert_i$ justifying the value, as shown in lines 68 and 75. BVREADY messages carry the same information as BVECHO messages plus an additional certificate bv_cert containing $h(d_r)$ BVECHO messages justifying the BVREADY message, constructed in line 79. This way, as soon as a process receives a BVREADY message with a value (line 82), it already obtains $h(d_r)$ BVECHO messages too, meaning it can ABV-deliver that value adding it to bin_vals (lines 80 and 83). Non-faulty processes broadcast signed BVECHO messages for their estimate (line 68) and for all values for which they receive at least $\lfloor \frac{n-q-t}{2} \rfloor - d_r + 1$ signed BVECHO messages from distinct processes.

We prove in the technical report [27] that waiting for this many BVECHO messages for a value v guarantees that all non-faulty processes ABV-deliver v. In particular, we show in the technical report [27] that our ABV-broadcast satisfies the following properties: (i) ABV-Termination, in that every non-faulty process eventually adds at least one value to bin_vals; (ii) ABV-Uniformity, in that non-faulty processes eventually add the same values to bin_vals; (iii) ABV-Obligation, in that if $\lfloor \frac{n-q-t}{2} \rfloor - d_r + 1$ non-faulty processes ABV-broadcast a value v, then all non-faulty processes ABV-deliver v; (iv) ABV-Justification, in that if a non-faulty process ABV-delivers a value v then v was ABV-broadcast by a non-faulty process; and (v) ABV-Accountability, in that every ABV-delivered

value contains a valid certificate from the previous round.

We show in Lemma IV.1 that Basilic's AABC satisfies AABC-active accountability, but we defer the rest of the proofs of actively accountable binary consensus to the Appendix A.

Lemma IV.1 (AABC-Active accountability)**.** *Basilic's AABC satisfies active accountability.*

Proof. We show that if a faulty process p_i sends two conflicting messages to two subsets A, $B \subseteq N$, each containing at least one non-faulty process, then eventually all non-faulty processes terminate, or instead they receive a PoF for p_i and remove it from the committee, after which they all terminate.

First, we observe that no process gets stuck in some round. Process p_i cannot get stuck in phase 1 since, by ABV-Termination, every non-faulty process eventually ABV-delivers a value.

We show now that a process also does not get stuck waiting on phase 2. First, notice that every value that is included in an ECHO message from a non-faulty process is eventually delivered to bin_vals. Then, note that all non-faulty processes eventually deliver $h(d_r)$ ECHO messages, or instead, when the timer expires, processes will exchange their delivered ECHO messages and be able to construct PoFs and remove d_r deceitful processes that are preventing termination. In the latter case, after removing all deceitful processes from the committee and updating the threshold, processes will now deliver enough ECHO messages to terminate phase 2, since $h(d_r) < n - q - t - d$ for $d_r = d$.

Then, we show that all non-faulty processes always hold a valid certificate to broadcast a proper message, which could otherwise prevent termination of a phase during the ABV-broadcast in phase 1. For an estimate whose parity is the same as that of the finished round $r-1$ (line 101), process p_i must have received a valid certificate for the round (or it would not have terminated such round). If the parity matches in this round r, then a non-faulty process can always construct a valid certificate from the delivered estimates in round $r-1$ (line 99).

As a result, all processes always progress infinitely in every round. Consider the first round r after GST where (i) the coordinator is non-faulty and (ii) all deceitful processes have been detected and removed by all non-faulty processes. In this case, every non-faulty process will prioritize the coordinator's value, adopting it as their ECHO message adding only that value. Hence, every non-faulty process adopts the same value, and decides either in round r or round $r+1$ (by Lemma A.7). □

E. Actively accountable Reliable Broadcast

Algorithm 4 shows Basilic's *actively accountable reliable broadcast* (AARB). The protocol is analogous to the secure broadcast presented in previous work [26], with the difference that we also introduce a timer that non-faulty processes use to periodically broadcast their set of delivered ECHO messages, in order to detect deceitful processes. We refer to the process that starts the AARB protocol as the *source*. The protocol starts when the source broadcasts an INIT message with its proposed value v (line 107). Upon delivering that message, all non-faulty processes also broadcast a signed ECHO message with v (line 109). Then, once a process p_i delivers $h(d_r)$ distinct signed ECHO messages for the same value v, p_i first broadcasts a READY message (line 112) with a certificate containing the $h(d_r)$ ECHO messages justifying v (constructed in line 111), and then AARB-delivers the value (line 113). The same occurs if instead a process delivers just one valid READY message containing a valid certificate justifying it in lines 114-118.

As it occurs with Basilic's AABC protocol presented in Algorithms 2 and 3, upon cross-checking newly received signed messages with previously delivered ones (lines 120 and 123), non-faulty processes can detect deceitful faults and update the committee (lines 121 and 124), removing them at runtime, by calling update-committee. This can also occur when receiving a list of PoFs (line 125). Note that this is the same call to the same function as in the AABC protocol shown in Algorithm 2, because non-faulty processes update the committee across the entire Basilic protocol, and not just for that particular instance of AARB or AABC where the deceitful process was detected. We show in Appendix A that Basilic's AARB protocol satisfies the following properties of actively accountable reliable broadcast:

- **AARB-Unicity.** Non-faulty processes AARB-deliver at most one value.
- **AARB-Validity.** Non-faulty processes AARB-deliver a value if it was previously AARB-broadcast by the source.
- **AARB-Send.** If the source is non-faulty and AARB-broadcasts v, then non-faulty processes AARB-deliver v.
- **AARB-Receive.** If a non-faulty process AARB-delivers v, then all non-faulty processes AARB-deliver v.
- **AARB-Accountability.** If two non-faulty processes AARB-deliver distinct values, then all non-faulty processes receive PoFs of the deceitful behavior of at least $2h(d_r) - n$ processes including the source.
- **AARB-Active accountability.** Deceitful behavior does not prevent liveness.

