Blue fish, red fish, live fish, dead fish

Victor Shoup
Offchain Labs
victor@shoup.net

February 5, 2025

Abstract

We show that the DAG-based consensus protocol Tusk [DKSS22] does not achieve liveness, at least under certain reasonable assumptions on the implementation that are consistent with its specification. In addition, we give a simple 2-round variation of Tusk with lower latency and strong liveness properties, but with suboptimal resilience. We also show that another 2-round protocol, GradedDAG [DZX+24], which has optimal resilience, also has liveness problems analogous to Tusk.

1 DAG-rider and Tusk

DAG-rider [KKNS21] is a remarkably simple, modular, and elegant asynchronous atomic broadcast protocol [CKPS01]. Tusk [DKSS22] is a "a practical extension of DAG-Rider" that "modifies DAG-Rider into an implementable system and improves its latency in the common case".

We assume that the reader is familiar with these protocols, but we quickly review the salient features of both. There are n parties $P_1, \ldots, P_n$ connected by secure but *asynchronous* point-to-point channels, and an adversary may corrupt at most $f < n/3$ parties.

In both of these protocols, parties construct a common DAG (directed acyclic graph) in a round-by-round fashion. While all parties have a view of the same DAG, at any instant in time, they may have somewhat different views. In each round, each party *reliably broadcasts* (in the sense of [Bra87]) a node v together with a set $\mathsf{Successor}(v)$ of nodes from the previous round to which v points. In the very first round, $\mathsf{Successor}(v)$ is empty, while in subsequent rounds we must have $|\mathsf{Successor}(v)| \geq n - f$. (DAG-rider also allows so-called "weak edges" that point to nodes in more distant rounds, but we shall ignore these here.) Whenever a party P has received v, and each $w \in \mathsf{Successor}(v)$ has already been added to its local DAG, P will add v to its local DAG. Each party moves onto the next round as soon as it adds $n - f$ nodes for the current round to its local DAG.

The sequence of rounds is organized into a sequence of *waves*, where each wave consists of a small, fixed number of rounds. When each party finishes the last round of a wave, it will obtain the value of a common coin that retroactively elects a node in the first round of the wave as the *leader* for that wave. The party then applies a *commit rule* to decide if that leader should be *explicitly committed*. If the party does explicitly commit the leader

for the current wave, it may also *implicitly commit* the leaders from some previous waves (using a different rule).

Tusk implements an *overlapping wave* optimization, in which the last round of one wave is the first round of the next. DAG-rider does not implement such an optimization.

Both protocols satisfy a *safety property*, which ensures that all honest parties commit (either explicitly or implicitly) the same sequence of leaders.

DAG-rider satisfies a very strong *liveness property*: in each wave, all honest parties explicitly commit the leader of that wave with constant probability. We will show that Tusk does not satisfy such a strong liveness property.

2 Assumptions and results

We assume n is a large number of the form $n = 3f + 1$. We assume the adversary corrupts exactly f parties and completely controls network scheduling. That is, the adversary may decide exactly which messages are delivered and when. The adversary is still subject to the usual "eventual delivery" requirement, which means that it cannot refuse to deliver a given message sent from one honest party to another indefinitely.[1]

Tusk uses 3 waves per round. Recall that a leader for a wave is a node in the first round of the wave that is elected in the last round in the wave. The commit rule in Tusk says that this node is explicitly committed if it is pointed to by at least $f + 1$ nodes in the second round.

This commit rule is applied by each party on its local view of the DAG: for a party to explicitly commit a leader v, it must be the case that its local DAG contains v as well as $f + 1$ nodes in the next round that point to v. Crucially, we assume that this commit rule is applied by each party *only at the time it finishes the last round of the wave*. Although [DKSS22] is a bit vague on this point, this assumption is consistent with the plain meaning of the text, as well as with the logic of the DAG-rider protocol — which is described in the paper [KKNS21] with detailed pseudocode, and on which the Tusk protocol is based. The authors of [DKSS22] also confirm (personal communication) that this is the correct interpretation of their paper. (We shall consider the implications of an alternative commit strategy in Section 4.1).

Tusk uses a common coin subprotocol to elect leaders, which in [DKSS22] is suggested can be implemented using threshold BLS signatures [BLS01]. Although [DKSS22] does not specify the reconstruction threshold for the coin, it seems safe to assume that this is $f + 1$, as in [KKNS21].[2] The authors of [DKSS22] also confirm (personal communication) that

[1] Note that in a long-running system with no bound on the number of messages sent, formally defining the notion of "eventual delivery" is actually somewhat tricky. However, the precise definition of this notion will have no impact on our arguments here.

