On the Subject of Non-Equivocation

Defining Non-Equivocation in Synchronous Agreement Systems

Mads Frederik Madsen
IT University of Copenhagen
Copenhagen, Denmark
mfrm@itu.dk

Søren Debois
IT University of Copenhagen
Copenhagen, Denmark
debois@itu.dk

ABSTRACT

We study *non-equivocation* in synchronous agreement protocols: the restriction on faulty processes that they cannot act differently towards distinct non-faulty processes. Guarantees of non-equivocation have been used to provide improved fault tolerance in agreement protocols, and various mechanisms for achieving it, have been proposed. However, the exact meaning of non-equivocation varies subtly in the literature. In this paper, we propose two different formal notions of non-equivocation: *strong* and *weak*. We define both as fault models for synchronous agreement protocols with reliable channels, and we show how the two models yield distinct bounds for the minimal number of communication rounds required and the maximum number of faulty processes tolerable to achieve agreement: 1 round, $n > t$ for strong non-equivocation; and $t + 1$ rounds, $n > 2t$ for weak non-equivocation. This makes weak non-equivocation the only fault model with a lower bound on fault tolerance of $n > 2t$ for broadcast agreement and interactive consistency, confirming the folklore knowledge that equivocation is, in a sense, the most critical of the Byzantine faults. Finally, we show how the weak and strong non-equivocation fault models relate to well-known agreement problems: strong non-equivocation corresponds to Byzantine broadcast and weak non-equivocation to crusader agreement.

CCS CONCEPTS

• **Theory of computation → Distributed algorithms**.

KEYWORDS

Agreement, non-equivocation, fault tolerance, synchronous systems

ACM Reference Format:
Mads Frederik Madsen and Søren Debois. 2020. On the Subject of Non-Equivocation: Defining Non-Equivocation in Synchronous Agreement Systems. In *Proceedings of The 39th ACM Symposium on Principles of Distributed Computing (PODC'20).* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Equivocation is the act of a faulty process *acting differently* when correct behaviour is to be consistent. In this paper, we study *non-equivocation* in synchronous agreement protocols: the guarantee that otherwise Byzantine processes will never equivocate.

The notion of equivocation was introduced informally and without the name by Lamport et al. [25]. Subsequently, Chun et al. [11] named it and showed that assuming non-equivocation (in one form) allows for an improvement in fault tolerance. Mechanisms providing non-equivocation can be constructed in different ways, with the most well-known being perhaps Attested Append-Only Memory (A2M) [11] and Trusted Incrementers (TrInc) [26].

Studies of non-equivocation have used subtly different definitions of the concept, leading to subtly different results. E.g.,[27] proposes the non-equivocation based "f-resilient" condition for agreement in synchronous systems, and conjecture informally that this property will be enough also in asynchronous ones. Meanwhile, [12] proves that transferable authentication is a necessary addition to non-equivocation for a fault tolerance improvement in asynchronous systems. The result of [12] does not contradict the conjecture in [27] as the two papers are referring to different notions of non-equivocation.

Thus, we propose two different concepts of non-equivocation: *strong* and *weak*. Both require faulty processes to not "lie differently", but weak non-equivocation allows faulty processes to selectively omit messages to some participants, where strong non-equivocation does not. In the above example [27] refers to a notion like that of strong non-equivocation, while [12] refers to one like that of weak non-equivocation.

Both A2M and TrInc can provide at best weak non-equivocation since a malicious or faulty process using a trusted module can still choose to simply not send messages. On the other hand, partial broadcast channels [13, 21, 22, 33], and local broadcast channels [23, 24] provide strong non-equivocation, as they guarantee broadcast messages are reliably and identically received by all neighbours.

Technically, we work in the original framework of Pease et al. [30], where we define both weak and strong non-equivocation as fault models for synchronous systems with reliable channels. We show for both how to solve the agreement problem of *interactive consistency* [30]. For strong non-equivocation, this is trivial; we obtain a 1-round, $n > t$ fault-tolerant algorithm simply by broadcasting a value, where n is the number of processes, and t is the number of faulty processes. For weak non-equivocation, we show how the problem is solvable using the algorithm of Pease et al. [30] with only minor adjustments. In this case, $t+1$ rounds are necessary, and we obtain exactly a fault-tolerance of $n > 2t$.

We relate the present results to the existing body of work on agreement protocols in Table 1 on page 3. To the best of our knowledge, weak non-equivocation is the only fault model with a lower bound of $n > 2t$ on the fault tolerance of broadcast agreement and interactive consistency in a synchronous model with reliable channels. This result demonstrates formally how equivocation is at the core of impossibility of Byzantine fault tolerance.

The notions of weak and strong non-equivocation as fault models fit neatly with our existing understanding of agreement problems in the following sense. Under guarantees of strong non-equivocation, it is trivial to implement Byzantine broadcast; conversely, given a Byzantine broadcast oracle, it is trivial to obtain strong non-equivocation. Similarly, under guarantees of weak non-equivocation, it is trivial to solve crusader agreement [15]; and conversely, weak non-equivocation is straightforward to obtain given a crusader agreement oracle. Essentially, properties of non-equivocation in synchronous models come down to properties of an oracle for the corresponding agreement problem.

The weak non-equivocation fault model resembles the authenticated Byzantine fault model in that both can be achieved by the use of cryptographic mechanisms. However, we show that while they do share a subspace of runs not in the omission fault model, both also contain a subspace of runs that the other fault model does not. So as fault models, they overlap, but only partly.

Overview. We proceed as follows. After presenting related work, we give in Section 2 a system model, in particular recalling the framework of Pease et al. [30]. We introduce strong non-equivocation in Section 3, followed by weak non-equivocation in Section 4. In Section 5 we give bounds on rounds and fault tolerance, and in Section 6 we prove those bounds tight. Finally, we discuss relations to other agreement problem variants in Section 7.

Related work. In *agreement protocols* processes attempt to agree on one or more values. Processes are given an input value, a *private value*, and must agree on one or more of these values, the *decision value(s).* Table 1 shows optimal fault tolerance for agreement problems under various fault models, including results of the present paper. The notion of (authenticated) Byzantine agreement problems was introduced by Pease, Shostak, and Lamport [30][25], formulated as interactive consistency and the Byzantine generals problem. In interactive consistency, the non-faulty processes must agree on a value for each process, while in the Byzantine generals problem (i.e. Byzantine broadcast agreement) the non-faulty processes must agree on the private value of a pre-specified process. Interactive consistency can be achieved by running an instance of broadcast agreement per process [19]. Consensus is the problem formulation where every process suggests a value, and non-faulty processes decide on a single common value. It notably differs from interactive consistency and broadcast agreement in the validity property [3, 19]: if all non-faulty processes suggest the same value, then all non-faulty processes must decide that value. This stronger validity property makes the problem impossible to solve, unless $n > 2t$, when processes are allowed to lie about their value. Consensus can be achieved, if $n > 2t$, by running a majority function an interactive consistency vector, and similarly broadcast agreement can be achieved by running a consensus algorithm on the received values of a broadcast [19]. Uniform consensus differs from the other agreement problems in that also *faulty* processes must agree on the decided value [32, 34]. Fault-tolerant agreement is generally impossible under asynchrony (FLP-impossibility), as shown by Fischer, Lynch, and Paterson [20] because non-faulty processes cannot distinguish a faulty process from delayed messages. This impossibility is commonly circumvented with either partial synchrony [18], relaxation of termination or determinism [5, 7, 31] or the use of failure-detectors [9, 10].

Non-equivocation was suggested by Chun et al. [11] as a means of improving fault-tolerance, exemplified via improvements to PBFT by Castro and Liskov [8] for asynchronous Byzantine consensus with relaxed termination. The non-equivocation mechanism used was an attested append-only log. Levin et al. [26] showed how to construct such a log more efficiently using increment-only counters. Non-equivocation is strongly linked to broadcast primitives, and such two primitives have been used to achieve non-equivocation; partial broadcast channels [13, 21, 22, 33] and local broadcast channels [23, 24]. Here, non-equivocation means that messages sent through special channels are received reliably and identically by all neighbours. This corresponds to our notion of *strong non-equivocation* in Section 3 but limited to subsets of all processes. Other non-equivocation notions, e.g. that of [1, 4, 11, 12, 14, 27], are weaker system-wide properties, where faulty processes may omit messages. This corresponds to our notion of *weak non-equivocation* in Section 4. Algorithms that use non-equivocation to achieve agreement with improved fault-tolerance include [1, 14, 27].

Fitzi and Maurer [21] showed that synchronous Byzantine broadcasts (i.e. Byzantine generals) can be achieved with $n > 2t$ using a partial broadcast mechanism, which provides a property akin to strong non-equivocation, between only subsets of 3 processes. Chun et al. [11] showed how non-equivocation can be used to implement an $n > 2t$ resilient PBFT solution. Clement et al. [12] showed that neither non-equivocation nor transferable authentication is enough to give an $n > 2t$ resilient agreement protocol under asynchrony with relaxed termination; both are needed. Lastly, Jaffe et al. [22] examined the trade-off between equivocation and redundancy by showing under which conditions a fault-tolerance of $n = 2t + h$, for any $h > 0$, can be achieved using partial broadcast channels providing (strong) non-equivocation.

