Dfinity Consensus, Explored

Ittai Abraham, Dahlia Malkhi, Kartik Nayak, and Ling Ren

VMware Research – {iabraham,dmalkhi,nkartik,lingren}@vmware.com

Abstract. We explore a Byzantine Consensus protocol called Dfinity Consensus, recently published in a technical report. Dfinity Consensus solves synchronous state machine replication among $n = 2f + 1$ replicas with up to f Byzantine faults. We provide a succinct explanation of the core mechanism of Dfinity Consensus to the best of our understanding. We prove the safety and liveness of the protocol specification we provide. Our complexity analysis of the protocol reveals the follows. The protocol achieves expected $O(f \times \Delta)$ latency against an adaptive adversary, (where Δ is the synchronous bound on message delay), and expected $O(\Delta)$ latency against a mildly adaptive adversary. In either case, the communication complexity is unbounded. We then explain how the protocol can be modified to reduce the communication complexity to $O(n^3)$ in the former case, and to $O(n^2)$ in the latter.

1 Introduction

The Byzantine agreement/broadcast problem was first studied by Pease, Shostak, and Lamport in [13,11]. Dolev and Strong gave in [6] the first polynomial protocol in the synchronous and authenticated setting, achieving $f + 1$ rounds and $O(n^3)$ communication. It has been shown that deterministic protocols are subject to an $f+1$ lower bound on round complexity [8,6] and a $\Omega(n^2)$ lower bound on communication complexity [5]. Randomized protocols have been introduced and achieved expected $O(1)$ round [14,3,7,10,1,4]. Only recently, communication complexity has been improved to $O(n^2)$ with $f < n/2$ [1] and to $O(n \cdot \mathsf{poly}\log(n))$ with $f < (\frac{1}{2} - \epsilon)n$ [4].

A recent publication of the DFINITY Technology Overviews Series [9] describes a blockchain consensus mechanism. At its core is a Byzantine fault tolerant state machine replication protocol in the synchronous, authenticated, and honest majority ($n \geq 2f + 1$) setting called Dfinity Consensus. In this report, we explore the Dfinity Consensus protocol to the best of our understanding. We will refer to the algorithm we extracted by the name Dfnty. (We do not attempt to review the entire Dfinity blockchain consensus mechanism. Topics such as group sampling, Sybil resistance, incentives, and gossiping are outside the scope of this report.) Dfnty follows the randomized paradigm and achieves the follows. Against an adaptive adversary, the protocol achieves expected $O(f \times \Delta)$ latency (where Δ is the synchronous bound on message delay, hence comparable to $O(f)$ rounds), and against a *mildly adaptive adversary* [12], expected $O(\Delta)$ latency. However, we show that in either case, an adversary can make the protocol's communication complexity unbounded. We then show a simple approach to improve the communication complexity to $O(n^2)$ per round, hence expected $O(n^3)$ against an adaptive adversary, and $O(n^2)$ against a mildly adaptive one.

We remark that in the same settings, Abraham et al. solve in [1] synchronous state machine replication against a static mildly adaptive adversary with $O(n^2)$ expected communication complexity and $O(\Delta)$ expected latency. Abraham et al. solve in [2] single-shot Byzantine Agreement, achieving the same complexities against an adaptive adversary, and their solution can be applied to state machine replication if one desires.

2 Model and Notation

Synchrony and signatures. The network is assumed to be synchronous, i.e., a message sent at time t arrives by time $t + \Delta$. We use $\langle x \rangle_p$ to denote a message x signed by replica p. For efficiency, it is customary to sign the hash digest of a message.

Local state. A replica p keeps track of all valid iteration-k blocks in a set $\mathcal{B}_k$.

Protocol for each replica p in iteration k.