[2] The formal definition of a common coin (or "global perfect coin") in [KKNS21] is logically inconsistent. The termination condition says that "if at least $f + 1$ parties call *choose_leader* ...". This should say "if at least $f + 1$ *honest* parties call *choose_leader*". The unpredictability condition says that "if fewer than $f + 1$ parties call *choose_leader* ...". This should say "if no honest parties call *choose_leader*", unless the intent was to define a common coin with a reconstruction threshold of $2f + 1$. But if that was the intent, this would be inconsistent with the termination condition as well as the subsequent recommended implementation of the common coin via a $(f + 1)$-out-of-n threshold signature. We also not that the definition of a common coin in the follow-up work [SGSK22] is slightly different from that in [KKNS21] but is also logically inconsistent.

this is the correct interpretation of their paper. (We shall consider the implications of a high-threshold common coin in Section 4.2). As we are assuming f parties are corrupt, it follows that the adversary will learn the value of a common coin as soon as one of the honest parties reveals its share of that coin. Note that this is the only way in which we exploit the fact that f parties are corrupt — our attack works even if the corrupt parties otherwise follow the protocol. We shall assume that P_1 is honest, but the choice of corrupt parties among $P_2, \ldots, P_n$ is arbitrary.

Crucially, we assume that a party reveals its share of the common coin for a wave *as soon as it finishes round 2 of the wave*. With this assumption, the adversary learns the leader for a wave as soon as the first honest party finishes round 2 of that wave. Again, although [DKSS22] is a bit vague on this point, this assumption is consistent with the plain meaning of the text in [DKSS22]. Indeed, the paper states that shares of the random coins are piggybacked with nodes being broadcast (just as in [KKNS21]). Moreover, to achieve the stated latency results, this share must be broadcast in round 3 of a wave, which supports our assumption. The authors of [DKSS22] also confirm (personal communication) that this is the correct interpretation of their paper.

As mentioned above, other than the fact that the adversary knows f shares of the common coin, all parties behave honestly. However, the adversary controls the network, and can hence decide exactly when a node that has been reliably broadcast is delivered to any individual party. That said, we will restrict ourselves to adversaries that adhere to the following *causal-delivery restriction*:

> the adversary will only deliver a node v to a party if there are $2f + 1$ parties participating in the reliable broadcast of v, and each of these parties has already added each $w \in \mathsf{Successor}(v)$ to its local DAG.

This restriction is useful in that it applies to a wide variety of DAG-building subprotocols, including those that may use an implementation of reliable broadcast in which a party only supports the reliable broadcast of v when it already has added each $w \in \mathsf{Successor}(v)$ to its local DAG. This restriction only makes our attack more powerful. Although [DKSS22] is not entirely clear in this point, the authors of [DKSS22] confirm (personal communication) that the DAG-building subprotocol Narwhal used by Tusk does indeed impose this restriction.

An implementation of Tusk could perhaps choose to implement the DAG-building subprotocol in such a way that each party only finishes a round when its own node for that round has been added to its local DAG. In fact, the protocol could even require that a node broadcast by a party in one round *always* points to a node broadcast that same party in the previous round. The paper [DKSS22] does not suggest that any such *self-delivery restriction* is required; moreover, the authors of [DKSS22] confirm (personal communication) that Narwhal does not impose any such restriction, and that there are practical reasons for not doing so. (We shall consider the implications of a *self-delivery restriction* in Section 4.3).

Under these assumptions, we show the following:

> *There is an efficient adversary that makes all parties proceed through an arbitrary number of waves such that for all sufficiently large n, only P_1 ever commits any leaders — parties $P_2, \ldots, P_n$ never commit a leader in* any *wave.*

This attack relies on the adversary's ability to *temporarily* delay the delivery of certain nodes to some parties. However, all nodes broadcast in one wave will be delivered to all parties by the end of the next wave.

This situation stands in stark contrast to the liveness property enjoyed by the DAG-rider protocol, which guarantees that in each wave, with probability at least 2/3, all honest parties will explicitly commit the leader of that wave.

Besides providing this attack, we consider possible mitigations in Section 4. None of these mitigations are entirely satisfactory, as they do not restore strong liveness (similar to DAG-rider), but they may be acceptable in practice.

In Section 5 we try to give some insight into why these weaknesses appear in Tusk, and what broader lessons there might be.

