Blockchain Foundations **NTUA, Spring 2026**

Theory Exercise 4 Solutions

Problem 1

Recall the longest chain PoS protocol, Ouroboros Praos, we studied in class. Ouroboros Praos is *available* but not *accountable*. We modify Ouroboros Praos to include a slashing mechanism: if the same key is used to create two blocks for the same height, we slash the key. Is the modified protocol t-accountably safe? If yes, prove it and find t. If not, is there another slashing mechanism that could make the protocol t-accountably safe?

Recall that a protocol is *t-accountably safe* if, whenever safety is violated, at least t parties can be *irrefutably* proven to have deviated from the protocol.

The modified protocol is not t-accountably safe. An adversary holding more than half of the stake can break safety without ever equivocating, so the rule slashes nobody. It mines privately: starting from a block B on the public chain, it keeps its blocks secret and extends its own branch by one block for each slot it wins. With majority stake the private branch grows faster than the public chain, so its lead eventually exceeds any confirmation depth k. Releasing it then forces a reorg deeper than k, reverting an already-confirmed block and violating common prefix. Throughout this attack, the adversary places a single block at each height of its branch and never signs two blocks at the same height, so the equivocation rule finds nothing to slash.

No slashing mechanism can help. The failure is not special to the specific slashing mechanism. This is the *availability–accountability dilemma*: no protocol can be both available and t-accountably safe. Since Ouroboros Praos is available, no choice of slashing mechanism can make it accountable.

Problem 2

In this problem, we will consider the Simplex protocol run by n permissioned nodes, with a tunable quorum size q for notarization.

1. Suppose we want to maximize the resilience of the protocol, i.e., maximize the number of adversarial nodes f that can be tolerated such that the protocol is both safe and live. Derive the optimal quorum size q and show that the resulting optimal resilience is $f = n/3$.

   For safety, the quorum size should be large enough so that the adversary cannot notarize two conflicting blocks. For this, we use a quorum intersection argument. If two

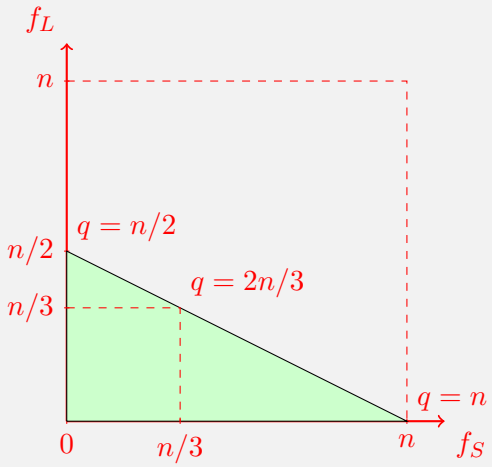
conflicting blocks are notarized, they have received q votes each. At most f adversary parties may vote for both blocks. With n parties in total, for both blocks to be notarized, we require $2q - f \leq n$. To prevent double notarization, we require $2q > n + f$. For liveness, the quorum should be small enough so that the adversary cannot prevent notarization by not voting. So we require $q \leq n - f$.
With both these requirements together,

$$\frac{n+f}{2} < q \leq n - f.$$

This implies that $\frac{n+f}{2} < n - f$, i.e. $f < n/3$. Thus the optimal resilience is $n/3$, and the resulting optimal quorum size is $q = 2n/3$.

2. Suppose you believe that an attack against safety is more likely than an attack against liveness (since a double-spend can provide significant rewards to the attacker). Hence, you want to tune Streamlet to increase the resilience against safety attacks, even at the expense of decreasing the resilience against a liveness attack. Can this be done? If so, exhibit and plot the tradeoff between the two resiliences. If not, explain why not.

By increasing the quorum size further, we can improve the resilience against safety attacks. From the analysis in part (a), Streamlet is resilient against $f_S < 2q - n$ adversary parties for safety attacks. But then, the liveness resilience is $f_L \leq n - q$. Thus, we can tune the quorum size to increase safety resilience and decrease liveness resilience, subject to the constraint $f_S + 2f_L < n$. This tradeoff is shown in the plot below.



Problem 3

Consider the Simplex protocol we presented in class. Consider the following variations of the protocol and in each case explain if the security (safety and liveness) of the protocol is preserved.

In each case, provide a proof that the protocol retains each virtue or a show a counterexample in which it fails.

1. We completely remove the timer of the protocol.
2. We change the timer from 3Δ to 2Δ.
3. We change the timer from 3Δ to 4Δ.

