Let It TEE: Asynchronous Byzantine Atomic Broadcast with $n \geq 2f + 1$

Marc Leinweber
Institute of Information Security and Dependability (KASTEL), Karlsruhe Institute of Technology (KIT), Germany

Hannes Hartenstein
Institute of Information Security and Dependability (KASTEL), Karlsruhe Institute of Technology (KIT), Germany

Abstract

Asynchronous Byzantine Atomic Broadcast (ABAB) promises, in comparison to partially synchronous approaches, simplicity in implementation, increased performance, and increased robustness. For partially synchronous approaches, it is well-known that small Trusted Execution Environments (TEE), e.g., MinBFT's unique sequential identifier generator (USIG), are capable of reducing the communication effort while increasing the fault tolerance. For ABAB, the research community assumes that the use of TEEs increases performance and robustness. However, despite the existence of a fault-model compiler, a concrete TEE-based approach is not directly available yet. In this brief announcement, we show that the recently proposed DAG-Rider approach can be transformed to provide ABAB with $n \geq 2f + 1$ processes, of which f are faulty. We leverage MinBFT's USIG to implement Reliable Broadcast with $n > f$ processes and show that the quorum-critical proofs of DAG-Rider still hold when adapting the quorum size to $\lfloor \frac{n}{2} \rfloor + 1$.

2012 ACM Subject Classification Security and privacy → Distributed systems security

Keywords and phrases Byzantine Fault Tolerance, State Machine Replication, Trusted Execution Environments, Asynchrony

Funding This work was supported by funding from the topic Engineering Secure Systems of the Helmholtz Association (HGF).

1 Introduction

Atomic Broadcast primitives play a crucial role for Byzantine-fault tolerant State Machine Replication (SMR). A prominent example for Byzantine-fault tolerant SMR is PBFT [4]. PBFT operates in the partially synchronous timing model while tolerating f faulty processes in a set of $n \geq 3f + 1$ processes in total. Its seminal contribution is the introduction of a timer-based view change. Veronese et al. [9] found that, by implementing a signature service that assigns a unique counter value to each signature it produces in a Trusted Execution Environment (TEE), PBFT's communication complexity can be reduced and the fault tolerance can be increased to $n \geq 2f + 1$ while still tolerating Byzantine faults. TEEs are hardware extensions that are assumed to only fail by crashing and their integrity can be verified remotely. However, as shown by Miller et al. [8], the partially synchronous timing model has inherent limits and view change-based algorithms tend to be quite complex to implement. While it is known that any asynchronous crash fault-tolerant algorithm can be compiled to withstand Byzantine faults using TEEs [2, 5], there is no concrete TEE-based Asynchronous Byzantine Atomic Broadcast (ABAB) algorithm directly available yet and the performance benefits of TEEs for the asynchronous case can only be assumed. In this brief announcement, we transform DAG-Rider [7] to provide ABAB with $n \geq 2f + 1$ processes. DAG-Rider uses n Reliable Broadcast instances to disseminate process messages and to construct a directed acyclic graph (DAG) that captures the communication history of all processes. In a second step, each process derives consensus on the order of transactions using a

Common Coin, e.g. as proposed in [3]. We give a quick recap on the core ideas of DAG-Rider and explain the adaption *TEE-Rider*. Besides using TEE-based Reliable Broadcast and changing the required quorums from $2f+1$ to $\lfloor \frac{n}{2} \rfloor + 1$, we leave DAG-Rider unchanged. We show that the quorum-based arguments of DAG-Rider still hold for TEE-Rider.