1. **Propose a block and wait.** Arbitrarily select a certified $B_{k-1} \in \mathcal{B}_{k-1}$. Create $B_k := \langle v, r, \mathcal{C}(B_{k-1}) \rangle_p$ and broadcast it. Wait for 2Δ time.
2. **Vote for the best ranked block(s).** Let B_k be the best ranked block in $\mathcal{B}_k$. If p has not voted for B_k, vote for B_k and forward it to all replicas. If multiple blocks tie for the best rank, vote for and forward all of them. Repeat this step until a certificate $\mathcal{C}(B_k)$ for some B_k is received.
3. **Forward certificates.** Upon receiving a certificate $\mathcal{C}(B_k)$, broadcast it and enter the next iteration.
4. **Attempt to commit up to iteration $(k-2)$.** If all valid $B_{k-1} \in \mathcal{B}_{k-1}$ have the same predecessor block B_{k-2}, commit B_{k-2} and its predecessors. ▷ This step can be executed anytime after Step 1

Fig. 1. Dfnty consensus algorithm

Verifiable ranks. The Dfnty protocol runs in iterations. In each iteration, replicas are assigned random ranks that can be verified by other replicas. Let $r_p(k) \in \{0,1\}^\lambda$ denote the rank of replica p in iteration k (λ is a security parameter). $r_p(k)$ should be random, verifiable and unique. All three properties can be obtained from verifiable random functions (VRF). In Dfnty, a smaller VRF output is a better rank. The replica with the smallest rank in an iteration is called the *leader* of that iteration.

Block format. A block B_k produced by replica p in iteration k has the following format:

$$B_k = \langle v, r, \mathcal{C}(B_{k-1}) \rangle_p$$

where v denotes the proposed value, $r = r_p(k)$ is the proposer's rank, and $\mathcal{C}(B_{k-1})$ is a *certificate* for its predecessor block B_{k-1}. A certificate is simply a set of $f+1$ votes (using aggregate signatures for better efficiency). We call a block B *certified* iff $\mathcal{C}(B)$ exists. The first block $B_1 = \langle v, r, \perp \rangle_p$ contains no certificate since it has no predecessor. Every subsequent block B_k must specify a predecessor block B_{k-1} and include a certificate for it. If B_l is an ancestor of B_k ($l < k$), we say B_k *extends* B_l. Blocks from the same iteration are ranked by their proposers' ranks.

3 Protocol

Upon entering a new iteration, every replica makes a proposal and waits for 2Δ time (Step 1). After the wait, each replica votes for the best ranked block(s) it has received so far. Furthermore, whenever a replica votes for a block, it also forwards the block to all other replicas. If a replica is forwarded a block with a rank equal to or better than the best ranked block it has voted so far, it votes for the block (Step 2). This process continues until a certificate is formed. As soon as an honest replica obtains a certificate, it broadcasts the certificate to all other replicas and enters the next iteration (Step 3).

The protocol ensures three invariants.

I An honest leader's block will be uniquely certified in an iteration.
II In each iteration, at least one block, but possibly many, will be certified.
III At the end of iteration k, if all iteration-$(k-1)$ blocks extend the same B_{k-2}, then B_{k-2} is uniquely extendable. i.e., no other iteration-$(k-2)$ block can be extended from then on.

Invariant I. This invariant is the key to Dfnty's liveness. It makes sure an honest leader can make progress. To see why it holds, we first point out that replicas enter an iteration within Δ time apart from each other. The first honest replica that transitions to the new iteration forwards a certificate. Within Δ time, all other

honest replicas will receive the certificate and enter the new iteration. The certificate effectively serves as a synchronization message for all replicas to transition (roughly) together. Thus, if the leader is honest, the 2Δ time wait suffices for every replica to receive the leader's proposed block. Then, each of the $f+1$ honest replicas will vote for *only* the leader's block.

Invariant II. We have already discussed that, if the leader is honest, its block will get certified. If the leader is Byzantine, however, it may send its block to some but not all replicas, or send multiple equivocating blocks to different replicas. But no matter what it does, at least one block will be certified. To see this, observe that the best ranked block known to any honest replica will soon be received by all other honest replicas. This block will get certified unless some Byzantine replica presents an even better ranked block. Since there are only f Byzantine ranks, they can delay the formation of a certificate by $f\Delta$ time at most.

Committing a block. Although an honest leader's block will be uniquely certified, it is non-trivial for other honest replicas to find this out. Observe that a certificate requires only one honest signature ($f+1$ signatures in total). If a Byzantine replica proposes and votes for a worse-ranked block, honest replicas must account for the possibility that this vote comes from an honest replica, in which case, a conflicting certificate *may* have formed and been kept secret by Byzantine replicas. Therefore, Dfnty uses a weaker condition called unique-extensibility to commit blocks.