F. Basilic's fault tolerance in the BDB model

We show in Figure 4 the combinations of Byzantine, deceitful and benign processes that Basilic tolerates, depending on the initial threshold h_0. The solid lines represent the variation in tolerance to benign and deceitful processes as the number of Byzantine processes varies for a particular threshold. For example, for $h_0 = \frac{2n}{3}$, if $t = 0$ then $d < \frac{n}{3}$ and $q < \frac{n}{3}$. As t increases, for example to $t = \lceil \frac{n}{6} \rceil - 1$, then $d < \frac{n}{6}$ and $q < \frac{n}{6}$.

We compare our Basilic's fault tolerance with that of previous works in Figure 5. In particular, we represent multiple values of the initial threshold $h_0 \in \{5n/9, 2n/3, 3n/4, 5n/6\}$ for Basilic. First, we show that classical Byzantine fault-tolerant (BFT) protocols tolerate only the case $t < n/3$ with a blue triangle dot (▲) in the figure. This is the case of most partially synchronous BFT consensus protocols [19], [10], [20], [21], [18]. Notice that Zero-loss Blockchain [18] (ZLB) also tolerates instead $d < 5n/9$ and $3q + d < n$ faults, where

Algorithm 4 Basilic's AARB with initial threshold h_0.

AARB-broadcast$_{h_0}(v_i)$: ▷ *executed by the source*
  broadcast(INIT, v_i) ▷ *broadcast to all*
  Upon receiving (INIT, v_i) **from** p_j **and not having sent** ECHO**:**
    boadcast(ECHO, v, j) ▷ *echo value to all*
  Upon receiving $h(d_r)$ (ECHO, v, j) **and not having sent a** READY**:**
    Construct $cert_i$ containing at least $h(d_r)$ signed msgs (ECHO, v, j)
    broadcast(READY, $v, cert_i, j$) ▷ *broadcast certificate*
    AARB-deliver(v, j) ▷ *AARB-deliver value*
  Upon receiving (READY, $v, cert, j$), **and not having sent a** READY:
    if (verify($cert$) $= False$) **then continue**
    Set $cert_i$ to be one of the valid certs received (READY, $v, cert, j$)
    broadcast(READY, $v, cert_i, j$) ▷ *broadcast certificate*
    AARB-deliver(v, j) ▷ *AARB-deliver value*
  Upon receiving a signed message s_msg**:**
    $pofs \leftarrow$ check-conflicts($\{s\_msg\}$, sig_msgs) ▷ *returns ∅ or PoFs*
    update-committee($pofs$) ▷ *remove fraudsters*
  Upon receiving a certificate $cert\_msg$**:**
    $pofs \leftarrow$ check-conflicts($cert\_msg$, sig_msgs) ▷ *returns ∅ or PoFs*
    update-committee($pofs$) ▷ *remove fraudsters*
  Upon receiving a list of PoFs $pofs\_msg$**:**
    if (verify-pofs($pofs\_msg$)) **then** ▷ *if proofs are valid then*
      update-committee($pofs\_msg$) ▷ *exclude from committee*

Rules:

1) Processes broadcast their current delivered signed INIT and ECHO messages once a timer $timer$, initially set to Δ, reaches 0, and reset the timer to Δ.

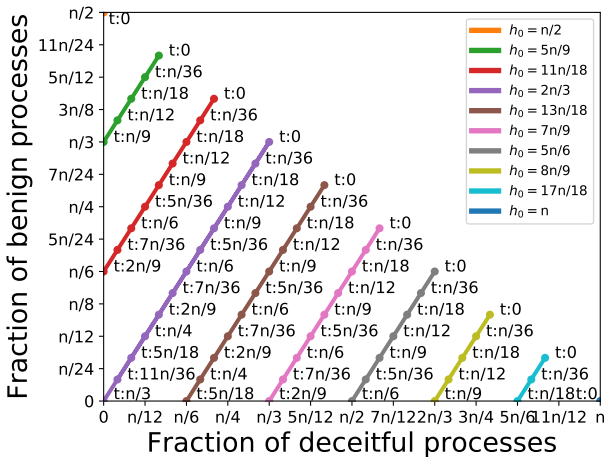

Fig. 4: Combinations of benign, deceitful and Byzantine processes that Basilic tolerates, for an initial threshold h_0.

d and q is the number of deceitful and benign faults, but that ZLB does not solve consensus for these bounds, and instead it recovers from disagreements. Second, we represent Flexible BFT [9] in their greatest fault tolerance setting in partial synchrony. As we can see, such setting overlaps with Basilic's initial threshold of $h_0 = 2n/3$. However, the difference lies in that while Basilic tolerates all the cases in the solid line $h_0 = 2n/3$, Flexible BFT only tolerates a particular dot of the line, set at the discretion of each client. That is, Flexible BFT's clients must decide, for example, whether they tolerate either $\lceil 2n/3 \rceil - 1$ total faults, being none of them Byzantine, or instead tolerate $\lceil n/3 \rceil - 1$ Byzantine faults, not tolerating any additional fault. Basilic can however tolerate any range satisfying both $h_0 > \frac{n+d+t}{2}$ for safety and $h_0 \leq n - q - t$ for liveness, which allows our clients and servers to tolerate significantly more combinations of faults for one particular threshold $h_0 \in (n/2, n]$. For this reason, we represent the line of Flexible BFT as a dashed line, whereas Basilic's lines are solid. For each initial voting threshold h_0, the maximum number of Byzantine processes Basilic tolerates is $t < \min(2h_0 - n, 1 - h_0)$, which is obtained by setting $q = d = 0$ and resolving both bounds for safety and liveness.

