Preliminary Analysis on Fork Choice Rule for Beacon Chain

Daejun Park
Runtime Verification, Inc.

October 20, 2020

Introduction

The consensus mechanism of the Beacon chain consists of two major components: the Casper FFG and the fork choice rule, where the Casper FFG is for *passive global* consensus, while the fork choice is for *active local* consensus. The Casper FFG consensus (with the help of the weak subjectivity mechanism) is *global* in the sense that finalized blocks are never reverted (under the honest majority assumption), while it is *passive* in that it does *not* specify how to choose which blocks to be finalized. The fork choice consensus, however, is *local* in the sense that chosen blocks are not final and can be reverted later, while it is *active* in that it explicitly specifies how to choose blocks to vote and extend.

The Casper FFG is orthogonal to the fork choice rule, and can be coupled with an arbitrary fork choice rule. In an extreme case, for example, a *randomized* fork choice can be used, which is ultimately safe because it is not possible for adversaries to bias the pure randomness of the honest validators, but not efficient for liveness because the consensus is merely probabilistic where the probability is low.

Thus the major goal of the fork choice rule is to contribute to the liveness while minimizing the potential for being biased by adversaries. In other words, it is desired to have “good” choices to be likely preserved throughout the slots of an epoch, and “bad” choices to be quickly revoked. It is also desired to behave properly even in the presence of conflicting justified and/or finalized blocks, regardless of whether corrupted validators have been (or could be) slashed or not.

Recently, however, attacks on the current fork choice rule were found that can lead to indefinite liveness failure. The bouncing attack and timing attack are major examples. Although partial solutions for the specific attack vectors have been proposed, it is currently unclear whether they are sufficient and general enough to prevent other types of indefinite liveness failures.

In this report, we present a preliminary analysis on the current fork choice rule of the Beacon chain, and discuss certain issues against the liveness. We also propose a new fork choice rule to illustrate how to address some of the issues. Although the new fork choice rule enjoys more desired properties, it is still vulnerable to the censorship attack due to a fundamental aspect of the Beacon chain design. It is currently unknown whether there exists a provably secure fork choice rule under the current Beacon chain design.

Recap: Accountable Safety and Plausible Liveness of the Casper FFG

The accountable safety and the plausible liveness theorems were formally and mechanically proved in Coq. These theorems are general and hold for arbitrary fork choice rules. However, it is important to note that the precious meaning of these theorems is slightly weaker than the intuitive notion of the classic safety and liveness properties of the decentralized consensus. We clarify the subtlety below.

The accountable safety theorem states that: “if there exist conflicting finalized blocks, then there must *exist* at least ⅓ of validators who violated the slashing conditions.” Here note the *existence* of the “bad” validators. It does *not* say that those validators are necessarily slashable. Indeed, there exist scenarios that conflicting finalized blocks appear, but *no* validators can be slashed. (See below.)

The plausible liveness theorem states that: if there exist at least ⅔ of validators who have never violated the slashing conditions and created only “valid” attestations, then there is always a “chance” for the validators to finalize a new block without violating the slashing conditions. Here the “valid” attestation means that the source is justified and the target is a descendant of the source. This means that the plausible liveness holds for any fork choice rule that allows validators to produce only valid attestations. Here, note that the theorem does *not* say that newly finalized blocks will be always eventually produced. It only says that the validators are “able” to finalize a new block *without* violating the slashing conditions, but it does *not* say when and how they reach a consensus on which blocks to finalize. Indeed, there exist scenarios that no blocks are newly finalized in both synchronous and asynchronous network assumptions, notably the bouncing and timing attacks.

Violating slashing conditions without being slashed. There are two cases for validators who violated the slashing conditions to be not slashed: 1) validators exiting before being slashed, and 2) preventing slashing operations from being included in blocks. The first case appears in the long range attack. That is, when a conflicting block is revealed long later after all validators who violated the slashing conditions already exited and withdrew their stake, they cannot be penalized. However, this attack can be prevented by the weak subjectivity mechanism. (See the separate document on the weak subjectivity analysis.) The second case could appear in the short range attack, but this needs to be analyzed in a more general setting of censoring certain blocks to be included on chain. (We leave this as future work.)

Known Attacks on Current Fork Choice Rule

There exist two major known attacks against the current fork choice rule, that can cause *indefinite* liveness failure.