2 TEE-Rider: Transforming DAG-Rider to $n \geq 2f+1$

We make use of the following definition of Atomic Broadcast for a set of processes $P, n := |P|$:

▶ **Definition 1** (Atomic Broadcast). *Each process $p_i \in P$ receives client transactions t via events* clientRequest(t)*. Correct processes deliver tuples (t, r, p_i), where t is a client transaction, $r \in \mathbb{N}_0$ a round number, and $p_i \in P$ the process that initially received t, satisfying the following properties:*
Agreement. *If a correct process $p_i \in P$ delivers (t, r, p_j), then every other correct process $p_k \in P, k \neq i$ eventually delivers (t, r, p_j) with probability 1.*
Integrity. *For each round $r \in \mathbb{N}_0$ and process $p_j \in P$, a correct process $p_i \in P$ delivers (t, r, p_j) at most once.*
Validity. *If a correct process $p_i \in P$ receives an event* clientRequest(t)*, then every correct process $p_k \in P$ eventually delivers (t, r, p_i) with probability 1.*
Total Order. *Let m_1 and m_2 be any two valid tuples that are delivered by any two correct processes $p_i, p_j \in P$. If p_i delivers m_1 before m_2, then p_j delivers m_1 before m_2.*

TEE-Rider adapts DAG-Rider [7] to implement asynchronous Byzantine Atomic Broadcast for a set of $n \geq 2f+1$ processes of which at most f may deviate arbitrarily from the protocol. The processes communicate via messages over authenticated point-to-point links. Each message sent will eventually be delivered. Each process is equipped with a TEE that implements MinBFT's unique sequential identifier generator (USIG) [9]. The USIG is used to implement Reliable Broadcast (informally: Atomic Broadcast without the Total Order property) with a fault tolerance of $n > f$ as, e.g., defined in [6]. An instance of the Reliable Broadcast abstraction has two functions: broadcast(r, m) to reliably broadcast exactly one arbitrary message m for round r to all processes in P, and delivered$()$ which returns all messages that were received by the instance since the last call to delivered$()$. Additionally, we assume a common coin scheme, e.g. as defined by Cachin et al. [3] using threshold signatures, that produces a uniformly distributed common random number p out of $\{p \mid p \in \mathbb{N}_0 \colon p < n\}$ for all correct processes and a name $i \in \mathbb{N}_0$ as soon as $f+1$ processes invoked toss(i); repetitive calls with same the i yield the same p. We require for the setup of the common coin and the USIGs, i.e., remote attestation and key exchange, synchrony and a public key infrastructure. We note that, for the common coin's threshold signature scheme, dealerless variants and those with an asynchronous setup exist. To guarantee liveness, i.e., validity, an infinite stream of client request events at each correct process is required.

The adapted DAG-Rider algorithm executed by a correct process $p_i \in P$ is shown in Algorithm 1. Quorum size changes are highlighted with a comment. The core of the approach is the construction and interpretation of a (local) DAG that captures received transactions and the observed communication sequence between processes. The DAG is structured in rounds and a round contains at maximum one vertex per process, i.e., n vertices. Rounds are addressed in an array style and the local view of a process p_i on the DAG is indicated by an index i. The very first round $DAG_i[0]$ is initialized with n hard-coded "genesis" vertices. A vertex in round r has two types of edges: strong edges point to vertices of round $r-1$ and weak edges point to vertices of any round $r' \leq r-2$. As soon as p_i received $\lfloor \frac{n}{2} \rfloor + 1$ *valid* vertices for a round r, i.e., $\lfloor\frac{n}{2}\rfloor+1$ vertices referencing $\lfloor\frac{n}{2}\rfloor+1$ vertices of round $r-1$ as strong edges ($v.strong$, l. 11), for which it also knows its predecessors (l. 15), p_i will *complete* round r and transition to round $r+1$. Now, as soon as p_i receives a client transaction, it will become the payload of a vertex v which is created and broadcast by p_i for round $r+1$ (ll. 44 and 21-27). The vertex v connects to all vertices p_i received for round r (l. 23). If p_i received vertices u for older rounds that are not reachable from the newly created vertex using the transitive closure of strong and weak edges (a 'path'), u will become a weak edge of v ($v.weak$, l. 26). The new vertex is broadcast using Reliable Broadcast instance i to all processes (l. 27). Every fourth round a so-called wave, consisting of four rounds, is completed (l. 19) and the DAG structure is used to derive a total order on the transactions (ll. 28-42). Each wave w has exactly one wave leader v which is chosen calling $coin.\text{toss}(w)$ from the vertices of w's first $\text{round}(w,1)$. The random number is used to select the process whose vertex is to be used as wave leader. If v was not (yet) received or there are no $\lfloor\frac{n}{2}\rfloor+1$ vertices in the w's fourth $\text{round}(w,4)$ that have v in their transitive closure of strong edges (a 'strong path'), the wave cannot be committed (l. 30). If wave w can be committed, process p_i checks first if there are wave leaders of waves w' between the last wave that was committed (variable $decidedWave$) and the current wave w that were received in the meantime and are connected to the leader of the wave $w'+1$ (ll. 33-36). The wave leaders are used as the root for a deterministic graph traversal to determine the total order of transactions (ll. 38-42).