Invariant III. Invariant III is Dfnty's commit rule and key to safety. A replica p deems an iteration-$(k-2)$ block B_{k-2} as uniquely-extendable, and hence commits it, if at the end of iteration k, no iteration-$(k-1)$ block seen by p extends a different iteration-$(k-2)$ block (Step 4). Why does this condition suffice to commit? Observe that a certificate requires at least one honest vote, and that an honest replica votes for an iteration-$(k-1)$ block only in iteration $(k-1)$. Thus, if some honest replica votes for an iteration-$(k-1)$ block B'_{k-1} extending $B'_{k-2} \neq B_{k-2}$, then by the synchrony assumption, all replicas would have received B'_{k-1} at the end of iteration k (every iteration lasts at least 2Δ) and would not have deemed B_{k-2} as uniquely extendable. We formalize this argument in the proof of Lemma 1.

If the leader of an iteration is Byzantine, multiple blocks may get certified in its iteration. In this case, Dfnty follows the natural solution of postponing the commit decision to future iterations. As mentioned earlier, blocks are chained across iterations, i.e., an iteration-$(k+1)$ block includes a certificate for an iteration-k block. Eventually, a future iteration that has a uniquely certified/extendable block will commit that block and all its ancestors (including some block from iteration k).

3.1 Safety and Liveness

Lemma 1 (Unique-extensibility). *At the end of iteration k, if all iteration-$(k-1)$ blocks extend the same B_{k-2}, then B_{k-2} is uniquely extendable.*

Proof. Suppose B_{k-2} is not uniquely extendable. Then, at some point, another block B'_{k-2} is extended by some block B'_{k-1}, and at some later point, B'_{k-1} gets extended (certified). At least one honest replica h votes for B'_{k-1} in iteration $k-1$. When h votes (in iteration $k-1$), h forwards B'_{k-1} to all replicas, so all honest replicas receive B'_{k-1} at the end of iteration k. Thus, to every honest replica, not all blocks in iteration-$(k-1)$ extend B_k, a contradiction.

Theorem 1 (Safety). *Honest replicas always commit the same block B_k for each iteration k.*

Proof. Suppose an honest replica h commits B_k in iteration $j \geq k+2$ by committing block B_{j-2} and another honest replica h' commits B'_k in iteration $j' \geq k+2$ by committing $B_{j'-2}$. Due to the commit rule, B_{j-2} and $B_{j'-2}$ are both uniquely extendable (Lemma 1). If $j = j'$, B_{j-2} and $B_{j'-2}$ must be the same block (and extend the same B_k) in order for both to be uniquely extendable. Else, without loss of generality, assume $j' > j$. Since B_{j-2} is uniquely extendable, $B_{j'-2}$ extends B_{j-2}, and the two blocks extend the same B_k.

Theorem 2 (Liveness). *If the leader of iteration k is honest, its iteration-k block B_k is committed at the end of iteration $(k+2)$.*

Proof. The proof is straightforward from Invariant I. If the leader of iteration k is honest, its block B_k is uniquely certified and will thus be committed at the end of iteration $(k+2)$.

3.2 Efficiency Analysis

Adversarial model. We consider three adversarial models: static, adaptive and mildly adaptive. A static adversary needs to decide which replicas to corrupt before the protocol starts. An adaptive adversary can decide which replicas to corrupt any time during the protocol execution. A mildly adaptive adversary, defined in [12], can decide which replicas to corrupt during protocol execution, but needs $\Theta(\Delta)$ time to take control of a replica once it decides to corrupt that replica.

Latency. An adaptive adversary may choose to corrupt an honest leader immediately after it is elected and emits an equivocating message, causing a succession of $O(f)$ bad iterations to happen. Thus, the expected latency against an adaptive adversary is $O(f \times \Delta)$. We note that this attack can happen at most once, though for a large n, such a delay can be quite prohibitive.

Once an adversary has corrupted f replicas, or if the adversary is static or mildly adaptive, then in expectation, an honest leader is selected every two iterations. If a leader of iteration k is honest, the block it proposes is committed 2Δ time after the start of iteration $k+2$ (observe that Step 4 can be executed anytime after Step 1). Thus, in all these non-adaptive settings, the expected latency is three iterations plus 2Δ.