We also present a simple variation of Tusk with less latency (two rounds per wave) and strong liveness properties (similar to DAG-rider, but with a lower commit probability). We call this variation *2-Tusk*. (Protocol 2-Tusk does not use the overlapping wave optimization, and as such, still commits a leader every other round, just like Tusk.) The trade-off is that 2-Tusk has suboptimal resilience ($f < n/(3+\sqrt{3}) \approx n/4.732$). While the commit probability in each wave is a fairly small constant (≈ 0.2), in the common case (with random message delivery), the commit probability is very close to 1. We briefly present and analyze 2-Tusk in Section 6.

We also briefly compare 2-Tusk to GradedDAG [DZX+24], which is another 2-round-per-wave asynchronous DAG protocol (but with optimal resilience), and observe that it has liveness problems analogous to Tusk.

We note that while there is an "academic" implementation of Tusk (see `https://github.com/facebookresearch/narwhal/`), we know of no "production" implementation of Tusk to which our attacks would apply.

3 An attack

So now to our attack on Tusk. We initially attack a simplified version of Tusk with *non-overlapping waves*. We then discuss how the argument can be easily adapted for overlapping waves.

We focus on a single wave. We describe an adversary that makes all parties finish the wave, but while P_1 may explicitly commit the leader in this wave, no other party explicitly commits to the leader of this wave (assuming n is sufficiently large). For $i = 1, \ldots, n$ and $r = 1, 2, 3$, let $v_{i,r}$ be the round-r node in the wave reliably broadcast by party P_i.

Stage 1. Each party P_i begins round 1 of the wave by broadcasting the round-1 node $v_{i,1}$. As a precondition, we shall assume that every successor of every such round-1 node has already been added to every party's local DAG. With this precondition, we maintain the following invariant throughout the attack: whenever our adversary delivers a node v to a party P, the *causal-delivery restriction* will be satisfied, and P can immediately add v to its local DAG.

We define sets of round-1 nodes $S_1, \ldots, S_{2f+1}$, each of size $2f + 1$, as follows. The first f sets $S_1, \ldots, S_f$ are defined as arbitrary sets of size $2f + 1$ round-1 nodes. The remaining

$f+1$ sets $S_{f+1}, \ldots, S_{2f+1}$ are defined in such a way that for $j = 1, \ldots, n$, the node $v_{j,1}$ appears in at most f of these sets. For n sufficiently large, this can always be done, for example, using the following "banded matrix" type of construction. For $i = 1, \ldots, f+1$, we set

$$S'_{f+i} = \{v_{1,1}, \ldots, v_{n,1}\} \setminus \{v_{k,1} : k \equiv i \bmod (f+1), 1 \leq k \leq n\}.$$

By construction, for $j = 1, \ldots, n$, the node $v_{j,1}$ appears in at most f of these sets. Except for very small values of n, each of these sets is also of size at least $2f+1$. We can then derive S_{f+i} from S'_{f+i} for $i = 1, \ldots, f+1$ by deleting elements to obtain sets of size exactly $2f+1$.[3]

The adversary now arranges delivers to parties $P_1, \ldots, P_{2f+1}$ the subsets $S_1, \ldots, S_{2f+1}$ of round-1 nodes, respectively, without delivering (for now) any round-1 nodes to the remaining f parties $P_{2f+2}, \ldots, P_n$. Parties $P_1, \ldots, P_{2f+1}$ finish round-1.

Stage 2. Now parties $P_1, \ldots, P_{2f+1}$ broadcast their corresponding round-2 nodes $v_{1,2}, \ldots, v_{2f+1,2}$. Note that for $i = 1, \ldots, 2f+1$, we have $\mathsf{Successor}(v_{i,2}) = S_i$. After this, the adversary delivers

- to $P_1, \ldots, P_{2f+1}$, all outstanding round-1 nodes, and
- to P_1, the round-2 nodes $v_{1,2}, \ldots, v_{2f+1,2}$.

Party P_1 finishes round 2.

Stage 3. Now party P_1 broadcasts its round-3 node $v_{1,3}$. Note that $\mathsf{Successor}(v_{1,3}) = \{v_{1,2}, \ldots, v_{2f+1,2}\}$. In addition, the index ℓ of leader for the wave is revealed to the adversary at this time. So now, the adversary delivers to the remaining f parties $P_{2f+2}, \ldots, P_n$ sets of round-1 nodes $S_{2f+2}, \ldots, S_n$, respectively, where each of these sets is an arbitrary subset of $2f+1$ round-1 nodes that *excludes the leader* $v_{\ell,1}$. The f parties $P_{2f+2}, \ldots, P_n$ finish round 1.