- **Bouncing Attack**: Only a partial mitigation was adopted, and there still exists an attack scenario that cannot be prevented by the current fork choice rule.

- **Timing Attack**: A mitigation was proposed, but it is not sufficient in the *adaptive* corruption model.[1] Another mitigation by introducing a bit of randomness was proposed, but it makes the validator nodes to behave nondeterministically.

Block Reordering Attack on Current Fork Choice Rule

A desired property of fork choice rules is to prevent the so-called block reordering (see the definition below). Intuitively, it allows "good" choices made by benign validators to be likely preserved throughout a certain period (e.g., the period of an epoch). It is relatively straightforward to prove the liveness of a fork choice rule that satisfies such a property.

Definition (Block Reordering). Suppose the fork choice rule identifies a canonical block b at slot n. We say that the block b is reverted or reordered if later the fork choice rule identifies another canonical block at slot $n' > n$, which is *not* a descendant of the block b.

Unfortunately, however, in the current fork choice rule, the block reordering is prevented only in a very limited setting. Below we first describe a strong condition that prevents the block reordering, and then illustrate how the block reordering becomes feasible in weaker conditions.

Proposition (Limited Block Reordering Tolerance). Assume the model of *full* synchrony and *static* corruption over *static* validators. Suppose that there exists a chain starting from a genesis block (or a finalized or weak subjectivity checkpoint block). If the chain gets votes from *all* honest validators who are assigned to the slots associated with the chain, then the probability of block reorganization is negligible.

Proof Sketch. Let the committee size of each slot be $3k$, then probabilistically, we have $2k$ honest validators for each slot. In order to revert, e.g., the last block of the chain (say at slot n), the block proposer at slot $n+1$ needs to propose a block by extending the block at slot $n-1$, and

[1] In addition to the existing attack conditions, the adversary dynamically can corrupt (bribe) all the 32 proposers for each epoch, so that the block proposers broadcast the proposed block only after 1/3 of slot passes. Then the weight on the block proposer has no effect, and the adversary can use the same timing attack mechanism to maintain the 50-50 split. Bribing all the block proposers for each epoch shouldn't be unrealistically expensive, (i.e., a realistic assumption in the adaptive corruption model), because the number of proposers for each epoch is constant, no matter how many validators exist. In other words, the mitigation proposal was meant to make the attack more expensive, requiring to corrupt $O(n)$ validators, but the attack can be still done by corrupting only $O(1)$ validators in the adaptive corruption model.

all validators (more precisely, more than 2.5k validators among the 3k validators) assigned to slot n+1 needs to vote for the new block. Since these behaviors do not follow the fork choice rule, it requires that ⅚ of the committee in the slot is corrupted, whose probability is negligible. Similar reasoning can be applied to calculate the probability of reverting more blocks, which is exponentially smaller in the number of blocks to be reverted.

Remark. The block reorganization tolerance property does not necessarily hold if the chain gets votes from ⅔ of the assigned validators, where they are not necessarily all honest. There are several scenarios.

- If the votes are not uniformly distributed over the slots, but say, only made for the first ⅔ of the slots, then the last ⅓ of the slots get no votes, and the blocks in those slots can be easily reverted if the block proposer for the next slot happens to be corrupted.
- If a half of the votes come from corrupted validators, then there could exist another chain that gets votes from (slightly less than) ⅔ of the assigned validators (due to double voting from corrupted validators[2]), in which case the whole chain can be reverted by using the timing attack.
- If a half of the votes are made by corrupted validators, but not immediately broadcasted, then the honest validators can see only ⅓ of votes for the chain, which is easier to revert by using the timing attack (with fewer violating the slashing conditions).