To calculate the expected latency per iteration, we consider two scenarios: (1) Optimistic case: when the actual communication delay is small compared to Δ, (2) Pessimistic case: when the actual communication delay is Δ. In the optimistic case, in each iteration, only Step 1 incurs a wait of 2Δ. All other steps execute at the actual network speed. Thus, the expected latency to commit is $2\Delta \times 3 + 2\Delta = 8\Delta$. In the pessimistic case, Step 1 incurs 2Δ. In Step 2, if the best t ranks belong to Byzantine replicas, which happens with 2^{t+1} probability, a certificate will be formed in $(t+1)\Delta$ time. In expectation, this step takes 2Δ time. Thus, the expected latency to commit is $4\Delta \times 3 + 2\Delta = 14\Delta$.

Communication complexity. We measure communication complexity by the number of signed messages sent by honest replicas for committing a block. The communication complexity in bits can be easily obtained by multiplying the length of a signature (in bits). If the leader of an iteration is honest, each honest replica proposes its own block (Step 1) and votes for exactly one block (Step 2). With the use of aggregate signatures, a certificate is a single signature. Thus, the communication complexity in an honest leader's iteration is $O(n^2)$. When the leader is Byzantine, unfortunately, Dfnty's communication complexity is unbounded. The Byzantine leader can propose an arbitrary number of equivocating blocks, all with the same rank. Every honest replica has to vote for all of these blocks, resulting in unbounded communication.

4 Improving the Communication Complexity

In this section, we describe a simple modification of Dfnty that achieves expected $O(n^2)$ communication complexity per iteration. Plugging this into the efficiency calculation in the previous section, this brings the expected communication complexities to $O(n^3)$ against an adaptive adversary, and $O(n^2)$ against a static or mildly adaptive adversary.

We noted earlier that in Dfnty, if a Byzantine leader proposes many blocks with the same (best) rank, every honest replica is required to vote for all of these blocks. To see why this is required, consider the naïve alternative where each honest replica votes for one block per proposer. Now, if a Byzantine leader proposes different equivocating blocks to different replicas in Step 1, none of them will get a certificate (since each gets one vote). Meanwhile, other proposers' blocks have inferior ranks, so they will not get any vote.

In other words, the unbounded communication is a result of two design decisions: (1) replicas do not vote for blocks with inferior ranks (to ensure Invariant I), and (2) Dfnty requires at least one certificate to form in each iteration (i.e., Invariant II). We observe that if either constraint is removed, we can reduce each iteration's communication complexity to $O(n^2)$. A protocol without the second invariant exists in the literature [1,2].

The protocol achieves consensus in expected $O(1)$ rounds with an expected $O(n^2)$ communication complexity with optimal resilience against a strongly rushing adaptive adversary. We refer readers to [1,2] for the details of this protocol.

In the following modification, we remove the first constraint. The key idea is the following: If an honest replica h receives two or more blocks from the same proposer p, it has detected p as Byzantine. If this happens, the replica forwards two of these blocks to all other replicas as an accusation of misbehavior against p. Whenever such an equivocation is observed, the effective rank of the replica is lower than the ranks of all replicas who did not equivocate. Thus, the effective rank can be computed as a pair $(\text{equivocate}_p(k), r_p(k))$ where $\text{equivocate}_p(k) = 1$ if replica p has equivocated in iteration k. With the modified rank, every replica still votes for the best-ranked proposer at every point in time (recall that lower ranks are better ranks).

Safety, liveness, and Invariant II. The proof of liveness (Invariant I) remains unchanged because an honest leader who does not equivocate is unaffected. The proof of safety (Invariant III) also remains unchanged because it only relies on (1) the fact that honest replicas vote for iteration-k blocks in iteration k, and (2) synchrony. Specifically, it does not rely on how blocks are ranked. For Invariant II, observe that if a leader is Byzantine, either its block will receive a certificate, or within 2Δ time honest replicas will learn of an equivocation and start voting for the next best-ranked proposer. Eventually, some block will get certified.

Efficiency analysis. For communication complexity, if the t best ranked proposers are Byzantine, which happens with $2^{-(t+1)}$ probability, then communication in this iteration is bounded by $(2t+1)n^2$. Thus, the expected communication complexity per iteration is $\sum_0^f \frac{(2t+1)n^2}{2^{t+1}} < n^2 \sum_0^\infty \frac{2t+1}{2^{t+1}} = O(n^2)$. Against an adaptive adversary, the expected number of iterations is $O(f)$, hence the expected communication complexity is $O(n^3)$. Against a static or mildly adaptive adversary, we require in expectation three iterations $+2\Delta$ time to commit; thus the expected communication is $O(n^2)$.