2 SYSTEM MODEL AND NOTATION

A system comprises n synchronous processes connected with synchronous and reliable point-to-point channels. Up to t processes may be faulty, and can deviate arbitrarily from the protocol, but not break the guarantees awarded by the fault model. To simplify the presentation, we separate the message-exchange protocol from the rest of the algorithm. By slight abuse of language, we will refer to the message-exchange as *the protocol*, and the calculation made by a process on its received messages as *the algorithm.*

We define $P = \{p, q, r, \ldots\}$ to be the set of all processes, N to be the subset of all non-faulty processes, and F to be the subset of faulty processes: $N \cup F = P$, $N \cap F = \emptyset$, $|N| = n$, and $|F| = t$. As usual, P^* denotes all finite strings over P, P^+ denotes all non-empty finite strings, and we refer to an element of P^* as a *process-sequence.*

Table 1: Optimal fault tolerance of different agreement problems in different fault models, assuming synchronous processes and reliable, synchronous channels. Rows titled in bold represent results of the present paper.

Fault model	Broadcast agreement	Interactive consistency	Consensus	Uniform consensus
Byzantine	$n > 3t$ [25][†]	$n > 3t$ [30]	$n > 3t$ [25][‡]	N/A
Weak non-equivocation	$n > 2t$	$n > 2t$	$n > 2t$	N/A
Authenticated Byzantine	$n > t$ [25]	$n > t$ [30]	$n > 2t$ [35]	N/A
Omission	$n > t$ [34]	$n > t$ [34]	$n > t$ [34]	$n > 2t$ [32]
Crash	$n > t$ [28]	$n > t$ [28]	$n > t$ [28]	$n > t$ [34]
Strong non-equivocation	$n > t$	$n > t$	$n > 2t$	N/A

[†] Aka Byzantine generals — n processes, t faulty

[‡] Aka Byzantine agreement

Each message sent comprises of a value $v \in V$, and a member of P^+, representing the route the value has been sent along. Each process can correctly determine the immediate sender of a message. Note that lying about the process-sequence of a message is equivalent to lying about the value [30], so we consider only lies about values for the remainder of this paper.

We will define non-equivocation as a fault model for the protocol. A fault model limits which messages faulty processes may send. For this purpose, we use the framework of [30], where the message-exchange is represented as a map $\sigma : P^+ \rightarrow V$ from process-sequences to values. For instance, $\sigma(p|q) = v$ means that q has sent v to p, ("|" denotes string concatenation); and $\sigma(p|q|r) = v$ that q has reported to p that r has reported the initial value v to q. Processes do not send messages to themselves, so, abusing notation, we consider only process-sequences which do not contain consecutive duplicate characters, e.g., we disregard $p|p|q$.

A process p obviously has access only to the messages it has received. Formally, we define σ_p as $\forall w \in P^*\colon \sigma_p(w) = \sigma(p|w)$. For ease of notation, and since process-sequences cannot include consecutive duplicates, we let $\sigma_p(p|w) = \sigma_p(w)$. In cases of message omissions, we use $\perp$ to denote the absence of a value in σ. We use $\langle\perp\rangle$ as the value specifying the absence of a value, that is, if $\sigma(p|r) = \perp$ then a correct p will report $\langle\perp\rangle$ to any q: $\sigma(q|p|r) = \langle\perp\rangle$.

2.1 Interactive consistency

Recall the *interactive consistency problem* [30]. Each process p starts with a private value v_p and must compute a vector, vec_p, of size n such that:

Agreement: All non-faulty processes compute the same vector: $\forall p, q \in N\colon vec_p = vec_q$

Validity: If q is non-faulty, all non-faulty processes sets the q'th element in the vector to q's private value: $\forall p, q \in N\colon vec_p[q] = v_q$

Termination: All non-faulty processes must eventually compute a vector.

Interactive consistency is known to be reducible to Byzantine broadcast and vice versa [19], so to solve interactive consistency, it is sufficient to give an algorithm describing Byzantine broadcast. We exploit this equivalence to interchangeably solve one or the other, depending on which is more convenient: We generally find it more convenient to solve Byzantine broadcast, however, interactive consistency supports a more intuitive induction proof in Section 5.

For Sections 3 to 5, we use the following protocol, which is a specialized round-based message-passing protocol. All processes go through $t + 1$ *information exchange broadcast rounds.* In the first broadcast round, the non-faulty processes broadcast their private values, and subsequently, in round r they broadcast the values they received in round $r - 1$, along with the route that value has taken so far. As we consider only a finite number of rounds and synchronous channels and processes, *termination* of the protocol is trivial, and we need only consider the termination of the algorithm.

This protocol gives us the following guarantee: a non-faulty process neither lies about its private value, nor its previously received messages.

Definition 2.1 (Non-faulty process guarantee). *A non-faulty process truthfully broadcast values, reporting $\langle\perp\rangle$ only when they have observed a message omission:*

$$\forall q \in N : \forall p \in P \setminus \{q\} : \forall w \in P^* :$$
$$\sigma(p|q|w) = \begin{cases} \langle\perp\rangle, & \textit{if } \sigma(q|w) = \perp \\ \sigma(q|w), & \textit{otherwise} \end{cases}$$

In the non-faulty process guarantee, $\langle\perp\rangle$ can be interpreted as the reporting process reporting that there exists a suffix of the process-sequence of that message, let's call it w', for which $\sigma(w') = \perp$. Note that $\langle\perp\rangle$ is simply a value, and *can* be reported as any process' private value. Also, note that this interpretation is not a guarantee; a faulty process can report $\langle\perp\rangle$ having received neither $\perp$ nor $\langle\perp\rangle$ in the previous round. *Faulty* processes are allowed to deviate arbitrarily from the protocol and so from Definition 2.1; they are restricted only by the fault model.

3 STRONG NON-EQUIVOCATION

We define strong non-equivocation fault model as a weakening of the Byzantine fault model:

Definition 3.1 (Strong non-equivocation). *For any given process-sequence, all processes—faulty or not—report the same value to all other processes:*

$$\forall p, r, r' \in P\colon \forall w \in P^*\colon \sigma(r|p|w) = \sigma(r'|p|w)$$

Since w can be the empty string and is prefixed with the same p on both sides of the equality, this is equivalent[1] *to*

$$\forall r, r' \in P \colon \forall w \in P^+ \colon \sigma(r|w) = \sigma(r'|w)$$

Under strong non-equivocation, a faulty process may still lie: a faulty process q may receive v from p in round 1, but proceed to claim to other processes that p said w in round 2. Similarly, a faulty process may omit messages under the strong non-equivocation fault model, but only if it omits all messages in that round.

The strong non-equivocation fault model overlaps with the crash fault model: a faulty process may fail by not sending any messages from a given round onwards, thereby acting as if crashed. However, the strong non-equivocation fault model does not contain the crash fault model: in the crash fault model a faulty process can fail during its broadcast, leading to a situation where some processes have received a message from the crashing process, while others have not; a violation of Definition 3.1. Conversely, a faulty process can lie in a given round in the strong non-equivocation fault model, which is not allowed in either the general omission- or the crash fault model. So the strong non-equivocation fault model must overlap with a strict subspace of the crash fault model, but also extend into a strict subspace of the Byzantine fault model, see Figure 4 on page 9 for an illustration of the fault model spaces.

Given a guarantee of strong non-equivocation, it is trivial to solve *Byzantine broadcast*, and so also interactive consistency. For source q, each correct p simply decides on the value it was sent by q in round 1: $vec_p[q] = \sigma_p(p|q)$. Clearly, if q is correct, each correct process sets the same value. If q is faulty, then by Definition 3.1, each other process has received the same message from q, and so all decide on the same value. (We give a formal proof of correctness in Appendix A.) Note that the strong non-equivocation fault model really is very strong: all is done in one round, and within that single round, the source process *cannot fail* in a way that makes it in any way distinguishable from a correct process, making interactive consistency and Byzantine broadcast solvable for $n > t$. Note that for the problem of consensus, the lower bound is $n > 2t$, as processes can lie about their private value (see Section 1).

We argue that Byzantine broadcast and strong non-equivocation are interchangeable in the sense that strong non-equivocation implies a trivial solution to Byzantine broadcast as shown above. Similarly, a Byzantine broadcast oracle can be used to make algorithms assuming the strong non-equivocation fault model work in the Byzantine fault model. Informally, the latter holds because, after Byzantine broadcast, all non-faulty processes agree on some value for the source process, which is exactly strong non-equivocation.

Strong non-equivocation is most useful in systems with partial trust, or given primitives providing the properties to parts of the system. For examples, the extensive work on partial broadcast channels, e.g. [13, 21, 22, 33], use a definition of non-equivocation exactly like strong non-equivocation; every process on the partial broadcast channel receive the same messages. Using this primitive, Fitzi and Maurer [21] consider the problem of achieving global broadcast, i.e. Byzantine broadcast, from such strongly non-equivocating partial broadcast channels. Given that strong non-equivocation is interchangeable with Byzantine broadcast, the problem of achieving global broadcast from partial broadcast channels can be rephrased as a problem of *extending* strong non-equivocation across systems; achieving system-wide strong non-equivocation from strongly non-equivocating partial broadcast channels.

[1]We are implicitly understanding this requirement to be only when $r'|w$ contains no consecutive duplicates, see Section 2.