3 Correctness Argument

Lemmas 1 and 2 of the original DAG-Rider publication [7] are crucial for Total Order and Agreement. The following Lemmas 2 and 3 show the corresponding results for a quorum size of $\lfloor\frac{n}{2}\rfloor+1$. Results for Integrity and Validity simply follow from the original paper.

▶ **Lemma 2.** *If a correct process $p_i \in P$ commits the wave leader v of a wave w when it completes wave w in $\text{round}(w,4)$, then any valid vertex v' of any process $p_j \in P$ broadcast for a round $r \geq \text{round}(w+1,1)$ will have a strong path to v.*

Proof. Since p_i commits v in $\text{round}(w,4)$, the direct commit rule is fulfilled (l. 30): $\exists U \subseteq DAG_i[\text{round}(w,4)]\colon |U| \geq \lfloor\frac{n}{2}\rfloor+1 \wedge \forall u \in U\colon \text{strongPath}(u,v)$. A valid vertex must reference at least $\lfloor\frac{n}{2}\rfloor+1$ distinct vertices of the previous round with a strong edge (l. 11). Thus, a process $p_j \in P$ broadcasting a valid vertex v_j for $\text{round}(w+1,1)$ selected at least $\lfloor\frac{n}{2}\rfloor+1$ vertices of $\text{round}(w,4)$ as strong edges for v_j. Any two subsets of size $\lfloor\frac{n}{2}\rfloor+1$ of a superset of size n intersect at least in one element. Thus, every valid vertex of a process broadcast for $\text{round}(w+1,1)$ must have at least one edge to a vertex of U, and, via U to v. As every valid vertex of $\text{round}(w+1,1)$ has a strong path to v, and every valid vertex of $\text{round}(w+1,2)$ connects to at least $\lfloor\frac{n}{2}\rfloor+1$ vertices of $\text{round}(w+1,1)$, by induction, any valid vertex v' of any process $p_j \in P$ broadcast for a round $r \geq \text{round}(w+1,1)$ has a strong path to v. ◀

▶ **Lemma 3.** *When a correct process $p_i \in P$ completes $\text{round}(w,4)$ of wave w, then $\exists V_1 \subseteq DAG_i[\text{round}(w,1)], V_4 \subseteq DAG_i[\text{round}(w,4)]\colon |V_1| \geq \lfloor\frac{n}{2}\rfloor+1 \wedge |V_4| \geq \lfloor\frac{n}{2}\rfloor+1 \wedge (\forall v_1 \in V_1, \forall v_4 \in V_4\colon \text{strongPath}(v_4,v_1))$.*

Proof. By use of Reliable Broadcast and validity checks in ll. 11 and 15, faulty processes are limited to omission faults. Thus, the *get-core* argument of Attiya and Welch [1, Sec. 14.3.1] still holds [1, Sec. 14.3.3]: Let $A \in \{0,1\}^{n\times n}$ be a matrix that contains a row for each possible vertex of $\text{round}(w,3)$ and a column for each possible vertex of $\text{round}(w,2)$. Let $A[j,k]=1$ if the vertex of process p_j of $\text{round}(w,3)$ has a strong edge to the vertex of process