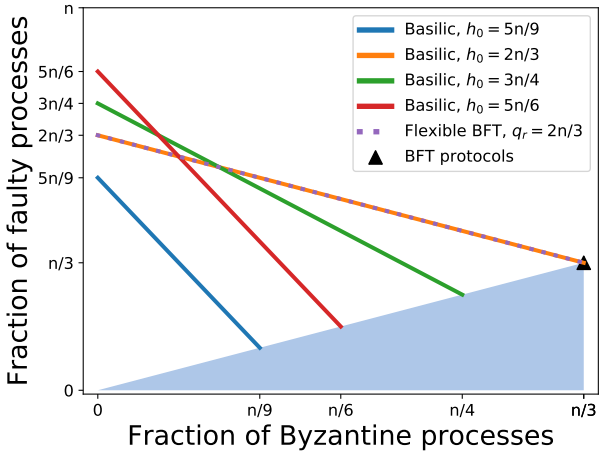

Fig. 5: Fraction of faulty processes, compared with fraction of Byzantine processes, for a particular initial threshold h_0 of the general Basilic protocol, compared with other works.

G. Basilic's correctness

We show in Lemma IV.2 that Basilic satisfies active accountability. We defer to Appendix A the rest of the proofs that show that the Basilic class of protocols solves the actively accountable consensus problem for the resilient-optimal bounds of the impossibility results shown in Section III.

Lemma IV.2 (Active accountability)**.** *Basilic satisfies active accountability.*

Proof. We show that if a faulty process p_i sends two conflicting messages to two subsets $A, B \subseteq N$, each containing at least one non-faulty process, then eventually all non-faulty processes terminate, or instead they receive a PoF for p_i and remove it from the committee, after which they all terminate.

First, analogously to Lemma IV.1, w.l.o.g. we treat only the case $d_r = 0$, since all conflicting messages that can be sent in Basilic are messages of Basilic's AARB or AABC, that already satisfy active accountability (see Lemmas IV.1 and A.6). This means that if $d_r > 0$, then non-faulty processes eventually update the committee and threshold, after which they recheck if they hold enough signed messages to terminate. Next, we prove termination. By the AARB-Send property (Lemma A.3), all non-faulty processes will eventually deliver the proposals from non-faulty processes. Eventually all non-faulty processes propose 1 in all binary consensus whose index corresponds to a non-faulty proposer, and by AABC-Validity

decide 1. Since eventually $h(d_r) \leq n - q - d - t$ if enough d_r prevent termination and are thus detected and removed, we can conclude that at least $h(d_r)$ binary consensus instances will terminate deciding 1.

Once non-faulty processes decide 1 on at least $h(d_r)$ proposals, they propose 0 to the rest (line 9), and by AABC-Termination (Lemma A.11) all remaining binary consensus instances will terminate. Next, we show that for every binary consensus upon which we decided 1, at least one non-faulty process AARB-delivered its associated proposal. For the sake of contradiction, if no non-faulty process had AARB-delivered its associated proposal, then all non-faulty processes would have proposed 0, meaning by AABC-Validity that the final decision of the binary consensus would have been 0, not 1. As a result, by the AARB-Receive property (Lemma A.4), eventually all non-faulty processes will deliver the proposal for all binary consensus that they decided 1 upon. Finally, processes decide the value proposed by the proposer with the lower index. □

We summarize all proofs in the result shown in Theorem IV.3 to show that the Basilic protocol with initial threshold h_0 solves consensus if $d + t < 2h_0 - n$ and $q + t \leq n - h_0$. This result translates in the Basilic class of protocols solving consensus if $n > 3t + d + 2q$, as we show in Corollary A.19.

Theorem IV.3 (Consensus per threshold)**.** *The Basilic protocol with initial threshold h_0 solves the actively accountable consensus problem if $d + t < 2h_0 - n$ and $q + t \leq n - h_0$.*

Corollary IV.4 (Consensus)**.** *The Basilic class of protocols solves actively accountable consensus if $n > 3t + d + 2q$.*

V. Basilic's complexity

In this section, we show the complexities of Basilic. We execute one instance of Basilic's AARB reliable broadcast and of Basilic's AABC binary consensus per process. We prove these complexities in the appendix B.

A. Naive Basilic

We summarize the complexities of the three protocols without optimizations in Table I.

Complexity	AARB	AABC	Basilic
Time	$\mathcal{O}(1)$	$\mathcal{O}(n)$	$\mathcal{O}(n)$
Message	$\mathcal{O}(n^2)$	$\mathcal{O}(n^3)$	$\mathcal{O}(n^4)$
Bit	$\mathcal{O}(\lambda n^3)$	$\mathcal{O}(\lambda n^4)$	$\mathcal{O}(\lambda n^5)$

TABLE I: Time, message and bit complexities of Basilic AARB, AABC and the general Basilic protocol, after GST, and without optimizations.

B. Optimized Basilic

The complexities of Basilic after GST share the same asymptotic complexity of other recent works that are not actively accountable [10], [28], some of them not being accountable either [29], as we show in Table II. This is because the adversary cannot prevent termination of any phase. Thus, after GST, all processes can continue to the next phase or terminate the protocol by the time the timer for that phase expires, resulting in an execution equivalent to that of Polygraph (apart from one additional message broadcast in ABV-broadcast). In this table, naive Basilic represents the protocol we show in Algorithm 1 and Table I, whereas the following row, multi-valued Basilic, shows the analogous optimizations shown in Polygraph and applicable to the Basilic protocol as well [10]. The rows containing 'superblock' refer to the result of applying the additional superblock optimization [19], [30], which consists on deciding on the union of all $h(d_r)$ ($\mathcal{O}(n)$) proposals whose associated AABC instance output 1. This optimization is only available to protocols without a leader in which all processes propose a value [19], [10] (i.e. DBFT, Polygraph and Basilic in Table II). After these optimizations, the resulting normalized bit complexity (i.e. per decision) of Basilic is as low as those of other works that are only accountable and not actively accountable, such as BFT Forensics [28] or Polygraph [10]. Furthermore, since this is the lowest complexity to obtain accountability [10], this means that this is also optimal in the bit complexity. Note that other optimizations present in other works, such as the possibility to obtain an amortized complexity of $\mathcal{O}(\lambda n^2)$ in BFT Forensics per decision after n iterations of the protocol [31], is also possible in Basilic's consensus protocol. An additional advantage of Basilic, as well as of other leaderless protocols, compared to leader-based works [28], [31], is that the distribution of proposals scatters the bits throughout multiple channels of the network, instead of bloating channels that have the leader as sender or recipient, as previously noted [30].