4 WEAK NON-EQUIVOCATION

Key practical approaches to non-equivocation revolve around using cryptographic primitives in trusted modules, possibly hardware, making equivocation readily detectable by non-faulty processes. This is the approach taken by TrInc and A2M [11, 26]. In this case, strong non-equivocation is not an appropriate fault model, since the possibly faulty process employing the trusted module may still omit any message it chooses, including omitting messages to some but not all other processes within a single round. Thus, we define *weak non-equivocation*, where all processes still must report the same value to other processes for each process-sequence, but are allowed to omit messages to some subset of processes. Recall absence of a message is represented as $\perp$ in σ.

Definition 4.1 (Weak non-equivocation). *For any given process-sequence, all processes—faulty or not—report the same value to a subset of processes, and omit messages to the rest:*

$$\forall p, r, r' \in P \colon \forall w \in P^* \colon \sigma(r|p|w) \neq \perp \land \sigma(r'|p|w) \neq \perp \implies \sigma(r|p|w) = \sigma(r'|p|w)$$

Since w can be the empty string and is prefixed with the same p on both sides of the equality, this is equivalent to:

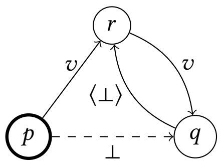
$$\forall r, r' \in P \colon \forall w \in P^+ \colon \sigma(r|w) \neq \perp \land \sigma(r'|w) \neq \perp \implies \sigma(r|w) = \sigma(r'|w)$$

Observe that the omission- and crash fault models are strict subspaces of the weak non-equivocation fault model as a faulty process can arbitrarily omit any message, and consequently also stop sending messages altogether from a given round onwards, thereby acting as if crashed See Figure 4 on page 9 for an illustration of the fault model spaces.

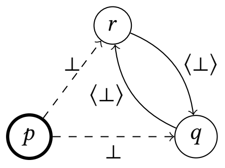
4.1 Algorithm for $t = 1$

To establish intuition about the general algorithm, we give in Algorithm 1 on page 5 a solution to Byzantine broadcast under weak non-equivocation when $t = 1$. (For the general case, see Section 5.) Note that for $n = 3$ this is the traditional Byzantine generals problem, which is *not* solvable in the Byzantine fault model [25].

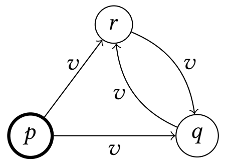
The algorithm proceeds as follows over two rounds. If a non-$\perp$ value is received in the first round (line 3), we decide on that value (line 4). Otherwise, the source must be faulty and (as $t = 1$) all other processes non-faulty. If any other non-faulty process received a value from the source, they correctly report that in the second round (line 5), and we decide on that value (line 6). Otherwise, no non-faulty process can have received a value in the first round, and so we decide on $\langle\perp\rangle$ as the value. The non-faulty processes all set the same value because they consider only the value sent by the source in the first round, and by weak non-equivocation (Definition 2.1), all who received the a value in round 1 received the same value. Altogether, we have correctness (for full proof, refer to the generalisation in Theorem 5.1):



(a) Report to only one



(b) Report to no one



(c) Act non-faulty

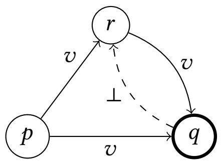
Figure 1: Three examples of $n = 3, t = 1$ executions, showing all possible actions a faulty source, p, can take under weak non-equivocation, parametric on the value of v.

Theorem 4.2. *In the weak non-equivocation fault model, interactive consistency is solvable for any $n > 1$ if $t \leq 1$.*

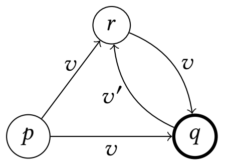
It is instructive to consider the different possibilities a faulty process has for lying under weak non-equivocation in the case $n = 3, t = 1$. Refer to Figures 1 and 2. In each case, it is immediately apparent to non-faulty nodes either who is faulty, or what the correct value should be. E.g., in Figure 1a, r knows to decide on v upon receiving it; q knows that p is faulty from the omission, thus deciding upon the value from r, which must be correct because $t = 1$. In Figure 2b, r knows that q is faulty as p could not have reported v' to q without violating weak non-equivocation (Definition 4.1).

4.2 Properties of weak non-equivocation

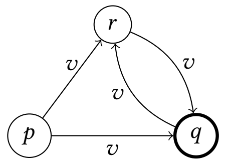
To reason about the general case, we introduce the formal device of *consistent majorities*. Consider a process p trying to decide on the value of a source process q. A consistent q-majority for a value v is a set Q of more than t processes, including q, who have consistently reported the value v to p, both for themselves and for each other.



(a) Don't relay



(b) Relay incorrectly



(c) Act non-faulty

Figure 2: Three examples of $n = 3, t = 1$ executions, showing all possible actions a faulty non-source, q, can take under weak non-equivocation, parametric on the values of v and v' and assuming that $v \neq v'$.

Algorithm 1 Algorithm for process p to set q'th value in vector

```
function WNE$_{t=1}(\sigma_p, p, q)$
    if $p = q$ then                                  ▷ $p$ is $q$
        return $\sigma_p(p)$
    else if $\sigma_p(p|q) \neq \perp$ then          ▷ $q$ reported a value to $p$
        return $\sigma_p(p|q)$
    else if $\exists r.\sigma_p(p|r|q) \neq \langle\perp\rangle$ then   ▷ Someone else told $p$ about $q$
        return $\sigma_p(p|r|q)$
    else                                             ▷ $q$ have reported $\perp$ to everyone
        return $\langle\perp\rangle$
    end if
end function
```

Definition 4.3 (consistent majority). *We write* $\mathsf{maj}^{\mathsf{q,v}}_{\mathsf{p,Q}}(t, \sigma_p, P)$ *and say that p has a consistent q-majority, when*

$$\forall w \in Q^* \colon |w| \leq t \colon \sigma_p(p|w|q) = v \wedge |Q| \geq t + 1 \wedge Q \subseteq P$$

Note that p having a consistent q-majority does not imply that any other process has a consistent q-majority, as p may have received different messages.

A consistent majority is related to the notion of a *quorum* but differs by, (1) the source process must be part of a consistent majority, and (2) the processes in a consistent majority Q also agree that everybody in Q agree on the same value, at least as far as p knows. The principle behind consistent majorities was described in [30].

Just from the message exchange σ, we can derive helpful relationships between the distribution of faulty processes and the existence of consistent majorities, *provided* we may assume $n > 2t$. Recall that $P = N \cup F$ is the set of n processes, where F is the set of at most t faulty processes, and N is the set of non-faulty processes. Let p, q and r be processes and let σ be a map of messages after at least $t + 1$ information exchange rounds according to Definition 2.1 and in the weak non-equivocation fault model (Definition 4.1).

Lemma 4.4. *If p is non-faulty, then all non-faulty processes will have a consistent majority for p's private value v_p:*

$$\forall q \in N \colon \exists Q \subseteq P \colon p \in N \implies \mathsf{maj}^{\mathsf{p,v_p}}_{\mathsf{q,Q}}(t, \sigma_q, P)$$

Proof. By Definition 2.1, all non-faulty processes correctly forward values each round. So p must have reported v_p to the set of non-faulty processes N in the first round, and all processes in N must have reported v_p to each other in all following rounds. Then N must be a consistent majority for v_p, noting $|N| > t$. □

Lemma 4.5. *If a process p does not have a consistent q-majority then q must be faulty:*

$$\forall p, q \in P \colon (\forall v \in V \colon \forall Q \subseteq P \colon \neg\mathsf{maj}^{\mathsf{q,v}}_{\mathsf{p,Q}}(t, \sigma_p, P)) \implies q \in F$$

Proof. Contraposition of Lemma 4.4. □

Lemma 4.6. *If p has a consistent q-majority for v, then it cannot also have a consistent q-majority for a distinct value v':*

$$\forall v, v' \in V\colon \forall p, q \in P\colon \exists Q, R \subseteq P\colon$$
$$\mathsf{maj}^{\mathsf{q,v}}_{\mathsf{p,Q}}(t, \sigma_p, P) \wedge \mathsf{maj}^{\mathsf{q,v'}}_{\mathsf{p,R}}(t, \sigma_p, P) \implies v = v'$$

Proof. By contradiction. Assume that for some process p, $\mathsf{maj}^{\mathsf{q,v}}_{\mathsf{p,Q}}(t, \sigma_p, P)$, $\mathsf{maj}^{\mathsf{q,v'}}_{\mathsf{p,R}}(t, \sigma_p, P)$ and $v \neq v'$ holds. By definition of consistent majority (Definition 4.3), any set Q s.t. $\mathsf{maj}^{\mathsf{q,v}}_{\mathsf{p,Q}}(t, \sigma_p, P)$ must include q, so it must be the case that $\sigma(p|q) = v$. But then by $\mathsf{maj}^{\mathsf{q,v'}}_{\mathsf{p,R}}(t, \sigma_p, P)$ it must be the case that $\sigma(p|q) = v'$, which contradicts $v \neq v'$. □

Lemma 4.7. *No consistent majority for $\perp$ can exist:*

$$\forall p, q \in P\colon \forall v \in V\colon (\exists Q \subseteq P\colon \mathsf{maj}^{\mathsf{q,v}}_{\mathsf{p,Q}}(t, \sigma_p, P)) \implies v \neq \perp$$