Algorithm 1 TEE-Rider pseudocode for process $p_i \in P, n := |P|, \boldsymbol{n \geq 2f+1}$

```
1: state $DAG$: array of sets of vertices, $DAG[0]$ initialized with “genesis” vertices
2: state $r$: $\mathbb{N}_0$, initialized with 0
3: state $decidedWave$: $\mathbb{N}_0$, initialized with 0
4: state $transactionsToPropose$: queue of client transactions $t$, initialized empty
5: state $buffer$: set of vertices, initialized empty
6: state $rb$: array of $n$ reliable broadcast instances with delivered() and broadcast$(r, m)$
7: state $coin$: common coin instance with toss$(w)$
8: while True do
9:     for $k \leftarrow 0$ up to $n-1$ do
10:        for $m = (r', v) \in rb[k]$.delivered() do
11:            if $|v.strong| < \lfloor \frac{n}{2} \rfloor + 1$ then continue      ▷ adjusted quorum size
12:            $v.source \leftarrow p_k; v.round \leftarrow r'; v.delivered \leftarrow$ False
13:            $buffer$.add$(v)$
14:    for $v \in buffer$ do
15:        if $v.round > r \lor \exists u \in v.strong \cup v.weak : u \notin \cup_{r' \geq 0} DAG[r']$ then continue
16:        $DAG[v.round]$.add$(v)$
17:        $buffer$.remove$(v)$
18:    if $|DAG[r]| < \lfloor \frac{n}{2} \rfloor + 1$ then continue      ▷ adjusted quorum size
19:    if $r \mod 4 = 0$ then waveReady$(\frac{r}{4})$
20:    $r \leftarrow r + 1$
21:    wait until $\neg transactionsToPropose$.isEmpty()
22:    $v \leftarrow$ new vertex
23:    $v.block \leftarrow transactionsToPropose$.dequeue(); $v.strong \leftarrow DAG[r-1]$
24:    for $r' \leftarrow r-2$ down to 1 do
25:        for $u \in DAG[r']$ do
26:            if $\neg$path$(v, u)$ then $v.weak$.add$(u)$
27:    $rb[i]$.broadcast$(r, v)$
28: function $waveReady(w)$
29:    $v \leftarrow coin$.toss$(w)$ ▷ Returns $\perp$ if round$(w, 1)$ vertex of chosen process is not in $DAG$
30:    if $v = \perp \lor |\{u \mid u \in DAG[\text{round}(w, 4)] : \text{strongPath}(u, v)\}| < \lfloor \frac{n}{2} \rfloor + 1$ then return
31:                                                                      ▷ adjusted quorum size
32:    $leadersStack \leftarrow$ new stack; $leadersStack$.push$(v)$
33:    for $w' \leftarrow w-1$ down to $decidedWave + 1$ do
34:        $u \leftarrow coin$.toss$(w')$
35:        if $u \neq \perp \land$ strongPath$(v, u)$ then
36:            $leadersStack$.push$(u)$; $v \leftarrow u$
37:    $decidedWave \leftarrow w$
38:    while $\neg leadersStack$.isEmpty() do
39:        $v \leftarrow leadersStack$.pop()
40:        $verticesToDeliver \leftarrow \{u \mid u \in \cup_{r'>0} DAG[r'] : \text{path}(v, u) \land \neg u.delivered\}$
41:        for $u \in verticesToDeliver$ in deterministic order do
42:            $u.delivered \leftarrow$ True; deliver $(u.block, u.round, u.source)$
43: upon $clientRequest(t)$
44:    $transactionsToPropose$.enqueue$(t)$
```

p_k of round$(w, 2)$ or p_j sends no vertex (or an invalid one) but p_k sends a valid vertex for round$(w, 2)$. As there are at least $\lfloor \frac{n}{2} \rfloor + 1 \leq n - f$ correct processes, each row of A contains at least $\lfloor \frac{n}{2} \rfloor + 1$ ones and A contains at least $n(\lfloor \frac{n}{2} \rfloor + 1)$ ones. Since there are n columns, there must be a column l with at least $\lfloor \frac{n}{2} \rfloor + 1$ ones. This implies there is a vertex v_l by process p_l in round$(w, 2)$ s.t. $\exists V_3 \subseteq DAG_i[\text{round}(w, 3)] \colon |V_3| \geq \lfloor \frac{n}{2} \rfloor + 1 \land \forall v_3 \in V_3 \colon \text{strongPath}(v_3, v_l)$. As at most f vertices in V_3 belong to faulty processes that may commit send omission faults for round$(w, 3)$ and $\lfloor \frac{n}{2} \rfloor + 1 \geq f + 1$, by quorum section at least one vertex of V_3 is received by any correct process $p_j \in P$ before it sends its vertex for round$(w, 4)$. Thus, every valid vertex in $DAG_i[\text{round}(w, 4)]$ has at least one strong edge to a vertex of V_3. Since v_l must be valid and thus has a strong edge to each vertex of a set $V_1 \subseteq DAG_i[\text{round}(w, 1)], |V_1| \geq \lfloor \frac{n}{2} \rfloor + 1$, any valid vertex of rounds $r \geq \text{round}(w, 4)$ has a strong path to every vertex, including V_1, reached by v_l via strong paths. Please note that the construction of the set V_1 is valid for all correct processes that complete the wave and, thus, represents the 'common core'. ◀