Finally, not only are the rest of the protocols in Table II not actively accountable, but also this means that they only solve consensus tolerating at most $t < n/3$ faults in the BDB model, whereas Basilic with initial threshold $h_0 = 2n/3$ solves consensus where $d + t < n/3$ and $q + t \leq n/3$ faults, hence tolerating the strongest adversary among these proposals.

TABLE II: Complexities of Basilic compared to other works.

Algorithm	Msgs	Bits	Acc.	Actacc.
PBFT [29]	$\mathcal{O}(n^3)$	$\mathcal{O}(\lambda n^4)$	✗	✗
Tendermint [32]	$\mathcal{O}(n^3)$	$\mathcal{O}(\lambda n^3)$	✗	✗
HotStuff [31]	$\mathcal{O}(n^2)$	$\mathcal{O}(\lambda n^2)$	✗	✗
DBFT superblock [19]	$\mathcal{O}(n^3)$	$\mathcal{O}(n^3)$	✗	✗
BFT Forensics [28]	$\mathcal{O}(n^2)$	$\mathcal{O}(\lambda n^3)$	✓	✗
Polygraph's binary [10]	$\mathcal{O}(n^3)$	$\mathcal{O}(\lambda n^4)$	✓	✗
Naive Polygraph [10]	$\mathcal{O}(n^4)$	$\mathcal{O}(\lambda n^5)$	✓	✗
Polygraph Multi-v. [10]	$\mathcal{O}(n^4)$	$\mathcal{O}(\lambda n^4)$	✓	✗
Polygraph superblock [10]	$\mathcal{O}(n^3)$	$\mathcal{O}(\lambda n^3)$	✓	✗
Basilic's AABC	$\mathcal{O}(n^3)$	$\mathcal{O}(\lambda n^4)$	✓	✓
Naive Basilic	$\mathcal{O}(n^4)$	$\mathcal{O}(\lambda n^5)$	✓	✓
Multi-valued Basilic	$\mathcal{O}(n^4)$	$\mathcal{O}(\lambda n^4)$	✓	✓
Basilic superblock	$\mathcal{O}(n^3)$	$\mathcal{O}(\lambda n^3)$	✓	✓

VI. Related Work

Accountability has been proposed for distributed systems by Haeberlen et al. [33] in PeerReview and particularly for the problem of consensus by Civit et al. [34] in Polygraph. We extended Polygraph in ZLB [18] to tolerate up to $5n/9$ deceitful faults for eventual consensus, but ZLB tolerates only $t < n/3$ for consensus. This work leverages accountability to replace deceitful processes by new processes. Unfortunately, ZLB requires deceitful processes to eventually stop trying to cause a disagreement.

Malkhi et al. [9] offers Flexible BFT, a failure model and theoretical results to tolerate $\lceil 2n/3 \rceil - 1$ alive-but-corrupt (abc) processes. An abc process behaves maliciously only if it knows it can violate safety, and behaves correctly otherwise. This is an even stronger assumption than ZLB's deceitful faults eventually behaving correctly. Additionally, their fault tolerance requires a commitment from clients to not tolerate a single Byzantine fault in order to tolerate $\lceil 2n/3 \rceil - 1$ abc faults, or to instead tolerate no abc faults if clients decide to tolerate $t = \lceil n/3 \rceil - 1$ Byzantine faults.

Neu et al.'s ebb-and-flow system [35] is available in partial synchrony for $t < n/3$ and satisfies finality in synchrony for $t < n/2$. They also motivate the need for a model like BDB in their recent availability-accountability dilemma [36]. Sheng et al. [28] characterize the forensic support of a variety of blockchains. Unfortunately, none of these works tolerate $q = \lceil \frac{n}{3} \rceil - 1$ benign and even $d = 1$ deceitful faults, or $d = \lceil \frac{n}{3} \rceil - 1$ and even $q = 1$ benign fault, a direct consequence of them not satisfying active accountability.

Upright [2] tolerates $n = 2u+r+1$ faults, where u and r are the numbers of commission and omission faults, respectively. Upright tolerates $n/3$ commission faults or instead $n/2$ omission faults, falling short of Basilic's $q + d < 2n/3$ deceitful and benign faults or $t < n/3$ Byzantine faults tolerated. Upright does also not tolerate more faults for commission than the lower bound for BFT consensus. Anceaume et al. [37] tolerate $t < n/2$ Byzantine faults for the problem of eventual consensus, at the cost of not tolerating $t = 1$ Byzantine fault for deterministic consensus. Our Basilic class also tolerates this case if h_0 is set to $h_0 = \lfloor \frac{n}{2} \rfloor + 1$, this is part of the Basilic class. We refer to the technical report [27] for a proof of correctness of Basilic for eventual consensus.