Proof. By contradiction. Assume that there exists a Q s.t. $\mathsf{maj}^{\mathsf{q},\perp}_{\mathsf{p,Q}}(t, \sigma_p, P)$. By Definition 4.3, $|Q| > |F|$, so there must exist a non-faulty process in Q, i.e. a non-faulty process that has reported $\perp$. But, by Definition 2.1, non-faulty processes never report $\perp$. □

Lemma 4.8. *If non-faulty processes p and q each have a consistent r-majority, then those must be for the same value:*

$$\forall p, q \in N\colon \forall v, v' \in V\colon$$
$$(\exists Q, R \subseteq P\colon \mathsf{maj}^{\mathsf{r,v}}_{\mathsf{p,Q}}(t, \sigma_p, P) \wedge \mathsf{maj}^{\mathsf{r,v'}}_{\mathsf{q,R}}(t, \sigma_q, P)) \implies v = v'$$

Proof. *By contradiction.* Assume that p and q have consistent r-majorities for respectively different values v and v'. By Definition 4.3, p having an consistent r-majority for v implies that $\sigma(p|r) = v$, and q having an consistent r-majority for v' implies $\sigma(q|r) = v'$. By weak non-equivocation (Definition 4.1), $\sigma(p|r) \neq \sigma(q|r)$ implies that either v or v' is $\perp$. But by Lemma 4.7, no consistent majority for $\perp$ can exist: a contradiction. □

5 SOLVING INTERACTIVE CONSISTENCY UNDER WEAK NON-EQUIVOCATION

To solve interactive consistency, it is sufficient to define an algorithm that achieves agreement, validity and termination for one source process, and then run that algorithm for each process in the system [19, 30]. Assuming $n > 2t$ and weak non-equivocation, we give such an algorithm in Algorithm 2, which is a generalisation of Algorithm 1 and an adaptation of the algorithm by Pease, Shostak, and Lamport [30], which assumes the Byzantine fault model. The key differences are the addition of $\perp$ and $\langle\perp\rangle$, the required size of consistent majorities in Definition 4.3, and the size of the set in line 20, which is adapted to fit with a fault tolerance of $n > 2t$.

Algorithm 2 relies on two insights. (1) If a process cannot find a consistent majority, then the source process must be faulty (Lemma 4.5). (2) After $t + 1$ rounds, if the source process is faulty, we can treat the other processes reports about the source process as their private values in a new instance of interactive consistency; write $\widehat{\sigma}$ for the restriction of σ where we remove any mention of the source. This $\widehat{\sigma}$ will have t message rounds regarding these values. Since the *faulty* source process is excluded, $\widehat{\sigma}$ can be used to recursively find the values reported to the rest of the processes,

Algorithm 2 Algorithm for process p to set source process q's value in an interactive consistency vector

```
function PSL($\sigma_p, n, t, p, q, P$)
1:  if $p = q$ then                                   ▷ $p$'s own private value
2:      if $\sigma_p(p) = \perp$ then        ▷ $\perp \notin V$, so choose default $\langle\perp\rangle$
3:          return $\langle\perp\rangle$
4:      else
5:          return $\sigma_p(p)$
6:      end if
7:  end if
8:  if $\exists Q \subseteq P\colon \mathsf{maj}^{\mathsf{q,v}}_{\mathsf{p,Q}}(t, \sigma_p, P)$ then     ▷ Consistent majority for $v$
9:      return $v$
10: end if   ▷ No consistent majority; $q$ is faulty (Lemma 4.5)
11: $\widehat{\sigma_p} \leftarrow \{\}$                          ▷ Start constructing $\widehat{\sigma_p}$
12: $\widehat{P} \leftarrow P \setminus \{q\}$                     ▷ Exclude $q$ from $\widehat{\sigma_p}$
13: for all $w \in \widehat{P}^*$, s.t. $p|w|q$ is in the domain of $\sigma_p$ do
14:     $\widehat{\sigma_p}(p|w) \leftarrow \sigma_p(p|w|q)$
15: end for                                       ▷ Done constructing $\widehat{\sigma_p}$
16: $\widehat{\mathsf{vec}} \leftarrow [\ ]$              ▷ Constructing $\widehat{\mathsf{vec}}$: IC vector of $\widehat{\sigma_p}$
17: for all $r \in \widehat{P}$ do
18:     $\widehat{\mathsf{vec}}[r] \leftarrow$ PSL($\widehat{\sigma_p}, n-1, t-1, p, r, \widehat{P}$)
19: end for                                       ▷ Done constructing $\widehat{\mathsf{vec}}$
20: if $\exists v \neq \langle\perp\rangle : |\{r \mid \widehat{\mathsf{vec}}[r] = v\}| \geq t$ then
21:     return $v$       ▷ Min. one non-faulty process received $v$
22: else
23:     return $\langle\perp\rangle$
24: end if
end function
```

by either finding a consistent majority for them or recursively applying the same method. In the base case, there is only one faulty process left, in which case Algorithm 2 behaves like Algorithm 1.

We proceed to prove correctness of Algorithm 2.

Theorem 5.1. *Algorithm 2 is correct.*

Proof sketch (full proof in Appendix C). **Validity**. Assume a non-faulty source p; we must show that for any non-faulty process, Algorithm 2 returns the private value v_p. Because p is non-faulty, all non-faulty processes have a consistent p-majority for v_p (Lemma 4.4), and cannot have a consistent p-majority for another value (Lemma 4.6). Thus Algorithm 2 will return v_p in line 9. **Agreement**. We must show that for all non-faulty processes, Algorithm 2 returns the same value. Consider a source p. If p is non-faulty, agreement follows trivially from validity (line 9), so assume p faulty. We proceed by induction on t. For the base case, $t = 1$, p either reports a value to some processes, in which case all others know to trust a forwarded value (line 21), or not report at value at all, in which case everybody agrees that this is the case (line 23), and choose a default value. For the induction case, consider two arbitrary non-faulty processes, and proceed by cases on which have a consistent p-majority. If they both do, then it is for the same value by Lemma 4.8. If neither has a consistent p-majority, then both know that p is faulty, and run Algorithm 2 recursively with p, obtaining a correct interactive consistency vector by IH. They both run a majority function on this vector (line 20), arriving at the same result. If one has a consistent majority, and the other does not, then the latter must have a recursive vector with a majority of the same value as that of the consistent majority by Definition 4.3 and IH (see also Lemma C.1 in Appendix C). **Termination**. Immediate from the observation that message exchange is bounded at $t + 1$; the domain of σ is then finite; the domain of σ_p is then also finite; and it is now obvious that all loops in Algorithm 2 terminates. □

Theorem 5.1 shows that interactive consistency can be solved for $n > 2t$ under weak non-equivocation. However, the solution is not efficient, requiring $\Theta(n^{t+1})$ messages to construct the necessary σ. Algorithms that solve interactive consistency in a polynomial number of messages, e.g. [16] exists for the Byzantine fault model, and we leave a polynomial algorithm for the weak non-equivocation fault model as future work.

6 TIGHTNESS

Having proved interactive consistency possible for $n > 2t$ under weak non-equivocation, we now show that it is a tight lower bound.

Theorem 6.1. *Given any finite number of rounds, k, and $n \leq 2t$, no synchronous deterministic algorithm exists which, given a σ in the weak non-equivocation fault model, always finds an interactive consistency vector with the properties of agreement, validity and termination.*

Proof. By contradiction. Assume a deterministic algorithm, A, where any process p can run $A(p, \sigma_p, q)$ and obtain a value for q fulfilling the requirements of agreement and validity. Without loss of generality assume that $n = 2t$ and that each run continues for k rounds, where $k \geq t + 1$. Partition all processes into sets O, Q, R and S, s.t. $|O| = \lfloor \frac{t}{2} \rfloor$, $|Q| = \lceil \frac{t}{2} \rceil$, $|R| = \lceil \frac{t}{2} \rceil$ and $|S| = \lfloor \frac{t}{2} \rfloor$. Note that all processes in any two partitions can be simultaneously faulty, except for Q and R as their combined size may be greater than t.

Let the source process be $p \in O$. We construct four runs with different faulty partitions, such that some non-faulty process will be unable to distinguish two specific runs from each other, leading to a contradiction if the algorithm achieves both validity and agreement. Let v_1 and v_2 be *distinct* values not equal to $\perp$ or $\langle\perp\rangle$. For ease of notation let $A(Q, \sigma_{Run1}, p)$ denote the value each process in Q will decide for p in run 1. We construct runs such that $A(Q, \sigma_{Run1}, p) = v_1$ by validity, and $A(Q, \sigma_{Run1}, p) = A(Q, \sigma_{Run3}, p)$ by determinism of A. Similarly we will construct the runs such that $A(R, \sigma_{Run2}, p) = v_2$ by validity, and $A(R, \sigma_{Run2}, p) = A(R, \sigma_{Run4}, p)$ by determinism of A. Lastly we will construct the runs such that $A(S, \sigma_{Run3}, p) = A(S, \sigma_{Run4}, p)$ by determinism of A, ultimately showing that S must decide both v_1 and v_2 which is impossible as $v_1 \neq v_2$.

We construct the runs as follows. Let the processes in O broadcast to the other partitions in round 2 exactly as p broadcast in round 1.