Stage 4. Parties $P_{2f+2}, \ldots, P_n$ broadcast their corresponding round-2 nodes $v_{2f+1,2}, \ldots, v_{n,2}$. Note that for $i = 2f+2, \ldots, n$, we have $\mathsf{Successor}(v_{i,2}) = S_i$. The adversary then delivers

- to the parties $P_{2f+2}, \ldots, P_n$, all outstanding round-1 nodes,
- to P_1, the remaining f round-2 nodes $v_{2f+1,2}, \ldots, v_{n,2}$, and
- to the parties $P_2, \ldots, P_n$, the $2f+1$ round-2 nodes $v_{f+1,2}, \ldots, v_{n,2}$ — while the other f round-2 nodes $v_{1,2}, \ldots, v_{f,2}$ must eventually be delivered to each of $P_2, \ldots, P_n$, such delivery will be deferred to some point in time after the current wave completes (which will not impede the progress of the current wave).

Parties $P_2, \ldots, P_n$ finish round 2.

[3] One could also just choose the sets $S_{f+1}, \ldots, S_{2f+1}$ as random subsets of size $2f+1$. For large n, these sets will satisfy the required property with overwhelming probability.

Stage 5. Parties $P_2, \ldots, P_n$ broadcast their corresponding round-3 nodes $v_{2,3}, \ldots, v_{n,3}$. Note that for $i = 2, \ldots, n$, we have $\mathsf{Successor}(v_{i,3}) = \{v_{f+1,2}, \ldots, v_{n,2}\}$. At this point in time, P_1 has added all round-2 nodes to its DAG, while $P_2, \ldots, P_n$ have added only $v_{f+1,2}, \ldots, v_{n,2}$. The adversary then delivers $v_{2,3}, \ldots, v_{n,3}$ to all parties. All parties immediately add these to their local DAG and so finish round 3.

Analysis. We claim that none of the parties $P_2, \ldots, P_n$ explicitly commit the leader $v_{\ell,1}$. To see this, note that when they finish round 3, each of these parties received the $2f + 1$ round-2 nodes $v_{f+1,2}, \ldots, v_{f,n}$ with successor sets $S_{f+1}, \ldots, S_n$. By construction, the leader $v_{\ell,1}$ appears in at most f of these sets (in at most f of the sets $S_{f+1}, \ldots, S_{2f+1}$ and in none of the others). Therefore, none of these parties commit the leader.

Note that in our attack, when one wave finishes, the precondition in Stage 1 for the next wave will be satisfied. Therefore, the same thing will happen in each and every wave: P_1 may explicitly commit, but $P_2, \ldots, P_n$ will not. As such, $P_2, \ldots, P_n$ will never implicitly commit either.

Adjusting the argument for overlapping waves. In the actual version of Tusk, round 3 of a given wave is the same as round 1 of the next wave. Let us decorate each node with a superscript indicating its wave number, writing $v_{i,r}^{(w)}$ for the node $v_{i,r}$ in wave w. With overlapping waves, we have $v_{i,1}^{(w+1)} = v_{i,3}^{(w)}$.

In Stage 5 of the above attack in wave w, the adversary delivers to all parties the $n - 1$ round-3 nodes $v_{2,3}^{(w)}, \ldots, v_{n,3}^{(w)}$. As we saw, when these nodes are delivered to a party, that party has already added all successors of these nodes to its local DAG, namely $v_{f+1,2}^{(w)}, \ldots, v_{n,2}^{(w)}$. As such, there are very few constraints on exactly when these round-3 nodes must actually be delivered. In our attack, in wave $w + 1$, these the round-3 nodes $v_{2,3}^{(w)}, \ldots, v_{n,3}^{(w)}$ need to be processed as round-1 nodes $v_{2,1}^{(w+1)}, \ldots, v_{n,1}^{(w+1)}$, strategically delivering some of them to some parties, while temporarily withholding them from others. However, if we want the adversary to adhere to our *causal-delivery restriction*, then we need to be careful not to deliver $v_{1,1}^{(w+1)} = v_{1,3}^{(w)}$ prematurely to *any* party, as $\mathsf{Successor}(v_{1,3}^{(w)})$ includes *all* round-2 nodes in wave w, and so we would have to deliver all these nodes to many parties before they complete wave w, thereby impeding our attack. The solution is to simply delay the delivery of the node $v_{1,1}^{(w+1)}$ to *any* party — including P_1 — at least until *all* parties have finished the first round of wave $w+1$. To this end, we simply arrange that the sets $S_1^{(w+1)}, \ldots, S_n^{(w+1)}$ do not contain $v_{1,1}^{(w+1)}$. This extra constraint is easily imposed while maintaining all of the other requirements. In addition, in the attack, wherever we deliver "all outstanding round-1 nodes" in wave $w + 1$, we omit $v_{1,1}^{(w+1)}$. The node $v_{1,1}^{(w+1)} = v_{1,3}^{(w)}$, together with all nodes from wave w, may be safely delivered to all parties as soon as all parties finish wave w.