Basilic is, to the best of our knowledge, the first protocol tolerating $n > 3t+d+2q$ in the BDB model, thanks to the property of active accountability. However, previous works already try to discourage misbehaviors by threatening of slashing a deposit or removing a faulty process, or both. Ranchal-Pedrosa and Gramoli [38] propose the Trap protocol, an accountable consensus protocol tolerating up to k rational players and t Byzantine players causing a disagreement by threatening deviant rationals, for $n > \max(\frac{3}{2}k + 3t, 2(k+t))$. Freitas de Souza et al. [39] provide an asynchronous accountable lattice agreement protocol. Shamis et al. [40] store signed messages in a dedicated ledger so as to punish processes in case of misbheavior. Buterin and Griffith propose the Casper [41] algorithm that incurs a penalty in case of double votes but does not ensure termination when $t < n/3$. Although Buchman et al. [42] aimed at slashing processes without accountability, the authors have recently incorporated accountability in [43]. Li et al. propose SUNDR [44] that requires cross-communication between non-faulty clients to detect failures. Lev-Ari et al. propose FairLedger [45] that requires synchrony to detect faulty processes.

VII. Conclusion

In this paper, we showed that it is impossible to solve consensus in the BDB model against an adversary controlling $n > 3t + d + 2q$, where t, d, and q are the number of Byzantine, deceitful and benign processes, respectively. We then present our Basilic class of protocols, the first class of resilient-optimal protocols for the consensus problem in the BDB model. Basilic solves actively accountable consensus tolerating any combination of t, d and q Byzantine, deceitful and benign processes, respectively, satisfying $h_0 > \frac{n+d+t}{2}$ for safety and $h_0 \leq n - q - t$ for liveness, for an initial voting threshold h_0. We prove this result to be resilient-optimal. We showed that Basilic's bit complexity is comparable to previous accountable consensus protocols that tolerate less faults.

Acknowledgements

This work is supported in part by the ARC Future Fellowship funding scheme (#180100496) and the Ethereum Foundation.

References

[1] A. Singh, P. Fonseca, P. Kuznetsov, R. Rodrigues, P. Maniatis *et al.*, "Zeno: Eventually consistent Byzantine-fault tolerance," in *NSDI*, 2009.

[2] A. Clement, M. Kapritsos, S. Lee, Y. Wang, L. Alvisi, M. Dahlin, and T. Riche, "Upright cluster services," in *SOSP*, 2009.

[3] M. Kapritsos, Y. Wang, V. Quéma, A. Clement, L. Alvisi, and M. Dahlin, "All about eve: Execute-verify replication for multi-core servers," in *OSDI*, 2012.

[4] S. Liu, P. Viotti, C. Cachin, V. Quéma, and M. Vukolic, "XFT: practical fault tolerance beyond crashes," in *OSDI*, 2016.

[5] T. Crain, C. Natoli, and V. Gramoli, "Red Belly: A secure, fair and scalable open blockchain," in *S&P*, 2021.

[6] L. Lamport, R. Shostak, and M. Pease, "The Byzantine generals problem," *ACM Trans. Program. Lang. Syst.*, 1982.

[7] C. Dwork, N. Lynch, and L. Stockmeyer, "Consensus in the presence of partial synchrony," *J. ACM*, 1988.

[8] M. Lokhava, G. Losa, D. Mazières, G. Hoare, N. Barry, E. Gafni, J. Jove, R. Malinowsky, and J. McCaleb, "Fast and secure global payments with stellar," in *SOSP*, 2019.

[9] D. Malkhi, K. Nayak, and L. Ren, "Flexible Byzantine fault tolerance," in *CCS*, 2019.

[10] P. Civit, S. Gilbert, and V. Gramoli, "Polygraph: Accountable byzantine agreement," in *ICDCS*, 2021.

[11] P. Kuznetsov, A. Tonkikh, and Y. X. Zhang, "Revisiting optimal resilience of fast byzantine consensus," in *PODC*, 2021.

[12] Y. Xue and M. Herlihy, "Hedging against sore loser attacks in cross-chain transactions," in *PODC*, 2021.

[13] I. Abraham, P. Jovanovic, M. Maller, S. Meiklejohn, G. Stern, and A. Tomescu, "Reaching consensus for asynchronous distributed key generation," in *PODC*, 2021.

[14] M. Backes and C. Cachin, "Reliable broadcast in a computational hybrid model with byzantine faults, crashes, and recoveries." in *DSN*, 2003.

[15] I. Abraham, K. Nayak, L. Ren, and Z. Xiang, "Good-case latency of byzantine broadcast: A complete categorization," in *PODC*, 2021.

[16] K. P. Kihlstrom, L. E. Moser, and P. M. Melliar-Smith, "Byzantine Fault Detectors for Solving Consensus," *The Computer Journal*, 2003.