We believe that the "one-pager" algorithm of DAG-Rider and the ease of adaption for TEEs make it a perfect textbook example for TEE-based ABAB as well as a good starting point to explore TEE-based versions of DAG-Rider follow-ups and related approaches.

References

1 Hagit Attiya and Jennifer L. Welch. *Distributed computing - fundamentals, simulations, and advanced topics (2. ed.)*. Wiley, 2004. doi:10.1002/0471478210.

2 Naama Ben-David, Benjamin Y. Chan, and Elaine Shi. Revisiting the power of non-equivocation in distributed protocols. In Alessia Milani and Philipp Woelfel, editors, *PODC '22: ACM Symposium on Principles of Distributed Computing, Salerno, Italy, July 25 - 29, 2022*, pages 450–459. ACM, 2022. doi:10.1145/3519270.3538427.

3 Christian Cachin, Klaus Kursawe, and Victor Shoup. Random oracles in constantinople: Practical asynchronous byzantine agreement using cryptography. *J. Cryptol.*, 18(3):219–246, 2005. doi:10.1007/s00145-005-0318-0.

4 Miguel Castro and Barbara Liskov. Practical byzantine fault tolerance and proactive recovery. *ACM Trans. Comput. Syst.*, 20(4):398–461, 2002. doi:10.1145/571637.571640.

5 Allen Clement, Flavio Junqueira, Aniket Kate, and Rodrigo Rodrigues. On the (limited) power of non-equivocation. In Darek Kowalski and Alessandro Panconesi, editors, *ACM Symposium on Principles of Distributed Computing, PODC '12, Funchal, Madeira, Portugal, July 16-18, 2012*, pages 301–308. ACM, 2012. doi:10.1145/2332432.2332490.

6 Miguel Correia, Giuliana Santos Veronese, and Lau Cheuk Lung. Asynchronous byzantine consensus with 2f+1 processes. In Sung Y. Shin, Sascha Ossowski, Michael Schumacher, Mathew J. Palakal, and Chih-Cheng Hung, editors, *Proceedings of the 2010 ACM Symposium on Applied Computing (SAC), Sierre, Switzerland, March 22-26, 2010*, pages 475–480. ACM, 2010. doi:10.1145/1774088.1774187.

7 Idit Keidar, Eleftherios Kokoris-Kogias, Oded Naor, and Alexander Spiegelman. All you need is DAG. In Avery Miller, Keren Censor-Hillel, and Janne H. Korhonen, editors, *PODC '21: ACM Symposium on Principles of Distributed Computing, Virtual Event, Italy, July 26-30, 2021*, pages 165–175. ACM, 2021. doi:10.1145/3465084.3467905.

8 Andrew Miller, Yu Xia, Kyle Croman, Elaine Shi, and Dawn Song. The honey badger of BFT protocols. In Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi, editors, *Proc. 2016 ACM SIGSAC Conf. on Computer and Communications Security, Vienna, Austria, 2016*, pages 31–42. ACM, 2016. doi:10.1145/2976749.2978399.

9 Giuliana Santos Veronese, Miguel Correia, Alysson Neves Bessani, Lau Cheuk Lung, and Paulo Veríssimo. Efficient byzantine fault-tolerance. *IEEE Trans. Computers*, 62(1):16–30, 2013. doi:10.1109/TC.2011.221.