4 Mitigations

4.1 Aggressive commit

Tusk is specified so that each party "lazily" applies the commit rule for a wave only at the point in time when it finishes that wave. One possible mitigation is to instead have each party "aggressively" apply the commit rule, so that it will explicitly commit a leader (using the same commit rule as before) in a wave w even after it has moved on to a later wave $w' > w$.

Using such an aggressive commit rule does soften the impact of our attack, since if P_1 explicitly commits the leader in wave w, then by the eventual delivery assumption, every party will eventually commit that leader. However, an adversary is still able to drive all parties forward through an unbounded number of waves with only one party explicitly committing anything at all. Note that to do this, the adversary must delay the delivery of certain nodes for longer periods of time: to prevent a party from committing in waves $w, \ldots, w'$, the adversary may have to delay the delivery of some nodes in each of these waves until wave $w' + 1$.

If nothing else, this unboundedness means that additional care must be taken to bound the storage requirement of the protocol — indeed, Tusk was touted as a protocol that supports garbage collection and bounded storage. So in addition to the aggressive commit rule, one also needs to implement a "back-pressure" rule: if a party goes too many waves without any commits, it will stop participating in any further waves (that is, it will stop sending or receiving messages associated with those further waves) until something is committed.

4.2 High-threshold common coin

Another possible mitigation is to use a high-threshold common coin with a reconstruction threshold of $2f + 1$ rather than $f + 1$. This is certainly possible (although setting up a high-threshold coin is a bit more expensive than setting up a low-threshold coin). Using such a coin seems to defeat our particular attack, and we are not aware of any other attack; however, it is an open question as to whether a stronger liveness property (like that for DAG-rider) can be proven for this Tusk variant.

4.3 Imposing a self-delivery restriction

Our specific attack on Tusk with overlapping waves required that we not deliver the "hot potato" node $v_{1,1}$ in a wave to any party — including P_1 — until all parties have finished round-1 of the wave. As mentioned earlier, an implementation could perhaps choose to implement the DAG-building subprotocol with a *self-delivery restriction*, in which each party only finishes a round when its own node for that round has been added to its local DAG; moreover, the protocol could even require that a node broadcast by a party in one round *always* points to a node broadcast that same party in the previous round. If this is case, our attack can very easily be modified to handle this, if we also drop the *causal-delivery restriction*. If we add this *self-delivery restriction* but keep the *causal-delivery restriction*, we can make a weaker version of our attack work in which parties P_1 and P_2 commit but no other party commits. The idea is that we would have parties P_1 and P_2 alternate roles

in each wave, with one of them playing the role of P_1 in our original attack, while the other (holding the "hot potato") is cut off until all other parties have finished round-1 of the wave.

5 What went wrong?

The DAG-rider protocol uses 4-round waves, where each wave essentially implements a "Common Core", or "Gather", protocol with a "binding core" (see [AJM+21]). The 3-round waves in Tusk do not implement such a "Common Core" protocol. The crux of the liveness argument for Tusk is Lemma 3 in [DKSS22]. The proof of that lemma says that the *first* honest party that finishes the second round of a wave will commit the leader of that wave with probability at least 1/3. But that is all it says — in particular, it does not say that any other party will commit anything at all. So in a nutshell, Tusk cuts too many corners to maintain the "binding core" property that was essential to establish the strong liveness property enjoyed by DAG-rider.

The reader may criticize our result by observing that our adversary is quite powerful, as it really needs complete control over scheduling the delivery of protocol messages. This is a fair criticism, and it shows that our attack might be rather hard to pull off in practice. Nevertheless, in terms of the accepted standards in the research community for analyzing the security properties of these kinds of protocols, it is a real attack.