[17] A. Clement, F. Junqueira, A. Kate, and R. Rodrigues, "On the (limited) power of non-equivocation," in *PODC*, 2012.
[18] A. Ranchal-Pedrosa and V. Gramoli, "Blockchain is dead, long live blockchain! accountable state machine replication for longlasting blockchain," *arXiv preprint arXiv:2007.10541*, 2020.
[19] T. Crain, V. Gramoli, M. Larrea, and M. Raynal, "DBFT: Efficient leaderless Byzantine consensus and its applications to blockchains," in *NCA*, 2018.
[20] R. Kotla, L. Alvisi, M. Dahlin, A. Clement, and E. Wong, "Zyzzyva: Speculative Byzantine fault tolerance," in *SOSP*, 2007.
[21] M. Yin, D. Malkhi, M. K. Reiter, G. G. Gueta, and I. Abraham, "HotStuff: BFT consensus with linearity and responsiveness," in *PODC*, 2019.
[22] Y. Lu, Z. Lu, Q. Tang, and G. Wang, "Dumbo-MVBA: Optimal multi-valued validated asynchronous Byzantine agreement, revisited," in *PODC*, 2020.
[23] C. Cachin, K. Kursawe, and V. Shoup, "Random oracles in constantinople: Practical asynchronous byzantine agreement using cryptography," *Journal of Cryptology*, 2005.
[24] C. Cachin, K. Kursawe, F. Petzold, and V. Shoup, "Secure and efficient asynchronous broadcast protocols," in *CRYPTO*, 2001.
[25] M. Ben-Or, R. Canetti, and O. Goldreich, "Asynchronous secure computation," in *STOC*, 1993.
[26] D. Malkhi, M. Merritt, and O. Rodeh, "Secure reliable multicast protocols in a wan," in *ICDCS*, 1997.
[27] A. Ranchal-Pedrosa and V. Gramoli, "Basilic: Resilient optimal consensus protocols with benign and deceitful faults," *arXiv preprint arXiv:2204.08670*, 2022.
[28] P. Sheng, G. Wang, K. Nayak, S. Kannan, and P. Viswanath, "BFT protocol forensics," in *CCS*, 2021.
[29] M. Castro and B. Liskov, "Practical Byzantine fault tolerance and proactive recovery," *ACM Trans. Comput. Syst.*, 2002.
[30] T. Crain, C. Natoli, and V. Gramoli, "Evaluating the Red Belly Blockchain," *arXiv preprint arXiv:1812.11747*, 2018.
[31] P. Tholoniat and V. Gramoli, "Formal verification of blockchain byzantine fault tolerance," in *FRIDA*, Oct 2019.
[32] E. Buchman, "Tendermint: Byzantine fault tolerance in the age of blockchains," 2016, Master's Thesis.
[33] A. Haeberlen, P. Kouznetsov, and P. Druschel, "PeerReview: Practical accountability for distributed systems," in *SOSP*, 2007.
[34] P. Civit, S. Gilbert, and V. Gramoli, "Brief announcement: Polygraph: Accountable byzantine agreement," in *DISC*, 2020.
[35] J. Neu, E. Tas, and D. Tse, "Ebb-and-flow protocols: A resolution of the availability-finality dilemma," in *S&P*, 2021.
[36] J. Neu, E. N. Tas, and D. Tse, "The availability-accountability dilemma and its resolution via accountability gadgets," *arXiv preprint arXiv:2105.06075*, 2021.
[37] E. Anceaume, A. Pozzo, T. Rieutord, and S. Tucci-Piergiovanni, "On finality in blockchains," *arXiv preprint arXiv:2012.10172*, 2020.
[38] A. R. Pedrosa and V. Gramoli, "Trap: The bait of rational players to solve byzantine consensus," in *AsiaCCS*, 2022.
[39] L. F. de Souza, P. Kuznetsov, T. Rieutord, and S. Tucci Piergiovanni, "Brief announcement: Accountability and reconfiguration - self-healing lattice agreement," in *DISC*, 2021.
[40] A. Shamis, P. Pietzuch, M. Castro, E. Ashton, A. Chamayou, S. Clebsch, A. Delignat-Lavaud, C. Fournet, M. Kerner, J. Maffre *et al.*, "PAC: Practical accountability for CCF," *arXiv preprint arXiv:2105.13116*, 2021.
[41] V. Buterin and V. Griffith, "Casper the friendly finality gadget," *arXiv preprint arXiv:1710.09437v4*, 2019.
[42] E. Buchman, J. Kwon, and Z. Milosevic, "The latest gossip on BFT consensus," *arXiv preprint arXiv:1807.04938*, 2018.
[43] E. Buchman, R. Guerraoui, J. Komatovic, Z. Milosevic, D. Seredinschi, and J. Widder, "Revisiting tendermint: Design tradeoffs, accountability, and practical use," in *DSN*, 2022, pp. 11–14.
[44] J. Li, M. Krohn, D. Mazières, and D. Shasha, "Secure untrusted data repository (SUNDR)," in *OSDI*, 2004.
[45] K. Lev-Ari, A. Spiegelman, I. Keidar, and D. Malkhi, "Fairledger: A fair blockchain protocol for financial institutions," *OPODIS*, 2019.

APPENDIX

A. Basilic Proofs

In this section, we prove the rest of the properties of Basilic, including its AABC and AARB protocols, but we refer to the technical report [27] for the proofs of ABV-broadcast.

1) Actively accountable reliable broadcast: In this section, we prove the properties of Basilic's AARB. In these proofs. We prove AARB-unicity in Lemma A.1, AARB-validity in Lemma A.2, AARB-send in Lemma A.3, AARB-Receive in Lemma A.4, AARB-accountability in Lemma A.5 and AARB-active accountability in Lemma A.6.

Lemma A.1 (AARB-Unicity)**.** *Non-faulty processes AARB-deliver at most one value.*

Proof. By construction all non-faulty processes AARB-deliver at most one value. □

Lemma A.2 (AARB-Validity)**.** *If non-faulty process p_i AARB-delivers v, then v was AARB-broadcast by p_s.*

Proof. Process p_i AARB-delivers v if it receives $h(d_r)$ messages $\langle \text{ECHO}, v, \cdot, \cdot \rangle$. Non-faulty processes only send an ECHO message for v if they receive $\langle \text{INIT}, v \rangle$. Thus, since $d + t < h(d_r)$, p_s AARB-broadcast v to at least one non-faulty process. □

Lemma A.3 (AARB-Send)**.** *If p_s is non-faulty and AARB-broadcasts v, then all non-faulty processes eventually AARB-deliver v.*

Proof. Deceitful processes either broadcast v or multicast v' to a partition A and v to a partition B. In the first case (in which all deceitful behave like non-faulty processes), since the number of benign and Byzantine processes is $q+t < n-h(d_r)$ it follows that at least $h(d_r)$ non-faulty processes will echo v, being that enough for all processes to eventually AARB-deliver it.