Run 1 (The processes of R and S are faulty):
- p broadcasts the value v_1 in the first round.
- In the second round, the processes in R broadcast the value v_2 and the processes in S broadcast $\langle\perp\rangle$.
- In the third round the processes in R broadcast the value v_2 and the processes in S broadcast $\langle\perp\rangle$, as the respective values received from O.
- All processes correctly report all other messages.

By validity, all processes in Q decide v_1.

Run 2 (The processes of Q and S are faulty):
- p broadcasts the value v_2 in the first round.
- In the second round, the processes in Q broadcast the value v_1 and the processes in S broadcast $\langle\perp\rangle$.
- In the third round the processes in Q broadcast the value v_1 and the processes in S broadcast $\langle\perp\rangle$, as the respective values received from O.
- All processes correctly report all other messages.

By validity, all processes in R decide v_2.

Run 3 (The processes of O and R are faulty):
- In the first round p broadcasts the value v_1 to all except for the processes in partition S, resulting in S recording $\perp$.
- In the second round the processes in R broadcast v_2, and O broadcasts v_1 to all but S, resulting in S recording $\perp$.
- In the third round the processes in R broadcast the value v_2 as the values received from O.
- All processes correctly report all other messages.

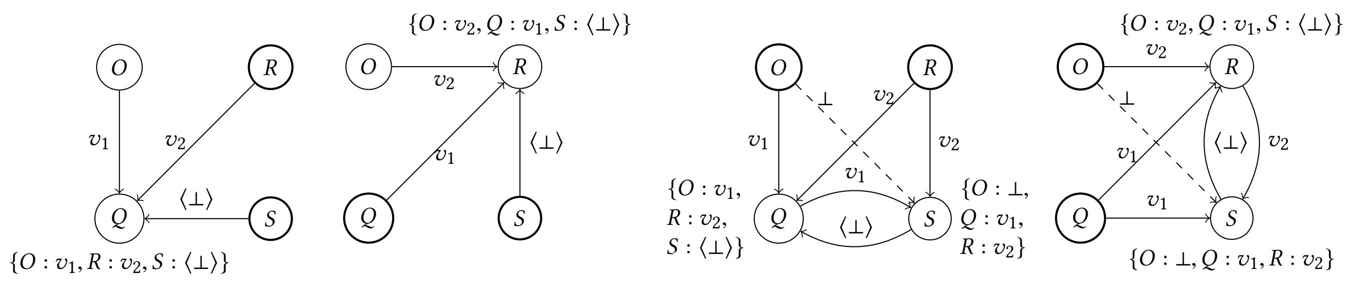
For the processes in Q, this run is indistinguishable from run 1: in both runs, they have received v_1 from O, v_2 from R, and $\langle\perp\rangle$ from S by the end of round 2, and all partitions will act the same for the remaining rounds. By determinism, Q should decide v_1 to be the value for p. By agreement, S must then decide v_1 to be the value for p.

Run 4 (The processes of O and Q are faulty):
- In the first round p broadcasts the value v_2 to all except for the processes in partition S, resulting in S recording $\perp$.
- In the second round processes in Q broadcast v_1, and O broadcasts v_2 to all but S, resulting in S recording $\perp$.
- In the third round the processes in Q broadcast the value v_1 as the values received from O.
- All processes correctly report all other messages.

For the processes in R, this run is indistinguishable from run 2: in both runs, they have received v_2 from O, v_1 from Q, and $\langle\perp\rangle$ from S by the end of round 2, and all partitions will act the same for the remaining rounds. By determinism, R should decide v_2 to be the value for p. By agreement, S must then decide v_2 to be the value for p. But for the processes in S, this run is indistinguishable from run 3: in both runs, they have received nothing ($\perp$) from any process in O, v_1 from all processes in Q, and v_2 from all processes in R by the end of round 2, and all partitions will act the same in the two runs for the remaining rounds. So, by determinism all processes in S should decide v_1 to be the value for p, leading to the contradiction that the processes in S should decide both v_1 and v_2 for distinct v_1 and v_2. □

Note that this proof uses determinism only to restrict that non-faulty processes decides the same value under indistinguishable runs. This restriction may be implied by the validity and agreement properties, making the proof have no requirement of determinism. We leave a more thorough investigation of this as future work.

The weak non-equivocation fault model is, to the best of our knowledge, the only fault model with this lower bound fault-tolerance for interactive consistency and broadcast in the synchronous model; the Byzantine fault model has $n > 3t$ and all


O
R
Q
S
v_1
v_2
$\langle\perp\rangle$
$\{O : v_1, R : v_2, S : \langle\perp\rangle\}$
$\{O : v_2, Q : v_1, S : \langle\perp\rangle\}$
$\perp$
$\{O : v_1, R : v_2, S : \langle\perp\rangle\}$
$\{O : \perp, Q : v_1, R : v_2\}$
$\{O : v_2, Q : v_1, S : \langle\perp\rangle\}$
$\{O : \perp, Q : v_1, R : v_2\}$

(a) Run 1 **(b) Run 2** **(c) Run 3** **(d) Run 4**

Figure 3: Runs described in the proof of Theorem 6.1. Faulty partitions are bold. Arrows are selected messages sent in round 2. The source process p is in O, so the arrows from O are messages in both round 1 and 2. For selected partitions, the messages received by end of round 2 are annotated. Note how partition Q receives the same messages in run 1 and 3, partition R receives the same messages in run 2 and 4, and partition S receives the same messages in run 3 and 4.

others has $n > t$ (see Table 1). This uniqueness confirms formally the folklore knowledge that equivocation is the most critical of the Byzantine fault and what necessitates the triple replication of processes in Byzantine agreement problems.

Considering the fault tolerance of general omission algorithms ($n > t$), Theorem 6.1 also suggests that the type of faults that separates general omission from weak non-equivocation, i.e. lying without equivocating, is what necessitates anything but the least amount of replication. This hypothesis is also backed up by the fault tolerance of the authenticated Byzantine fault model ($n > t$), where faulty processes are prevented from lying *at all* about (but not omitting) the messages of non-faulty processes.

A proof sketch of the tightness of the bound of $t + 1$ rounds is simply showing that weak non-equivocation encompasses the crash fault model. As it is known that the broadcast agreement cannot be solved for less than $t + 1$ rounds in the crash fault model [2], it follows that the problem requires at least as many rounds in weak non-equivocation. A formal proof can be found in Appendix B.

7 OTHER MODELS

In this section, we relate weak non-equivocation to (1) the problem of crusader agreement as defined by Dolev [15], and (2) the authenticated Byzantine fault model as defined by Pease et al. [30].

7.1 Weak non-equivocation and Crusader Agreement

From weak non-equivocation to crusader agreement. Weak non-equivocation (Definition 4.1) implies a trivial solution to the problem of *crusader agreement* [15]. Crusader agreement is an agreement problem weaker than Byzantine broadcast: non-faulty receiving processes must decide a value in agreement with all other non-faulty processes *unless* the receiving process *knows* that the sending process is faulty. Formally, processes in the crusader agreement problem decides a value for the process p's private value v_p, with these properties:

Crusader agreement: All non-faulty processes decide the same value unless they know that p is faulty.[2]

Validity: If process p is non-faulty, then all non-faulty processes decide v_p.

Termination: All non-faulty processes must eventually decide on a value, or know that p is faulty.

These properties are *exactly true* for any message in the weak non-equivocation fault model: for some message, a non-faulty process either receives a value from the sending process or it receives nothing; if the receiving process receives nothing, then it knows that the sending process is faulty; if the process receives a value, then it is guaranteed that every other non-faulty process that also received a value, have received the same value. Thus, a procedure where the non-faulty processes (1) decides on the received value if they receive one, and (2) otherwise sets the source process a faulty, solves crusader agreement under weak non-equivocation.

From Crusader Agreement to Weak non-equivocation. Suppose there exists an oracle producing crusader agreement upon request, and that we use this oracle to produce crusader agreement for the value of some source process p. After finishing deciding such a value, all processes will (knowingly) be either faulty; non-faulty and having decided a common value; or non-faulty and knowing that p is faulty. This is exactly the same three states a process will find itself in after p has broadcast a value under weak non-equivocation: for some v, all correct processes q will have either $\sigma_q(p) = v$ or $\sigma_q(p) = \perp$; in the latter case, q knows for a fact that p is faulty.

Dolev [15] showed that crusader agreement has a fault tolerance of $n > 3t$ in the pure Byzantine fault model, but that it only requires a constant of 2 information exchange rounds. Dolev's crusader agreement algorithm works by using a purifying function much like Definition 4.3[3] to agree on a value or know that the source is faulty. Assuming that $n > 3t$, we can use Dolev's crusader agreement

[2]"know that p is faulty" is a slightly imprecise definition. For more precision, the phrase "can prove that if the receiving process itself is non-faulty, then p must be faulty" can be used.

[3]The most notable difference between Definition 4.3 and Dolevs purifying function is that the purifying function does not require the source process to be part of the consistent majority, and, of course, that the consistent majority must be of size at least $2t + 1$, due to the lower fault tolerance.

algorithm as a primitive to map any σ with $t + 2$ rounds in the Byzantine fault model into a σ with $t + 1$ rounds in the weak non-equivocation fault model. The mapping proceeds by using the extra (last) round as the second round in Dolev's algorithm, thus achieving crusader agreement for the values transmitted in round $t + 1$. These values can then be used to achieve crusader agreement for the values transmitted in round t, and so on.