The reader may also criticize our result by claiming that we have just set up a "straw man" that was easy to knock down. We admit that this criticism is not entirely without merit. Indeed, it might be very natural for an implementer to implement the aggressive commit rule — it is fairly clear that this rule maintains safety and only improves performance, and it may well fit more naturally into the system architecture than the lazy commit rule. However, this was not the rule specified in [DKSS22]. Moreover, even if this rule is implemented, an implementer may be somewhat less likely to implement an explicit back-pressure rule — if the implementer assumes that Tusk enjoys a strong liveness property similar to DAG-rider, it would be safe to assume that it was essentially impossible for a party to see more than (say) 100 waves to go by without committing anything. Nevertheless, a paranoid, defense-in-depth implementation might naturally include such a back-pressure rule. In contrast, if we use DAG-rider's 4-round wave structure and its stronger commit rule (ignoring its "weak edges" logic), we do not need any such aggressive commit or back-pressure rules to maintain liveness and bounded storage.

In any case, for protocols such as this, it is important for an implementer to know which aspects of the protocol are "implementation details" that may be modified or omitted without concern for losing safety or liveness (or bounded storage), and which are not. If nothing else, we hope that our observations in this brief note make it clear that binding properties are essential in such protocols, and that care should be taken to ensure that these binding properties are satisfied, or if not, to implement appropriate mitigations to account for that.

We also hope to raise awareness to the fact that the definitions of "liveness" (and "eventual delivery") used throughout the literature on asynchronous atomic broadcast are a bit imprecise and inconsistent. For example, we think there should at least be an awareness of the distinction between

- a strong notion of liveness that prevents (with overwhelming probability) a party from running too far ahead of the last commit point (such as DAG-rider, as well as iterative multi-valued agreement protocols as in [CKPS01, MXC+16]), and
- a weak notion of liveness that does not (such as Tusk with an aggressive commit rule).

6 *2-Tusk:* a 2-round version of Tusk

In this section, we briefly describe and analyze a version of Tusk with just 2 rounds per wave, which we call *2-Tusk*. The rules for explicit and implicit commit are identical to Tusk (without the aggressive commit or back-pressure rules discussed above). The only other differences are as follows:

- Unlike Tusk, protocol 2-Tusk does not use the overlapping wave optimization.
- Instead of using a common coin with reconstruction threshold $f+1$ as in Tusk, protocol 2-Tusk uses a "high threshold" common coin with reconstruction threshold $n-f$.
- Instead of assuming $f < n/3$ as in Tusk, protocol 2-Tusk assumes $f < n/(3+\sqrt{3})$.

Note that $3+\sqrt{3} \approx 4.732$. Because a wave only consists of 2 rounds, each party reveals its share of the common coin after it finishes the first round of the wave. However, we are using a high-threshold common coin, and so the leader of the wave will only be revealed after $n-2f$ honest parties have finished that first round.

It is clear that 2-Tusk provides safety, following the same argument as for Tusk. Note that if we used the overlapping wave optimization in 2-Tusk, we would lose safety: to achieve safety using Tusk's commit rule, there must be at least one round between successive leaders.

As for liveness, we can show that in every wave, with probability at least $(f+1)/n$ (so $\approx 1/5$ when $f \approx n/5$), all honest parties will explicitly commit the leader of that wave. This is exactly the same liveness property enjoyed by DAG-rider, but with a smaller success probability. However, we will argue that in the common case, under a random message delivery assumption (the same as made in [DKSS22]), this success probability is much higher.

The worst-case liveness property can be proved as follows. We will use the same notation for naming vertices and their successor sets as above. Consider the point in time when $n-2f$ honest parties have finished round-1 and broadcast their round-2 nodes. Without loss of generality, suppose these are parties $P_1, \ldots, P_{n-2f}$, and for $i = 1, \ldots, n-2f$, let $S_i = \mathsf{Successor}(v_{i,2})$, each of which has size (at least) $n-f$. The common coin may be revealed at this time. Let us define the 0/1 matrix M with $n-2f$ rows and n columns, where the entry in the ith row and jth column is 1 iff $v_{1,j} \in S_i$. So by assumption the weight (number of 1 entries) of M is at least $(n-f)(n-2f)$.

Let us call a column of M *heavy* if its weight is at least $2f+1$. We claim that there must be at least $f+1$ heavy columns in M. This is a simple counting argument. Suppose the claim is false. Then there must be $(n-f)$ columns of weight at most $2f$, while the remaining f columns may (trivially) have weight at most $(n-2f)$. This implies that the weight of M is at most $2f(n-f)+(n-2f)f$. So we have

$$(n-f)(n-2f) \leq 2f(n-f)+(n-2f)f,$$

which simplifies to

$$n^2 - 6nf + 6f^2 \leq 0,$$

which implies

$$f \geq n/(3+\sqrt{3}),$$

a contradiction.