Consider instead some $d_r \leq d+t$ deceitful processes behave deceitful echoing different messages to two different partitions each containing at least one non-faulty process. Then when the timer expires and non-faulty processes exchange their delivered ECHO messages, all processes will update their committee removing the d_r detected deceitful. Thus, since processes also recalculate the thresholds and recheck them after updating the committee, this case becomes the aforementioned case where no deceitful process behaves deceitful. The same occurs if one of the partitions AARB-delivers a value while the other does not and reaches the timer (Lemma A.4). □

Lemma A.4 (AARB-Receive)**.** *If a non-faulty process AARB-delivers v from p_s, then all non-faulty processes eventually AARB-deliver v from p_s.*

Proof. First, since $d + t < 2h(d_r) - n$ it follows that deceitful and Byzantine processes can not cause two non-faulty processes to AARB-deliver different values (analogously to Lemma III.4). Then, before a process p_i AARB-delivers a value v, it broadcasts a READY message containing the

certificate that justifies delivering v. Thus, when p_j receives that READY message, it also AARB-delivers v. □

Lemma A.5 (AARB-Accountability)**.** *If two non-faulty processes p_i and p_j AARB-deliver v and v', respectively, such that $v \neq v'$, then all non-faulty processes eventually receive PoFs of the deceitful behavior of at least $2h(d_r) - n$ processes (including p_s).*

Proof. Non-faulty processes broadcast the certificates of the values they AARB-deliver, containing $h(d_r)$ signed ECHO messages from distinct processes. Hence, analogous to Lemma III.4, at least $2h(d_r) - n$ processes must have sent conflicting ECHO messages, and they will be caught upon cross-checking the conflicting certificates. Also, some non-faulty processes must have received conflicting INIT messages from p_s in order to reach the threshold $h(d_r)$ to AARB-deliver conflicting messages, meaning that p_s is also faulty. □

Lemma A.6 (AARB-Active accountability)**.** *The Basilic's AARB protocol satisfies active accountability.*

Proof. We prove here that if a number of faulty processes send conflicting messages to two subsets $A, B \subseteq N$, each containing at least one non-faulty process, then (i) eventually all non-faulty processes terminate without removing the faulty processes, or (ii) eventually all non-faulty processes receive a PoF for these faulty processes and remove them from the committee, after which, if the source is non-faulty, they terminate.

W.l.o.g. we consider just $p_A \in A$ and $p_B \in B$. If they both terminate despite the conflicting messages, we are finished. Suppose instead a situation in which only one of them, for example p_A, terminated AARB-delivering a value v. Then p_A broadcast a READY message with enough $h(d_r)$ ECHO messages in the certificate *cert* for p_B to also AARB-deliver v and terminate. Let us consider w.l.o.g. only one faulty process p_i. If a signature from p_i in *cert* conflicts with a local signature from p_i stored by p_B, then p_B constructs and broadcasts a PoF for p_i, and then updates the committee and the threshold. Then, it rechecks the certificate filtering out the signature by p_i, which would cause p_B to also AARB-deliver v (since the threshold also decreased accordingly).

Suppose neither p_A nor p_B has terminated yet. Then, when the timer is reached and they both broadcast their delivered INIT and ECHO, they will both be able to construct a PoF for p_i, after which they update the committee and the threshold. Then, if the source was non-faulty, non-faulty processes can terminate analogously to the previous case. □

2) Basilic binary consensus: We focus in this section on the properties of Basilic's binary consensus, AABC. We first prove that if all non-faulty processes start a round r with the same estimate v, then all non-faulty processes decide v in round r or $r + 1$. Then, we prove AABC-agreement in Lemma A.8, AABC-strong validity in Lemma A.9 and AABC-validity as Corollary A.10 of Lemma A.9, AABC-active accountability in Lemma IV.1, AABC-termination in Lemma A.11, and AABC-accountability in Lemma A.12. This thus makes AABC the first actively accountable binary consensus protocol, as we show in Theorem A.13.

Lemma A.7. *Assume that each non-faulty process begins round r with the estimate v. Then every non-faulty process decides v either at the end of round r or round $r + 1$.*

Proof. By ABV-Obligation, v is eventually delivered to every non-faulty process. By ABV-Justification, v is the only value delivered to each non-faulty process. As such, v is the only value in bin_vals and the only value echoed by non-faulty processes, since deceitful processes that prevent termination are removed from the committee when the timer expires (and the threshold is updated). This means that v will be the only value in $vals$. If $v = r \mod 2$ then all non-faulty processes decide v. Otherwise, by the same argument every non-faulty process decides v in round $r + 1$. □

Lemma A.8 (AABC-Agreement)**.** *If $d + t \leq 2h - n$, no two non-faulty processes decide different values.*

Proof. W.l.o.g. assume that the non-faulty process p_i decides v in round r. This means that p_i received $h(d_r)$ ECHO messages in round r, and that $vals = \{v\}$. Consider the ECHO messages received by non-faulty process p_j in the same round. If v is in p_j's $vals$ then p_j adopts estimate v because $v = r \mod 2$. If instead p_j's $vals = \{w\}$, $w \neq v$, then p_j received $h(d_r)$ ECHO messages containing only w.