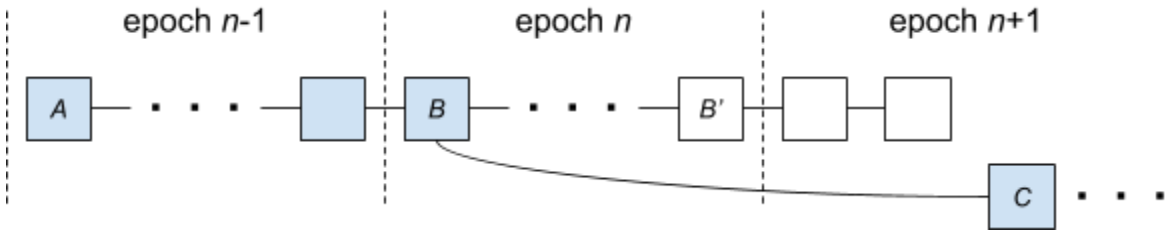
Remark. Note that the adversary can easily scatter the honest validators' votes, by proposing two blocks at a single slot, followed by the timing attack, so that the "canonical" block does not get votes from *all* honest validators even in a period of synchrony. Thus, the block reorganization tolerance property does *not* hold for the current fork choice rule. The fundamental problem of the current fork choice rule is that the decision is made based on the "*largest*" set of votes, instead of the "*majority*" set of votes. Even a single vote can be considered as "largest" if there exist no other votes, and thus can affect the decision, which hurts the security.

Specifically, in the partial synchrony setting, a chain can be reordered by the following attack.

Block reordering attack in partial synchrony. Suppose that there exists only a single chain until epoch n-1. At the beginning of epoch n, epoch n-1 is justified (i.e., block A justified). Now, the attackers start controlling the message delivery delay (i.e., entering a period of asynchronous), so that all the blocks proposed during epoch n are broadcasted immediately, but all attestations are delayed until the end of epoch n. Then, at the beginning of epoch n+1, the epoch n cannot be justified (i.e., block B not justified), even if there exists only a single chain (from block B to block B'), and more than ⅔ of validators voted for the chain. Now, the attackers keep delaying the attestations throughout epoch n+1. At some point, one of the attackers is designated to propose a block (probabilistically, within the first three slots of epoch n+1). Then,

[2] The corrupted validators clearly violated the slashing condition, but it is not necessarily clear whether all of them can be eventually slashed in the presence of continuous attacks.

the corrupted proposer proposes a block (i.e., block *C*) whose parent is the block at the first slot of epoch *n* (i.e., bypassing all the blocks throughout the epoch *n*), and immediately broadcasts the attestations to the proposed block. From then on, all honest validators will extend and vote for the new chain. At the end of epoch *n*+1[3], the attackers release all the attestations that were held, (i.e., entering a period of synchrony), and at the beginning of epoch *n*+2, the epoch *n*+1 with the new chain will be justified.



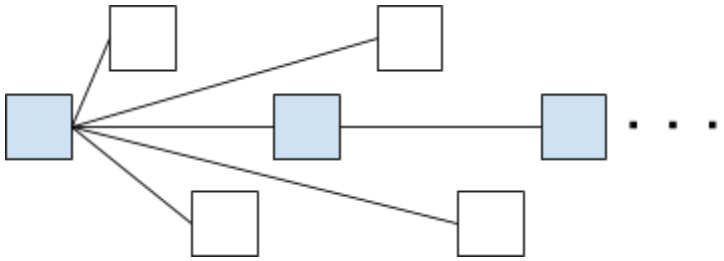
[3] The release can be done earlier, immediately after the new chain gets more than ½ of attestations.

Censorship Attacks under Partial Synchrony

Similarly to the block reordering attack, there exist censorship attacks (in partial synchrony) that censor blocks to be included on chain. The censorship attack is critical especially because it allows corrupted validators who violated slashing conditions to freely exit and withdraw without being slashed.

A simple censorship attack that allows only adversarial blocks to be included on chain indefinitely is as follows. Throughout each epoch, adversaries control the message delay so that for each slot to which a benign validator is assigned, the proposed block is delivered on time only to the committee for the slot, and attestations for the block are delayed until the end of the epoch. This allows adversaries to build a chain (within the epoch) that consists of only blocks proposed by corrupted validators, making all other blocks (i.e., ones proposed by benign validators) orphans even if they get sufficient votes. Then, at the end of the epoch, sufficient FFG votes will be available to justify the epoch, while the fork choice rule will select the tip of the adversarial chain. Adversaries can repeat this process indefinitely.

The following block tree illustrates the above attack. The leftmost block is an epoch boundary block, and the gray blocks are adversarial blocks proposed by corrupted validators. The white blocks are ones proposed by benign validators, where their parent block is equally the epoch boundary block because their benign block proposers could never see other blocks until the end of the epoch. Note that the committees will still vote for the white blocks, thus the epoch gets enough votes to be justified. Also, at the end of the epoch, the adversarial chain over the grey blocks will be the longest chain to be selected by the fork choice rule, thus to be extended in the next epoch. Note that probabilistically the length of the adversarial chain would be ⅓ * SLOTS_PER_EPOCH, assuming ⅓ of validators being corrupted.