Altogether, it seems any algorithm that operates correctly given a crusader agreement primitive, can replace that primitive with an assumption of weak non-equivocation. Examples of algorithms using such a primitive can be found in [6, 17, 29].

7.2 Comparison to the authenticated Byzantine fault model

In this section, we examine the weak non-equivocation fault model's relationship with the authenticated Byzantine fault model. We refer to the authenticated Byzantine fault model as defined in Pease et al. [30], which says that a faulty process cannot lie (but can *omit* messages) about the values of non-faulty processes:

$$\forall p \in N \colon \forall w', w \in P^* \colon \sigma(w'|p|w) = \sigma(p|w) \vee \perp .$$

Given that interactive consistency problem is solvable in the authenticated Byzantine fault model for $n > t$, and that the same problem is solvable for weak non-equivocation for a tight $n > 2t$, one would expect the authenticated fault model is the stronger fault model. However, we shall see that whereas the authenticated fault model and weak non-equivocation obviously have an intersection, they also have a non-empty set difference in either direction. As Clement et al. [12] showed in the asynchronous setting both authentication and non-equivocation is needed to achieve a fault tolerance of $n > 2t$ for agreement problems, this relationship between the fault models is not entirely surprising. See Figure 4 for a Venn diagram of spaces of σ's that exist in different fault models.

We demonstrate that (1) there exists σ's where algorithms for weak non-equivocation can solve agreement, but algorithms for the authenticated Byzantine fault model cannot; (2) there exists σ's where algorithms for the authenticated Byzantine fault model can solve agreement, but algorithms for weak non-equivocation cannot; and (3) there exists σ's where algorithms for either fault model can solve agreement, but algorithms for general omission cannot. Examples of each follow here:

(1) For any such σ, no process can equivocate, but some process must lie about the value of a non-faulty process. Let $n = 3, t = 1, p, r \in N$ and $q \in F$. Let r be the source process, and broadcast v in the first round. Then let q proceed to report that they received u in the next round. Any algorithm for authenticated Byzantine faults will wrongly conclude that r is faulty, as only fault processes can have their value spoofed, while any algorithm for weak non-equivocation will correctly conclude q is faulty, as r has not equivocated.

(2) For any such σ, no process can lie about the value of a non-faulty process but some process must equivocate. Let $n = 3, t = 1, p, q \in N$ and $r \in F$. Let r be the source process, and send a v to p, and a u to q in the first round, and let p and q proceed to report their received values. Any algorithm for authenticated Byzantine faults will correctly conclude that r is faulty, as different values can only exists for faulty processes, while any algorithm for weak non-equivocation will wrongly conclude the other non-faulty process is, in fact, faulty, as it "believes" that r is unable to equivocate.

(3) For any such σ, the following three properties must hold: (I) Some process must lie, so that general omission algorithms cannot solve agreement. (II) No process can equivocate, so that weak non-equivocation algorithms can solve agreement. (III) No faulty process can lie about the values of non-faulty processes so that authenticated Byzantine algorithms can solve agreement. From (I), (II) and (III), we can deduce that $t \geq 2$, and, by the lower bound fault tolerance of weak non-equivocation, that $n \geq 5$: Let $n = 5, t = 2, p, q, r \in N$ and $s, x \in F$. Let s be the source process, and send v to p in the first round and nothing, $\perp$, to q and r. Let x act as if it received u, but otherwise, let every process report correctly for the proceeding rounds[4]. Both weak non-equivocation and authenticated Byzantine algorithms will deduce that s is faulty, and that there is no majority for any specific value, and so decide some default value. Meanwhile, for a general omission algorithm, q and r know that s is faulty, but have received the contradicting values v and u, and will thus behave in an unspecified manner.

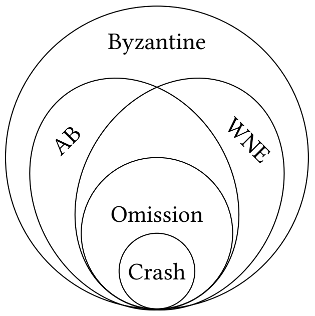


Figure 4: The relationship between spaces of σs described by various fault models. Abbreviations: AB – authenticated Byzantine, WNE – weak non-equivocation. Not pictured is strong non-equivocation, which overlaps with but is not contained in the crash fault model, yet contained in the weak non-equivocation fault model.

8 FUTURE WORK

Our results on fault-tolerance together with that of [12] suggests that the property of *transferable authentication* alone can bridge the gap between an agreement algorithm with non-equivocation in a synchronous system and an asynchronous system with weakened termination. An examination of this result, and in which other system models this translation is valid, could lead to more efficient translations between synchronous and asynchronous algorithms.

Given that the strong non-equivocation fault model is overlapping with the crash fault model, and that it implies a trivial

[4]For the sake of argument, let r and q receive u from x before they receive v from p, to prevent early stopping general omission algorithms from deciding v before receiving the contradicting u value

solution to Byzantine agreement as seen in Section 3, it seems that FLP-impossibility may not apply in systems using a primitive providing strong non-equivocation. This opens up questions of whether any weakening of termination, determinism or asynchrony is strictly needed in asynchronous agreement systems that use strong non-equivocation primitives such as partial broadcast channels, e.g. [13, 21, 22, 33].

9 CONCLUSION

In this paper, we addressed and formalised the notion of *non-equivocation* in synchronous agreement protocols. We have proposed two different notions of non-equivocation: *strong* and *weak*. Both require faulty processes to not "lie differently", however, weak non-equivocation allows faulty processes to selectively omit messages to some participants, where strong non-equivocation does not. We defined both formally as fault models for synchronous agreement protocols with reliable channels, and we showed how the two models yield distinct round- and fault tolerance-bounds for agreement: 1 round, $n > t$ for strong non-equivocation; and $t + 1$ rounds, $n > 2t$ for weak non-equivocation. This makes weak non-equivocation the only fault model with a lower bound on fault tolerance of $n > 2t$ for broadcast agreement and interactive consistency, thus confirming formally the folklore knowledge that equivocation is somehow the most critical of the Byzantine faults. Finally, we have shown how the weak and strong non-equivocation fault models relate to well-known agreement problems: strong non-equivocation corresponds to Byzantine broadcast and weak non-equivocation to crusader agreement.

REFERENCES

[1] Ittai Abraham, Srinivas Devadas, Danny Dolev, Kartik Nayak, and Ling Ren. 2017. Efficient Synchronous Byzantine Consensus. *arXiv:1704.02397 [cs]* (Sept. 2017). arXiv:cs/1704.02397 http://arxiv.org/abs/1704.02397

[2] Marcos Kawazoe Aguilera and Sam Toueg. 1999. A Simple Bivalency Proof That T-Resilient Consensus Requires T+1 Rounds. *Inform. Process. Lett.* 71, 3 (1999), 155–158.

[3] Chagit Attiya, Danny Dolev, and Joseph Gil. 1984. Asynchronous Byzantine Consensus. In *Proceedings of the Third Annual ACM Symposium on Principles of Distributed Computing (PODC '84)*. Association for Computing Machinery, New York, NY, USA.

[4] Michael Backes, Fabian Bendun, Ashish Choudhury, and Aniket Kate. 2014. Asynchronous MPC with a Strict Honest Majority Using Non-Equivocation. In *Proceedings of the 2014 ACM Symposium on Principles of Distributed Computing (PODC '14)*. Association for Computing Machinery, Paris, France, 10–19.

[5] Michael Ben-Or. 1983. Another Advantage of Free Choice (Extended Abstract): Completely Asynchronous Agreement Protocols. In *Proceedings of the Second Annual ACM Symposium on Principles of Distributed Computing (PODC '83)*. Association for Computing Machinery, Montreal, Quebec, Canada, 27–30.

[6] Malte Borcherding. 1997. Byzantine Agreement with Limited Authentication. In *Safe Comp 96*, Erwin Schoitsch (Ed.). Springer, London, 404–413.

[7] Gabriel Bracha and Sam Toueg. 1985. Asynchronous Consensus and Broadcast Protocols. *J. ACM* 32, 4 (Oct. 1985), 824–840.

[8] Miguel Castro and Barbara Liskov. 1999. Practical Byzantine Fault Tolerance. In *Proceedings of the Third Symposium on Operating Systems Design and Implementation (OSDI '99)*. USENIX Association, New Orleans, Louisiana, USA, 173–186.

[9] Tushar Deepak Chandra, Vassos Hadzilacos, and Sam Toueg. 1996. The Weakest Failure Detector for Solving Consensus. *J. ACM* 43, 4 (July 1996), 685–722.

[10] Tushar Deepak Chandra and Sam Toueg. 1996. Unreliable Failure Detectors for Reliable Distributed Systems. *J. ACM* 43, 2 (March 1996), 225–267.

[11] Byung-Gon Chun, Petros Maniatis, Scott Shenker, and John Kubiatowicz. 2007. Attested Append-Only Memory: Making Adversaries Stick to Their Word. In *Proceedings of Twenty-First ACM SIGOPS Symposium on Operating Systems Principles (SOSP '07)*. Association for Computing Machinery, Stevenson, Washington, USA, 189–204.

[12] Allen Clement, Flavio Junqueira, Aniket Kate, and Rodrigo Rodrigues. 2012. On the (Limited) Power of Non-Equivocation. In *Proceedings of the 2012 ACM Symposium on Principles of Distributed Computing (PODC '12)*. Association for Computing Machinery, Madeira, Portugal, 301–308.