Analogously to Lemma III.4, it is impossible for p_j and for p_i to receive $h(d_r)$ ECHO messages for v and for w, respectively. We then conclude, by Lemma A.7, that non-faulty processes decide value v in round $r + 1$ or round $r + 2$. □

Lemma A.9 (AABC-Strong Validity)**.** *If a non-faulty process decides v, then some non-faulty process proposed v.*

Proof. This proof is identical to Polygraph's proof of strong validity [10]. □

Corollary A.10 (AABC-Validity)**.** *If all processes are non-faulty and begin with the same value, then that is the only decision value.*

Lemma A.11 (AABC-Termination)**.** *Every non-faulty process eventually decides on a value.*

Proof. This proof derives directly from Lemma IV.1. □

Lemma A.12 (AABC-Accountability)**.** *If two non-faulty processes output disagreeing decision values, then all non-faulty processes eventually identify at least $2h - n$ faulty processes responsible for that disagreement.*

Proof. This proof is identical to Polygraph's proof of accountability [10], with the a generalization to any threshold $h(d_r)$ analogous to the one we make in Lemma A.5. □

Theorem A.13. *Basilic's AABC solves the actively accountable binary consensus problem.*

Proof. Immedaite from Corollary A.10 and Lemmas A.8, IV.1, A.11, and A.12. □

3) General Basilic protocol: We prove in this section the Basilic protocol's active accountability in Lemma IV.2, validity in Lemma A.15, termination in Corollary A.14, agreement in Lemma A.16, and accountability in Lemma IV.2, converging to Theorem A.18.

Corollary A.14 (Termination)**.** *Basilic satisfies termination.*

Proof. Trivial from Lemma IV.2. □

Lemma A.15 (Validity)**.** *Basilic satisfies validity.*

Proof. This is trivial by Corollary A.10 and the proofs of AARB. Suppose all processes begin Basilic with value v. If all processes are non-faulty then every proposal AARB-delivered was AARB-sent by a non-faulty process, and since all processes AARB-send v, only v is AARB-delivered.

Since initially processes only start an AABC instance for which they can propose 1, this means that eventually all processes start one AABC instance proposing 1. By Corollary A.10, this instance will terminate with all processes deciding 1. Since the rest of the AABC instances will eventually terminate by Lemma A.11, this means that processes will terminate at least one instance of AABC outputting 1. Upon calculating the minimum of all values (which are all v) whose associated bit is set to 1, all processes will decide v. □

Lemma A.16 (Agreement)**.** *Basilic satisfies agreement.*

Proof. Immediate from Lemmas A.8 and A.4. □

Lemma A.17 (Accountability)**.** *If two non-faulty processes output disagreeing decision values, then all non-faulty processes eventually identify at least $2h - n$ faulty processes responsible for that disagreement.*

Proof. Immediate from Lemmas IV.1 and A.6. □

Theorem A.18 (Theorem IV.3)**.** *The Basilic protocol with initial threshold $h_0 \in (n/2, n]$ solves the actively accountable consensus problem if $d + t < 2h_0 - n$ and $q + t \leq n - h_0$.*

Proof. Corollary A.14 and Lemmas IV.2, A.15, A.16, and IV.2 satisfy termination, active accountability, validity, agreement, and accountability, respectively. □

Corollary A.19 (Corollary IV.4)**.** *The Basilic class of protocols solves the actively accountable consensus problem if $n > 3t + d + 2q$.*

Proof. The proof is immediate from Theorem IV.3 after removing h_0 from the system of two inequations defined by $d + t < 2h_0 - n$ and $q + t \leq n - h_0$. □

B. Proofs of Basilic's complexities

We prove in this section the complexities of Basilic, and of Basilic's AARB and AABC, which we presented in Section V.

Lemma A.20 (Basilic's AARB Complexity)**.** *After GST and if the source is non-faulty, Basilic's AARB has time complexity $\mathcal{O}(1)$, message complexity $\mathcal{O}(n^2)$ and bit complexity $\mathcal{O}(\lambda \cdot n^3)$.*

Proof. After GST, non-faulty processes will have received a message from each non-faulty process and from each deceitful processes by the timeout. Thus, either non-faulty processes can terminate, or they broadcast their current list of ECHO and INIT messages, after which they remove the detected deceitful processes, and they can terminate too. Thus, the time complexity is $\mathcal{O}(1)$. Then, the message complexity is $\mathcal{O}(n^2)$, as each non-faulty process broadcasts at least one ECHO and READY message, and, in some executions, a list of ECHO messages that they delivered by the time the timer reaches 0. Since READY messages contain $\mathcal{O}(n)$ signatures, or $\mathcal{O}(\lambda n)$ bits, the bit complexity of Basilic's AARB is $\mathcal{O}(\lambda n^3)$. □

Lemma A.21 (Basilic's AABC Complexity)**.** *After GST, Basilic's AABC protocol has time complexity $\mathcal{O}(n)$, message complexity $\mathcal{O}(n^3)$ and bit complexity $\mathcal{O}(\lambda \cdot n^4)$.*

Proof. After GST, the Basilic protocol terminates in the first round (i) whose leader is a non-faulty process and (ii) after having removed enough deceitful faults so that they cannot prevent termination. Since $t+d+q < n$, we have that (i) holds in $\mathcal{O}(n)$. As for every added round in which deceitful faults prevent termination, a non-zero number of deceitful faults are removed, we have that (ii) holds in $\mathcal{O}(n)$ as well. This means that Basilic terminates in $\mathcal{O}(n)$ rounds. In each round during phase 1 of AABC, non-faulty processes execute an ABV-broadcast of $\mathcal{O}(n^2)$, obtaining $\mathcal{O}(n^3)$ messages. The bit complexity is $\mathcal{O}(\lambda n^4)$ as each message may contain up to two ledgers of $\mathcal{O}(n)$ signatures, or $\mathcal{O}(\lambda n)$ bits. The complexities of phase 2 are equivalent and obtained analogously to those of phase 1, as non-faulty processes may broadcast $\mathcal{O}(n)$ signatures if deceitful faults prevent termination of phase 2, or a certificate if they decide in this round. □

Theorem A.22. *The Basilic protocol has time complexity $\mathcal{O}(n)$, message complexity $\mathcal{O}(n^4)$ and bit complexity $\mathcal{O}(\lambda \cdot n^5)$.*

Proof. The proof is immediate from Lemma A.21 and Lemma A.20 since Basilic executes n instances of AARB and after n instances of AABC. □