Proposal for New Fork Choice Rule

In this section, we propose a new fork choice rule[4], inspired by the Streamlet protocol, to illustrate how to address the known issues of the current fork choice rule. Note that, however, this new fork choice rule still suffers from the issue caused by the fundamental design of the Beacon chain, "not every validator voting for every block." It is an open question whether there exists a fork choice rule (other than a randomized consensus) that is secure in partial synchrony, even if not every validator votes for every block.

The new fork choice rule is given as follows. (We note that this version is vulnerable to another type of bouncing attacks. A principled fix is presented in the "Fix for the bouncing attack" paragraph later. We did not incorporate the fix here to keep this initial presentation simple.)

At the beginning of each slot, each (honest) validator identifies the best chain(s) in his view as follows:

- Starting from the latest finalized[5] (or weak subjectivity checkpoint) block,
- Identify the longest chain(s) in terms of "notarized" blocks (i.e., the chain that contains the most "notarized" blocks).

Then, the block proposer proposes a block at the tip of the best chain (randomly choosing one of them[6], if multiple best chains exist.)

Then, each validator votes as follows:

- If the proposed block extends (one of) the longest chain(s) in his view, then votes for the block.
- Otherwise (i.e., the proposed block extends a non-longest chain, or no proposed block is received on time), votes for the tip of (any of) the longest chain(s) in his view.[7]

A block at slot n is "notarized", if it gets more than ⅔ of votes among the validators assigned into the slot n. (Note that a block at slot n can get votes from validators assigned to slot $m > n$, where such votes should *not* be counted for "notarization".)[8]

---

[4] It should be not hard to implement using the ingredients already implemented for the current fork choice rule.

[5] Note that it does not start from the latest *justified* block, which is vulnerable for the bouncing attack, as it is not unusual to have *conflicting* justified blocks.

[6] The random choice here is for making the attacker's bias harder. Any *deterministic* way of breaking the tie could be abused for the bouncing attack, thus a nondeterministic choice would be desired.

[7] This is not required for the security proof. Indeed, validators can just do nothing for this case in theory. But then they may lose the rewards, so this allows validators to vote the canonical chain in their view, and get the rewards if the chain indeed turns out to be the canonical one.

[8] This is because we allow validators to vote some blocks (to get the rewards if they can) in case that they don't agree on the proposed block, as mentioned earlier.

Remark. The security of this new fork choice rule requires the *static* corruption assumption. Otherwise, the adversary can corrupt (e.g., bribe) block proposers in some consecutive slots to reorder some (recent) blocks in his favor, which allows the adversary to censor transactions.

Analysis on New Fork Choice Rule

Security proof of the Streamlet protocol. Let us first review the security proof of the Streamlet protocol on which the new fork choice rule is based. There are some important properties of the Streamlet protocol.

P1. First, there could exist multiple notarized blocks of the same length. Suppose that a block *b* of length *n* is notarized at slot *k* (getting more than ⅔ votes). In slot *k*+1, another conflicting block *b'* of length *n* can be proposed and get ⅔ votes, if the votes for *b* have not yet been fully arrived at the beginning of slot *k*+1 (especially in a period of asynchrony).

P2. However, if a block *b* of length *n* is notarized at slot *k*, then it is *not* possible that another (conflicting) block *b'* of length *n*-1 is notarized at slot *k*' > *k*. This is because at least more than ⅓ honest validators already saw a notarized block of length *n*-1 at a previous slot (< *k*), and they wouldn't vote for *b'*, thus *b'* cannot get more than ⅔ votes.

P3. Therefore, if we have three consecutive blocks *b*, *b'*, and *b''* of length *n*, *n*+1, and *n*+2 at slot *k*, *k*+1, and *k*+2, respectively, then there exists no other conflicting blocks of length *n*+1. (We say that the block *b'* is finalized.) The proof is simple as follows. Suppose there exists another conflicting block *c* of length *n*+1 at slot *l*. Then *l* cannot be greater than *k*+2 nor less than *k*, because of the property P2. Furthermore, *l* cannot be *k*, *k*+1, nor *k*+2, because of the assumption that less than ⅓ of validators are malicious. This is a contradiction, thus we conclude that.