[13] Jeffrey Considine, Matthias Fitzi, Matthew Franklin, Leonid A. Levin, Ueli Maurer, and David Metcalf. 2005. Byzantine Agreement Given Partial Broadcast. *J. Cryptology* 18, 3 (July 2005), 191–217.

[14] Miguel Correia, Giuliana S. Veronese, and Lau Cheuk Lung. 2010. Asynchronous Byzantine Consensus with 2f+1 Processes. In *Proceedings of the 2010 ACM Symposium on Applied Computing (SAC '10)*. Association for Computing Machinery, Sierre, Switzerland, 475–480.

[15] Danny Dolev. 1981. *The Byzantine Generals Strike Again*. Technical Report. Stanford University, Stanford, CA, USA.

[16] Danny Dolev, Michael J. Fischer, Rob Fowler, Nancy A. Lynch, and H. Raymond Strong. 1982. An Efficient Algorithm for Byzantine Agreement without Authentication. *Information and Control* 52, 3 (March 1982), 257–274.

[17] Danny Dolev and Avi Wigderson. 1983. On the Security of Multi-Party Protocols in Distributed Systems. In *Advances in Cryptology*. Springer US, Boston, MA.

[18] Cynthia Dwork, Nancy Lynch, and Larry Stockmeyer. 1988. Consensus in the Presence of Partial Synchrony. *J. ACM* 35, 2 (April 1988), 288–323.

[19] Michael J. Fischer. 1983. The Consensus Problem in Unreliable Distributed Systems (a Brief Survey). In *Foundations of Computation Theory (Lecture Notes in Computer Science)*, Marek Karpinski (Ed.). Springer Berlin Heidelberg, 127–140.

[20] Michael J. Fischer, Nancy A. Lynch, and Michael S. Paterson. 1985. Impossibility of Distributed Consensus with One Faulty Process. *J. ACM* 32, 2 (April 1985), 374–382.

[21] Mattias Fitzi and Ueli Maurer. 2000. From Partial Consistency to Global Broadcast. In *Proceedings of the Thirty-Second Annual ACM Symposium on Theory of Computing (STOC '00)*. Association for Computing Machinery, Portland, Oregon, USA, 494–503.

[22] Alexander Jaffe, Thomas Moscibroda, and Siddhartha Sen. 2012. On the Price of Equivocation in Byzantine Agreement. In *Proceedings of the 2012 ACM Symposium on Principles of Distributed Computing (PODC '12)*. Association for Computing Machinery, Madeira, Portugal, 309–318.

[23] Muhammad Samir Khan, Syed Shalan Naqvi, and Nitin H. Vaidya. 2019. Exact Byzantine Consensus on Undirected Graphs under Local Broadcast Model. In *Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing (PODC '19)*. Association for Computing Machinery, Toronto ON, Canada, 327–336.

[24] Muhammad Samir Khan and Nitin H. Vaidya. 2019. Byzantine Consensus under Local Broadcast Model: Tight Sufficient Condition. *arXiv:1901.03804 [cs]* (Jan. 2019). arXiv:cs/1901.03804 http://arxiv.org/abs/1901.03804

[25] Leslie Lamport, Robert Shostak, and Marshall Pease. 1982. The Byzantine Generals Problem. *ACM Trans. Program. Lang. Syst.* 4, 3 (July 1982), 382–401.

[26] Dave Levin, John R. Douceur, Jacob R. Lorch, and Thomas Moscibroda. 2009. TrInc: Small Trusted Hardware for Large Distributed Systems. In *Proceedings of the 6th USENIX Symposium on Networked Systems Design and Implementation (NSDI'09)*. USENIX Association, Boston, Massachusetts, 1–14.

[27] Chuanyou Li, Michel Hurfin, Yun Wang, and Lei Yu. 2016. Towards a Restrained Use of Non-Equivocation for Achieving Iterative Approximate Byzantine Consensus. In *2016 IEEE International Parallel and Distributed Processing Symposium (IPDPS'16)*. IEEE, 710–719.

[28] Nancy A. Lynch. 1996. *Distributed Algorithms*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

[29] Stephen R. Mahaney and Fred B. Schneider. 1985. Inexact Agreement: Accuracy, Precision, and Graceful Degradation. In *Proceedings of the Fourth Annual ACM Symposium on Principles of Distributed Computing (PODC '85)*. Association for Computing Machinery, Minaki, Ontario, Canada, 237–249.

[30] M. Pease, R. Shostak, and L. Lamport. 1980. Reaching Agreement in the Presence of Faults. *J. ACM* 27, 2 (April 1980), 228–234.

[31] Michael O. Rabin. 1983. Randomized Byzantine Generals. In *24th Annual Symposium on Foundations of Computer Science (SFCS'83)*. 403–409.

[32] Philippe Raipin Parvedy and Michel Raynal. 2003. Uniform Agreement Despite Process Omission Failures. In *2003 IEEE International Parallel and Distributed Processing Symposium (IPDPS'03)*. IEEE.

[33] D. V. S. Ravikant, V. Muthuramakrishnan, V. Srikanth, K. Srinathan, and C. Pandu Rangan. 2004. On Byzantine Agreement over (2,3)-Uniform Hypergraphs. In *18th International Symposium on Distributed Computing (DISC'18)*, Rachid Guerraoui (Ed.). Springer, Berlin, Heidelberg, 450–464.

[34] Michel Raynal. 2002. Consensus in Synchronous Systems: A Concise Guided Tour. In *2002 Pacific Rim International Symposium on Dependable Computing, 2002 (PRDC'02)*. 221–228.

[35] Michel Raynal. 2010. *Fault-Tolerant Agreement in Synchronous Message-Passing Systems*. Morgan & Claypool.

Appendices

A STRONG NON-EQUIVOCATION CORRECTNESS PROOF

Algorithm 3 Algorithm for process p to set q'th value in vector, under the strong non-equivocation fault model

```
function SNE($\sigma_p$, $p$, $q$)
    if $p = q$ then
        return $\sigma_p(p)$
    else
        return $\sigma_p(p|q)$
    end if
end function
```

Lemma A.1. *Algorithm 3 guarantees validity for $n > t$, in the strong non-equivocation fault model.*

Proof. We must prove that for any non-faulty process, p, running Algorithm 3 for some other non-faulty process q, Algorithm 3 will produce q's private value v_q.
Assume $n > t + 1$, since validity is trivially ensured for $n = t + 1$.

Let p and q be non-faulty processes. By the Definition 2.1 we know that $\sigma(p|q) = \sigma(q)$, for all non-faulty q. □

Lemma A.2. *Algorithm 3 guarantees agreement for $n > t$, in the strong non-equivocation fault model.*

Proof. We must prove that for any two non-faulty processes, p and r, running Algorithm 3 for some third process q, Algorithm 3 will produce the same result for p and r.

Proof by contradiction. Let p and r be non-faulty processes. Assume that p and r get different results from Algorithm 3, for some other process q. Then $\sigma(p|q) \neq \sigma(r|q)$, but this contradicts Definition 3.1. □

Lemma A.3. *Algorithm 3 guarantees termination for any terminating protocol.*

Proof. The proof follows trivially from the fact that the protocol terminates, which implies that a lookup in σ terminates. □

B TIGHTNESS OF ROUNDS FOR WEAK NON-EQUIVOCATION

We will now show that $t + 1$ rounds is the least amount of rounds needed to solve interactive consistency in the weak non-equivocation fault model. We do this by arguing that the weak non-equivocation fault model completely encompasses the crash fault model (i.e. any σ under the crash fault model can exist under the weak non-equivocation fault model). Since it is well-known, e.g. from [2], that agreement problems in a crash fault model cannot be solved with less than $t + 1$ information exchange rounds, it must follow that agreement problems in weak non-equivocation must use at least as many. And with the proof of Theorem 5.1, we know that interactive consistency is solvable using $t + 1$ information exchange rounds, showing that $t + 1$ rounds is a tight lower bound.

That weak non-equivocation encompass crash faults follow trivially from the fact that faulty processes can arbitrarily not report anything and thus can act exactly as a crashed process by not reporting anything from some time and until the protocol has terminated. A formal proof follows:

Theorem B.1. *Any σ that can exist under the crash-failure model must also be possible under weak non-equivocation.*

Proof. By contradiction. Assume that there exists a σ in the crash-failure model that cannot exist under weak non-equivocation, and denote such a σ as σ_{crash}. In σ_{crash} there must exist two processes p, q, a (possibly empty) string w and value (or $\perp$) v such that $\sigma_{crash}(q|p|w) = v$, and no σ restricted under weak non-equivocation can produce the same v given the input $q|p|w$:

$$\forall \sigma_{WNE} \in WNE \colon \sigma_{WNE}(q|p|w) \neq v$$

where WNE is the space of possible σ's restricted under weak non-equivocation.

Now we construct a contradiction by cases on the faultiness of p:

Case 1: (p is non-faulty): as p is non-faulty, v must be a correct and valid value determined by w: $\sigma_{crash}(p|w) = v$, which must be possible in all σ's restricted under weak non-equivocation according to the non-faulty process guarantee. As such p cannot be non-faulty.
Case 2: (p is faulty): as p is faulty, it can act non-faulty which is identical to Case 1, which we have shown cannot be the case. Alternatively, p can act faulty, which in the crash-failure model means crashed. If p is crashed then $v = \perp$. But according to weak non-equivocation, is always allowed to report $\perp$, meaning any such σ must be possible under the restriction of weak non-equivocation. As such p cannot be faulty.