For latency, in the optimistic case, each iteration still requires 2Δ time, resulting in an expect 8Δ time to commit. In the pessimistic case, if t best ranked proposers are Byzantine, a certificate will be formed in $(2t+1)\Delta$ time. To see why, observe that a Δ time is required to form a certificate even in the absence of an adversary. Moreover, every Byzantine replica with a rank higher than honest replicas can delay a certificate formation by 2Δ time – one by not sending its block B to some honest replica h, and one by sending h an equivocating block B' when h is about to know B from other honest replicas. Thus, the expected latency of this step is $\sum_0^f \frac{(2t+1)\Delta}{2^{t+1}} < \sum_0^\infty \frac{(2t+1)\Delta}{2^{t+1}} = 3\Delta$. Step 1 still requires 2Δ, so each iteration requires expected 5Δ time. Against an adaptive adversary, this incurs $O(f \times \Delta)$ latency to commit in expectation. Against a mildly adaptive or static adversary, this results in an expected latency of 17Δ to commit.

Update from DFINITY. In private communication, the authors of DFINITY Technology Overview Series [9] responded to an earlier draft of this paper and mentioned that the DFINITY code uses this modified version of the protocol.

Acknowledgments

We thank Timo Hanke, Ben Maurer, Mahnush Movahedi, and Dominic Williams for useful discussions.

References

1. Ittai Abraham, Srinivas Devadas, Danny Dolev, Kartik Nayak, and Ling Ren. Efficient synchronous byzantine consensus. *arXiv preprint arXiv:1704.02397*, 2017.
2. Ittai Abraham, Srinivas Devadas, Danny Dolev, Kartik Nayak, and Ling Ren. Synchronous byzantine agreement with expected $O(1)$ rounds, expected $O(n^2)$ communication, and optimal resilience. Cryptology ePrint Archive, Report 2018/1028, 2018. `https://eprint.iacr.org/2018/1028`, To appear in Financical Cryptography and Data Security, 2019.
3. Michael Ben-Or. Another advantage of free choice (extended abstract): Completely asynchronous agreement protocols. In *Proceedings of the second annual ACM symposium on Principles of distributed computing*, pages 27–30. ACM, 1983.

4. TH Chan, Rafael Pass, and Elaine Shi. Communication-efficient byzantine agreement without erasures. *arXiv preprint arXiv:1805.03391*, 2018.
5. Danny Dolev and Rüdiger Reischuk. Bounds on information exchange for byzantine agreement. *Journal of the ACM (JACM)*, 32(1):191–204, 1985.
6. Danny Dolev and H. Raymond Strong. Authenticated algorithms for byzantine agreement. *SIAM Journal on Computing*, 12(4):656–666, 1983.
7. Pesech Feldman and Silvio Micali. An optimal probabilistic protocol for synchronous byzantine agreement. *SIAM Journal on Computing*, 26(4):873–933, 1997.
8. Michael J. Fischer and Nancy A. Lynch. A lower bound for the time to assure interactive consistency. *Information processing letters*, 14(4):183–186, 1982.
9. Timo Hanke, Mahnush Movahedi, and Dominic Williams. Dfinity technology overview series, consensus system. *arXiv preprint arXiv:1805.04548*, 2018.
10. Jonathan Katz and Chiu-Yuen Koo. On expected constant-round protocols for byzantine agreement. *J. Comput. Syst. Sci.*, 75(2):91–112, 2009.
11. Leslie Lamport, Robert Shostak, and Marshall Pease. The byzantine generals problem. *ACM Transactions on Programming Languages and Systems*, 4(3):382–401, 1982.
12. Rafael Pass and Elaine Shi. Hybrid consensus: Efficient consensus in the permissionless model. In *31st International Symposium on Distributed Computing, DISC*, 2017.
13. Marshall Pease, Robert Shostak, and Leslie Lamport. Reaching agreement in the presence of faults. *Journal of the ACM*, 27(2):228–234, 1980.
14. Michael O. Rabin. Randomized byzantine generals. In *Proceedings of the 24th Annual Symposium on Foundations of Computer Science*, pages 403–409. IEEE, 1983.