By the claim, the probability that the leader for the wave belongs to a party whose index labels a heavy column in M is at least $(f+1)/n$. When that happens, when any party completes the wave, it must collect $n-f$ round-2 nodes. These must include nodes from all but f of the parties $P_1, \ldots, P_{n-2f}$. This means that at least $(2f+1)-f = f+1$ of these round-2 nodes will point to the leader, which means that the party will commit the leader.

That completes the worst-case analysis. For the common case, one can adapt the proof of Lemma 5 in [DKSS22]. Here, we are assuming all of the parties (or least all but a very small fraction thereof) are honest, and that in each round, a party P_i receives nodes from a random subset S_i of $n-f$ parties. We assume that the sets $S_1, \ldots, S_n$ are mutually independent, and independent of the identity of any leader. Now consider a fixed party P. It collects $n-f$ round-2 nodes, each of which points to a random subset of round-1 nodes. The probability that P does not commit is seen to be the probability that we run $n-f$ independent Bernoulli trials, each with failure probability at most f/n, and record at least $n-2f$ failures. The expected number of failures among these trials is $(n-f)(f/n)$ which is much smaller than $n-2f$, and by Chernoff bounds, for large n, the probability that any individual party fails to commit is negligible.

With the requirement that $f/n < 1/(3+\sqrt{3}) \approx 0.211$, a minimally sized system has $f = 1$ and $n = 5$. For these parameters, the worst-case commit probability is $(f+1)/n = 2/5$. In the random message delivery model, we can calculate an upper bound on not committing as the probability that we run 4 Bernoulli trials, each with failure probability $1/5$, and record at least 3 failures. This probability is ≈ 0.027. So even in this small system, each party commits in each wave with over 97% probability. In addition, in the case where a single party is permanently crashed, each remaining party commits with probability 80% (the probability of not choosing the crashed leader).

As a purely mathematical curiosity, we note that the bound $f/n < 1/(3+\sqrt{3}) \approx 0.211$ is not optimal, in the sense that one can also prove that for every constant $\epsilon > 0$, there exists a constant $\delta > 0$, such that for all sufficiently large n, protocol 2-Tusk has commit probability of at least δ provided $f/n - \epsilon < (5-\sqrt{17})/4 \approx 0.219$.

6.1 Comparison to GradedDAG

GradedDAG [DZX+24]. other asynchronous DAG-based protocol with a similar 2-round-per-wave structure. This protocol has optimal resilience, requiring only $f < n/3$. It uses a variation of reliable broadcast with special features, called *graded reliable broadcast*, in the first round in each wave, and uses a specialized commit rule that exploits these features. We note that the analysis in [DZX+24] only proves a weak form of liveness, similar to that of Tusk. Namely, Lemma 6 in [DZX+24] only says that the *first* honest party that finishes

the first round of a wave w will explicitly commit the leader of that wave with probability at least $2/3$.[4]

While the safety argument implies that if another party later explicitly commits in a wave $w' > w$, then it will implicitly commit in wave w, the liveness argument does not say that any other party will ever explicitly commit anything at all. Indeed, it seems that the our liveness attack against Tusk can easily be adapted to GradedDAG, so that an adversary can drive all parties through an arbitrary number of waves with only one party ever committing anything at all. Moreover, the description of GradedDAG in [DZX+24] is quite clear that it is based on a lazy commit rule. The idea of the attack is as follows. The adversary delivers $2f + 1$ round-1 nodes to P_1 with grade 2, without delivering any other round-1 nodes with grade 2 to any other parties.[5] When this happens, the leader for the wave is revealed. After this, the adversary can arrange that every other party completes round-1 by delivering to them $2f + 1$ round-1 nodes with grade 2 that avoid the leader. From here, it is straightforward to make all honest parties complete the wave so that none of $P_2, \ldots, P_n$ commit the leader. We note that just as for Tusk, we can mitigate against this attack by using aggressive commit and back-pressure rules. Using a high-threshold coin may also help, but we do not see how to prove that. We note that the authors of [DZX+24] concur with the above observations (personal communication).

6.2 Comparision to 2PAC

The paper [Ram24] contains a number of variations of a consensus protocol called 2PAC, which has a 2-round structure. While not a DAG-based protocol itself, the paper describes a generic transformation that would convert this protocol into a DAG-based protocol with two rounds per wave (and optimal resilience). We have not been able to fully analyze the liveness properties of this protocol.