The above finality property (P3) simplifies the liveness proof. In a period of asynchrony, some undesired things can happen: e.g., no new block is notarized, or two (or multiple) chains grow alternately expanding with new notarized blocks. However, no notarized blocks can be reordered (due to P2), and no conflicting finalized blocks can exist (due to P3). Furthermore, once the network gets back to synchrony, a new finalized block can be always created when the block proposers for sufficiently many consecutive slots are honest.

Remark for the Streamlet protocol security. We emphasize two important ingredients of the protocol for the security. First, every validator votes for every single slot. This is essential for the property P2 (thus the property P3 as well). Second, a block proposer extends only a leaf block that was already notarized in his view. This is also critical for the property P2, as well as for the bouncing attack resilience. The bounding attack is hard because no matter how many votes are being held for a while and released later by malicious validators, only the leaf block of a chain can be newly notarized at a time. In other words, you cannot secretly hold more than one block over the same chain that can be notarized later at once.

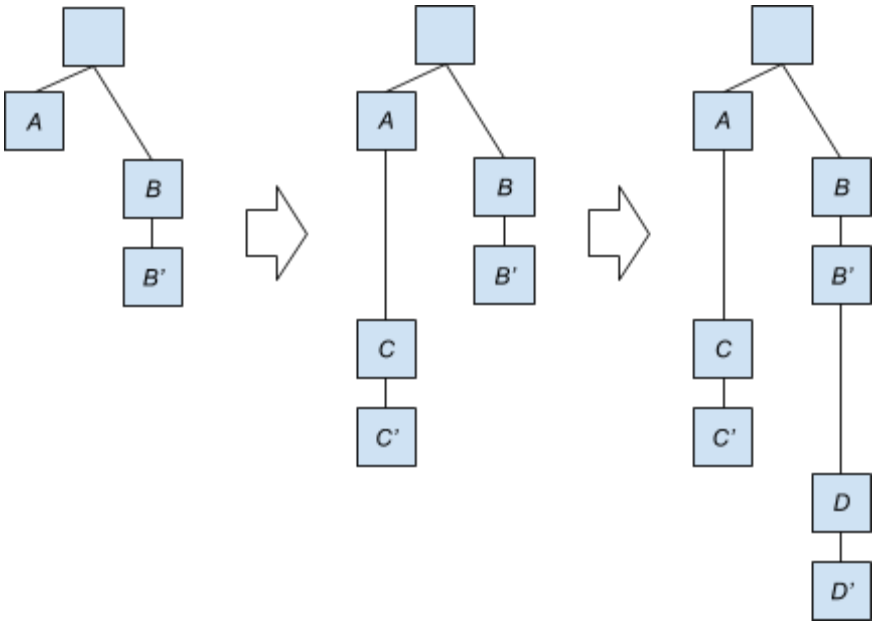
Fundamental differences between the new fork choice rule and the Streamlet protocol. There are two main differences:

- In our protocol, not all validators vote for every slot. Validators are grouped by multiple committees and a committee is assigned to each slot for voting. In other words, within an epoch, different slots have different disjoint validators for voting. This essentially

breaks the property P2, and thus notarized blocks can now be reordered in a period of asynchrony. Because of that, the new fork choice rule is prone to the short-range attack.

- In the new fork choice rule, a block proposer can extend a block even if it is not yet notarized in his view. This allows multiple blocks to be notarized later at once, making the fork choice rule to be vulnerable to the bouncing attack.

Short-range attack for the new fork choice rule. A small number of notarized blocks at the end of a chain could be reordered in a period of asynchrony as follows. In the period of those blocks being proposed and attested (and notarized), say from slot n to slot $n+k$, the attackers partition the network so that the committees for slots n through $n+k$ are totally disconnected from the committees for slots $n+k$ to $n+2k+1$. Then, the latter committees will propose, attest, and notarize new blocks bypassing those previous blocks. At the beginning of slot $n+2k+2$, the forked chain becomes longer than the original chain, the committee will keep extending the new forked chain, even if the network comes back to synchrony. ***NOTE***: The short-range attack, however, becomes harder as the number of blocks to be reordered increases, since the probability of the two committees being disjoint becomes much smaller. Roughly, the probability of succeeding in reordering notarized blocks over more than an epoch is very small (at least in the static corruption model).