Safety holds in all three cases. The consistency proof of Simplex uses only two facts about honest nodes: (a) an honest node votes for at most one non-dummy block per iteration, and (b) an honest node never casts both $\langle \mathsf{finalize}, h \rangle$ and $\langle \mathsf{vote}, h, \perp_h \rangle$. Fact (b) holds because a node finalizes only if its timer has *not* fired and votes for the dummy only *when* its timer fires. This is a mutual exclusion that is independent of the timer's *duration*, and that holds vacuously if there is no timer at all. The quorum-intersection arguments then go through unchanged. So safety is preserved in (a), (b) and (c), and we only need to examine liveness.

(a) No timer: liveness fails. Without a timer an honest node never votes for a dummy block, so height h can be notarized only if the leader L_h's proposal gathers $\geq 2n/3$ votes. If L_h is faulty and simply stays silent, no proposal appears, no honest node votes at height h, and no block (real or dummy) is ever notarized there. Every honest node is stuck in iteration h forever and nothing past height $h-1$ is finalized. Since the adversary controls at least one node and leaders rotate, such an h will occur and a single crashed leader halts the protocol, so liveness fails.

(b) Timer 2Δ: liveness fails. Liveness must hold for every actual delay $\delta < \Delta$, so let the adversary fix some δ with $\frac{2}{3}\Delta < \delta < \Delta$ and take an iteration h with an *honest* leader. Let r be the time the first honest node enters h. The adversary makes every other honest node enter at r as well, but delays the leader so that it enters and proposes only at $r + \delta$ (delays the message delivery by δ). The honest nodes see the proposal and vote at $r + 2\delta$, and its notarization at $r + 3\delta$. But a node that entered at r times out at $r + 2\Delta$, which is *before* $r + 3\delta$ since $\delta > \frac{2}{3}\Delta$; it therefore casts $\langle \mathsf{vote}, h, \perp_h \rangle$ instead of $\langle \mathsf{finalize}, h \rangle$. So b_h never gets a finalize-quorum, and repeating this in every iteration, no block is ever finalized: liveness fails.

(c) Timer 4Δ: security is preserved. Lengthening the timer does not affect anything. Finalization is guaranteed as long as no honest timer fires before $t' + 3\delta$ (the latest time everyone enters $h+1$ with an honest leader), and $4\Delta > 3\Delta > 3\delta$ for every $\delta < \Delta$, so every honest node still finalizes the honest leader's block. The only effect is on iterations with a byzantine leader: a node now waits 4Δ instead of 3Δ before voting for the dummy, so a skipped iteration costs $4\Delta + \delta$ rather than $3\Delta + \delta$.

Problem 4

We modify the Simplex protocol to include a timestamp r in each non-dummy block. When an honest party creates a block, they put their clock's time as the timestamp r. In a valid chain, consecutive blocks have non-decreasing timestamps and no timestamp is in the future. We assume

that honest parties have sychronized clocks and the network is synchronous. We are interested in how much the timestamp of the finalized tip of an honest party can deviate from their clock's time. Compute an upper bound v such that any honest party's tip will not deviate from their clock more than v except with negligible probability. Your bound does not need to be tight.

Let T be an honest party P's clock and let $b_{h'}$, with timestamp r, be its finalized tip (being finalized it is non-dummy, so it carries a timestamp). We work in the synchronous model: delay $\leq \Delta$, synchronized clocks, timer 3Δ.

The tip never runs ahead of the clock. Since $b_{h'}$ is finalized it is notarized, so some honest node voted for it after checking its timestamp is not in the future. As clocks are synchronized and that vote was no later than P's view, $r \leq T$; only the staleness $T - r \geq 0$ remains to bound.

Staleness. Timestamps are non-decreasing, so $r \geq r_j$, the timestamp of the most recent honest-proposed block b_j in P's chain, and it suffices to bound $T - r_j$. In the synchronous model an honest leader always succeeds. Its block is finalized and all parties advance within $O(\Delta)$. So every iteration between b_j and P's current one must have a corrupt leader, else a newer honest block would lie in the chain. Leaders are corrupt independently with probability $f/n < 1/3$, so a run of $k = \omega(\log \lambda)$ consecutive corrupt leaders occurs only with negligible probability (union over the $\text{poly}(\lambda)$ iterations); hence at most k iterations separate b_j from now. Each lasts at most 4Δ (timeout 3Δ plus $\leq \Delta$ to propagate the dummy notarization), so

$$T - r \;\leq\; T - r_j \;\leq\; 4(k+2)\,\Delta \;=\; O(k\,\Delta), \qquad k = \omega(\log \lambda),$$

the two extra iterations covering b_j's finalization and the current one. (If no honest block exists yet, fewer than k iterations have passed and $T \leq 4k\Delta$ already.) We may thus take $v = 4(k+2)\Delta$; the bound is loose, the point being $v = O(k\Delta)$, governed by the longest run of corrupt leaders.