Leading to the contradiction that p is neither faulty nor non-faulty. □

So we can conclude that the number of information exchange rounds of Algorithm 2 is optimal. This should not be mistaken as the algorithm being efficient or optimal in the number of messages; we conjecture that a message-efficient (polynomial) algorithm such as the one presented by Dolev et al. [16] could be modified to allow for optimal fault tolerance in a weak non-equivocation fault model, but leave this as future work.

C PROOFS OF CORRECTNESS FOR THE GENERAL WEAK NON-EQUIVOCATION ALGORITHM

The proof relies on the following technical Lemma.

Lemma C.1. *If a process p has a consistent q-majority for the value v, then p's $\widehat{\mathsf{vec}}_p$ must contain at least t v-values.*

$$\forall p, q \in P \colon \forall v \in V \colon (\exists Q \subseteq P \colon \mathsf{maj}^{q,v}_{p,Q}(t, \sigma_p, P)) \implies |\widehat{\mathsf{vec}}_p[\cdot] = v| \geq t$$

Proof. By Definition 4.3, each process in Q have consistently reported v wrt. all other processes in Q including q, to p:

$$\forall w \in Q^* \colon |w| \leq t \colon \sigma_p(p|w|q) = v$$

So $\widehat{\sigma_p}$, the map used to create $\widehat{\text{vec}}_p$, must have the property that each process in $Q \setminus \{q\}$ have consistently reported v about itself and all other processes in $Q \setminus \{q\}$, to p:

$$\forall w' \in (Q \setminus \{q\})^* : |w'| \leq t-1 : \widehat{\sigma_p}(p|w') = v$$

So, for each process in $Q \setminus \{q\}$, p must be able to find a consistent majority for v in the recursive call that makes up $\widehat{\text{vec}}$:

$$\forall r \in Q \setminus \{q\} : \text{maj}^{\text{r,v}}_{\text{p},(\text{Q}\setminus\{\text{q}\})}(t-1, \widehat{\sigma_p}, P \setminus \{q\})$$

And since $|Q| \geq t+1$ by Definition 4.3, then $(Q \setminus \{q\}) \geq t$, giving us the result that $\widehat{\text{vec}}_p$ must contain at least t v-values. □

We can now proceed to prove Algorithm 2 correct, by proving that it satisfies validity, agreement and termination:

Lemma C.2. *Algorithm 2 satisfies validity when $n > 2t$.*

Proof. Assume a non-faulty source p; we must show that for any non-faulty process, Algorithm 2 returns the private value v_p. Because p is non-faulty, all non-faulty processes have a consistent p-majority for v_p (Lemma 4.4), and cannot have a consistent p-majority for another value (Lemma 4.6). Thus Algorithm 2 will return v_p in line 9. □

Lemma C.3. *Algorithm 2 satisfies agreement when $n > 2t$.*

Proof. We must prove that every non-faulty process will get the same result by running Algorithm 2 for some source-process q. By cases on the faultiness of q:

Case 1: (q is non-faulty) Agreement follows from validity if q is non-faulty.

Case 2: (q is faulty) By induction on t:

Base step: $t = 1$. By cases on the q's first messages:

Case 2.i: First consider the case $\forall p \in P \setminus \{q\} : \sigma(p|q) = \bot$, i.e. where the source process does not send a single message in the first round. By Lemma 4.7, none of the non-faulty processes will have a consistent majority. Thus all processes will calculate $\widehat{\text{vec}}$, excluding the source process and make a recursive call with $t = 0$. By Definition 2.1 and since $t = 0$ in the recursive call, all other processes must have sent $\langle\bot\rangle$ in round 2. Thus all values in $\widehat{\text{vec}}$ must be $\langle\bot\rangle$, making Algorithm 2 return $\langle\bot\rangle$ for all non-faulty processes on line 23.

Case 2.ii: Next consider the case $\exists p \in P \setminus \{q\} : \sigma(p|q) \neq \bot$ i.e. where the source process sends a value v ($\neq \bot$) to one or more processes in the first round. For any process, p, receiving v in the first round, Algorithm 2 will return v on line 9 as $\{p, q\}$ is a consistent q-majority. All other processes must have received $\bot$ in the first round by weak non-equivocation (Definition 4.1), and thus, by Lemma 4.7, calculate $\widehat{\text{vec}}$, excluding q and with $t = 0$. By the non-faulty process guarantee (Definition 2.1), and Lemma 4.4, $\widehat{\text{vec}}$ consist only of $\langle\bot\rangle$ and a v value for each process that received v in the first round, which is at least 1. As $\widehat{\text{vec}}$ contains only $\langle\bot\rangle$ and v Algorithm 2 will, for all non-faulty processes that did not receive v in the first round, return v in line 21 since $1 \geq t = 1$.

Induction step: Induction hypothesis: Algorithm 2 satisfies agreement for $t-1$ for any $n > 2(t-1)$. We must prove that, given the induction hypothesis, Algorithm 2 satisfies agreement for t. Recall that by assumption $n > 2t$. Consider any two non-faulty processes, q and r, and a faulty source process p. The proof is on cases of existence of consistent p-majorities for q and r.

Case 2.iii: Consider the case $\exists v, v' \in V : \exists (Q, R) \subseteq P : \text{maj}^{\text{p,v}}_{\text{q,Q}}(t, \sigma_q, P) \wedge \text{maj}^{\text{p,v}'}_{\text{r,R}}(t, \sigma_r, P)$ i.e. where q has a consistent p-majority for some value v, and r has a consistent p-majority for some value v'. By Lemma 4.8, we know that $v = v'$.

Case 2.iv: Consider the case $\forall v, v' \in V : \forall Q \subseteq P : \neg\text{maj}^{\text{p,v}}_{\text{q,Q}}(t, \sigma_q, P) \wedge \neg\text{maj}^{\text{p,v}'}_{\text{r,Q}}(t, \sigma_r, P)$ i.e. where neither q nor r have a consistent p-majority. Both q and r will calculate $\widehat{\text{vec}}$ by calling Algorithm 2 with $n-1$ and $t-1$ and each process other than p as the source process. By assumption $n > 2t$, so $n - 1 > 2(t-1)$. As such the induction hypothesis applies, and so q's $\widehat{\text{vec}}$ and r's $\widehat{\text{vec}}$ must be equal.

Case 2.v: Consider the case $(\exists v \in V : \exists Q \subseteq P : \text{maj}^{\text{p,v}}_{\text{q,Q}}(t, \sigma_q, P)) \wedge (\forall v' \in V : \forall R \subseteq P : \neg\text{maj}^{\text{p,w}}_{\text{r,R}}(t, \sigma_R, P))$ i.e. where q has a consistent p-majority for some value v, and r does not have a consistent p-majority for any value. Let q calculate $\widehat{\text{vec}}$ using σ_q, *despite* having a consistent p-majority, and denote the result $\widehat{\text{vec}}_q$. By the induction hypothesis r will calculate the same $\widehat{\text{vec}}$ as q: $\widehat{\text{vec}}_r = \widehat{\text{vec}}_q$. By Lemma C.1 we know that there is at least t v-values in $\widehat{\text{vec}}_q$ and therefore also in $\widehat{\text{vec}}_r$. As there is t v-values in $\widehat{\text{vec}}_t$ Algorithm 2 returns v on line 21 for r, while returning v for q in line 9, as q has a consistent p-majority for v.

Note that in the case where $n = 2t + 1$, $\widehat{\text{vec}}_r$ will contain exactly $2t$ values, leaving room in $\widehat{\text{vec}}_r$ for a value different from v with t instances. However, such a value can only be $\langle\bot\rangle$ by the following argument. If there are two different values with t instances in $\widehat{\text{vec}}_r$, then q's consistent p-majority for value v must be of size exactly $t+1$ by consequence of Lemma C.1. As r does not have a consistent p-majority, q's consistent majority must contain at least 1 faulty process by Lemma 4.5. This implies that there is at least one non-faulty process, s, that is not part of q's consistent majority. Note that s and r may be the same process. According to weak non-equivocation (Definition 4.1) s must have received either v or $\bot$ in the first round. And by validity (Lemma C.2), $\widehat{\text{vec}}_r[s]$ must then be either v or $\langle\bot\rangle$, ensuring that if $\widehat{\text{vec}}_r$ t instances of a value other than v, that value must be $\langle\bot\rangle$.

□

Lemma C.4. *Algorithm 2 satisfies termination for any terminating protocol.*

Proof. Termination follows from the following facts: (1) when $t = 0$ there is always a consistent majority (Lemma 4.4); (2) all loops are finite, including the loop over all $w \in \widehat{P^*}$ s.t. $p|w|q$ is a key in σ (line 13), as there is a finite amount of keys in σ; and (3) n and t are finite, and so the number of recursion-calls to PSL($\widehat{\sigma_p}, n-1, t-1, p, r, \widehat{P}$) (line 18) must be finite. □

Proof of Theorem 5.1. Immediate from Lemmas C.2 (validity), C.3 (agreement), and C.4 (termination). □