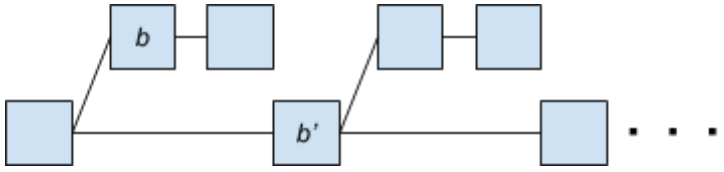
Bouncing attack for the new fork choice rule. Suppose that the committee size is $3x+1$, and initially the blocks A, B, and B' got $x+1$ votes (in the leftmost). At the end of the slot for B', the attackers reveal their hidden x votes for A, which makes A notarized, then the left chain is the longest notarized chain, and an honest proposer proposes C there. Suppose that the attackers somehow manage to have only $x+1$ (or at most $2x$) honest validators to vote for C.[9] Then C is

[9] This is possible in a period of asynchrony. But in a period of synchrony, it is not clear how feasible this would be.

not yet notarized, but still *C'* will be proposed in the left chain. During the slot for *C'*, the attackers reveal their hidden *x* votes for each of *B* and *B'*, immediately after *C'* gets *x*+1 votes from honest validators. Then, the right chain becomes the longest notarized chain, and *C'* will no longer get votes from honest validators. In the next slot, an honest proposer will propose *D* in the right chain, and the same process repeats.

Fix for the bouncing attack. Since the attack exploits the behavior that a new block can extend a non-notarized block, we can fix the fork choice rule so that a new block must extend a notarized block. However, naively forcing the strict extension rule could lead to a blowup in the block size (the number of transactions in a block) in a period of asynchrony, which is not desirable in practice. The idea of adopting the strict extension rule without blowing-up the block size is to have two parent links for each block---one for referring to a normal parent block, and another for referring to a latest notarized block.[10] Then, when identifying the longest notarized chain, we consider chains of blocks connected by the second type of links. This way, the length of a notarized chain grows monotonically, and the above bouncing attack is not possible. Note that this requires to modify the attestation condition as follows. When attesting a block, validators need to check whether the block referred by the second link is the leaf of (one of) the longest notarized chain(s) in their view; otherwise, they should not attest the block.

Censorship using the short-range attack. Although the short-range attack is practically feasible for only a small number of blocks, such a capability of reordering two or three consecutive blocks at the end of a chain makes it possible to censor blocks even in a static corruption model as follows. Suppose that ⅓ of validators are (statically) corrupted. Then, probabilistically, a corrupted validator would be assigned to propose a block for a slot for every three slots. Say that the goal of the censorship is to prevent slashing operations from being



included in any block during a period of a few epochs. Now, suppose that a block *b* is proposed that contains slashing operations. Then, the attackers partition the network, so that the block *b* and any other blocks that extend *b* are delivered only to the committees for those blocks, but not to any other (honest) validators. Soon after (probabilistically, two or three slots later), one of the corrupted validators would be assigned as a block proposer, and then he proposes and broadcasts a new block *b'* bypassing the block *b* and its descendants. Now, the block *b'* can get enough votes to be notarized, since the committee for block *b'* has never seen block *b* and its descendants. This process can be repeated until the end of the epoch, and at that point, the attackers are free to stop partitioning the network (i.e., the network getting back to the synchrony). Now, the epoch can be justified, because it has sufficient attestations (although

[10] In the ideal condition, these two links would refer to the same block. In any case, the parent block must be a descendant of the latest notarized block.

almost ⅔ of them were wasted for the bypassed blocks). The same process can be repeated for the future epochs until the corrupted validators who violated the slashing conditions exit.

Remark. This censorship attack is not possible if every validator votes for every slot. In our setting, the attack might be prevented by requiring additional conditions for "justifying" a chain, e.g., that a justified chain must have notarized blocks for more than ⅔ (or ½) of slots in each epoch. But such additional conditions might be abused for delaying the finalization (maybe indefinitely) if too strong, or could be still satisfied in the adaptive corruption model (i.e., by bribing block proposers) if not strong enough.