Acknowledgments

We thank Kartik Nayak and Nibesh Shrestha for comments on an early draft on this note, and for suggesting the aggressive commit rule as a possible mitigation. We thank Sasha Spiegelman and Alberto Sonnino for discussions on Narwhal and Tusk and other fish. We thank Xiaohai Dai for discussions on GradedDAG. We thank Matthieu Rambaud for discussions on 2PAC.

[4] We are assuming here a common coin with reconstruction threshold $f + 1$. Note, however, that the definition of a common coin in [DZX+24] is the same, logically inconsistent definition that appears in [KKNS21]. However, our assumption is entirely consistent with the proof of the lemma, which states that the leader may be revealed when the first honest party finishes the first round.

[5] The paper [DZX+24] introduces a reliable broadcast with two grades, 1 and 2. Roughly speaking, grade 1 implies consistency and grade 2 also implies that at least $f + 1$ honest parties have delivered with a grade of 1 or 2.

References

[AJM+21] I. Abraham, P. Jovanovic, M. Maller, S. Meiklejohn, G. Stern, and A. Tomescu. Reaching consensus for asynchronous distributed key generation, 2021. arXiv:2102.09041, http://arxiv.org/abs/2102.09041.

[BLS01] D. Boneh, B. Lynn, and H. Shacham. Short signatures from the Weil pairing. In C. Boyd, editor, *Advances in Cryptology - ASIACRYPT 2001, 7th International Conference on the Theory and Application of Cryptology and Information Security, Gold Coast, Australia, December 9-13, 2001, Proceedings*, volume 2248 of *Lecture Notes in Computer Science*, pages 514–532. Springer, 2001.

[Bra87] G. Bracha. Asynchronous byzantine agreement protocols. *Inf. Comput.*, 75(2):130–143, 1987.

[CKPS01] C. Cachin, K. Kursawe, F. Petzold, and V. Shoup. Secure and efficient asynchronous broadcast protocols. In J. Kilian, editor, *Advances in Cryptology — CRYPTO 2001*, 2001. Also at https://eprint.iacr.org/2001/006.

[DKSS22] G. Danezis, L. Kokoris-Kogias, A. Sonnino, and A. Spiegelman. Narwhal and Tusk: a DAG-based mempool and efficient BFT consensus. In Y. Bromberg, A. Kermarrec, and C. Kozyrakis, editors, *EuroSys '22: Seventeenth European Conference on Computer Systems, Rennes, France, April 5 - 8, 2022*, pages 34–50. ACM, 2022. Also at arXiv:2105.11827, http://arxiv.org/abs/2105.11827.

[DZX+24] X. Dai, Z. Zhang, J. Xiao, J. Yue, X. Xie, and H. Jin. GradedDAG: An asynchronous DAG-based BFT consensus with lower latency. Cryptology ePrint Archive, Paper 2024/142, 2024. https://eprint.iacr.org/2024/142.

[KKNS21] I. Keidar, E. Kokoris-Kogias, O. Naor, and A. Spiegelman. All you need is DAG. In A. Miller, K. Censor-Hillel, and J. H. Korhonen, editors, *PODC '21: ACM Symposium on Principles of Distributed Computing, Virtual Event, Italy, July 26-30, 2021*, pages 165–175. ACM, 2021. Also at arXiv:2102.08325, http://arxiv.org/abs/2102.08325.

[MXC+16] A. Miller, Y. Xia, K. Croman, E. Shi, and D. Song. The honey badger of BFT protocols. In E. R. Weippl, S. Katzenbeisser, C. Kruegel, A. C. Myers, and S. Halevi, editors, *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, Vienna, Austria, October 24-28, 2016*, pages 31–42. ACM, 2016. Also at https://eprint.iacr.org/2016/199.

[Ram24] M. Rambaud. Faster asynchronous blockchain consensus and MVBA. Cryptology ePrint Archive, Paper 2024/1108, 2024. https://eprint.iacr.org/2024/1108.

[SGSK22] A. Spiegelman, N. Giridharan, A. Sonnino, and L. Kokoris-Kogias. Bullshark: DAG BFT protocols made practical. In H. Yin, A. Stavrou, C. Cremers, and

E. Shi, editors, *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security, CCS 2022, Los Angeles, CA, USA, November 7-11, 2022*, pages 2705–2718. ACM, 2022. Also at arXiv:2201.05677, http://arxiv.org/abs/2201.05677.