Brief Announcement: Polygraph: Accountable Byzantine Agreement

Pierre Civit
UPMC, Paris 6, France
pierrecivit@gmail.com

Seth Gilbert
National University of Singapore, Singapore
seth.gilbert@comp.nus.edu.sg

Vincent Gramoli
The University of Sydney, Australia
EPFL, Lausanne, Switzerland
vincent.gramoli@sydney.edu.au

Abstract

In this paper, we introduce *Polygraph*, the first accountable Byzantine consensus algorithm. If among n users $f < n/3$ are malicious then it ensures consensus, otherwise it eventually detects malicious users that cause disagreement. Polygraph is appealing for blockchains as it allows to totally order blocks in a chain whenever possible, hence avoiding double spending and, otherwise, to punish at least $n/3$ malicious users when a fork occurs. This problem is more difficult than it first appears. Blockchains typically run in open networks whose delays are hard to predict, hence one cannot build upon synchronous techniques [5, 1]. One may exploit cryptographic evidence of PBFT-like consensus [2], however detecting equivocation would be insufficient. We show that it is impossible without extra logs of at least $\Omega(n)$ rounds [3]. Each round of Polygraph exchanges $O(n^2)$ messages.

2012 ACM Subject Classification Security and privacy → Distributed systems security

Keywords and phrases Fault detection, cryptography, equivocation, consensus

Digital Object Identifier 10.4230/LIPIcs.DISC.2020.45

Funding Part of this work is funded by ARC projects DP180104030 and FT180100496.

The Accountable Byzantine Agreement problem. We consider n processes, $f < n$ are Byzantine. Let t_0 be $\lceil \frac{n}{3} \rceil - 1$. Processes are sequential and asynchronous. We assume a PKI and that the network is partially synchronous. A verification algorithm V takes as input the state of a process and returns a set G of undeniable *guilty* processes, that is, every process-id of G is tagged with an unforgeable proof of culpability. We define the Accountable Byzantine Agreement problem similarly to the Byzantine Agreement: each process begins with a binary *input* and outputs a *decision*, satisfying the three usual properties (agreement, validity, and termination), and that there exists a verification algorithm that can identify at least $t_0 + 1$ Byzantine users whenever there is disagreement.

▶ **Definition 1** (Accountable Byzantine Agreement (Acc)). *An algorithm solves Acc if each process takes an input value, possibly produces a decision, and satisfies the following properties:*

- Agreement: *If $f \leq t_0$, then every honest process decides the same value.*
- Validity: *If all processes are honest and have the same input value, then that is the only decision value.*
- Termination: *If $f \leq t_0$, every honest process eventually decides a value.*
- Accountability*: There exists a verification algorithm V such that: if two honest processes decide distinct values, then eventually for every honest process p_j, for every state s_j reached by p_j from that point onwards, the verification $V(s_j)$ outputs a set of size at least $t_0 + 1$, containing exclusively Byzantine processes.*

34th International Symposium on Distributed Computing (DISC 2020).
Editor: Hagit Attiya; Article No. 45; pp. 45:1–45:3
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Polygraph. Polygraph is the first accountable Byzantine agreement protocol. It builds upon DBFT [4], an efficient consensus algorithm for blockchains. The notation broadcast(TAG,m)$\rightarrow$ $msgs$ denotes that p_i sends a message to every other process, with message type TAG, message content m and location $msgs$ to store received messages. We assume that every message is signed by the sender so the receiver can authenticate it. Finally, we write "receive k messages" to explain "receive messages from k distinct processes".

Algorithm 1 The Polygraph Protocol.

bin-propose(v_i):
  $e_i = v_i$
  $r_i = 0$
  $\tau_i = 0$
  $\ell_i[0] = \emptyset$
  repeat:
    $r_i \leftarrow r_i + 1$; ▷ *increment round*
    $\tau_i \leftarrow \tau_i + 1$ ▷ *increment timer*
    $c_i \leftarrow ((r_i - 1) \mod n) + 1$ ▷ *rotate coordinator*
  ▷ Phase 1:
    bv-broadcast(EST[r_i], e_i, $\ell_i[r_i - 1]$, i, bvs_i)
    if $i = c_i$ **then** ▷ *coordinator rebroadcasts*
      wait until ($bv_i[r_i] = \{w\}$) ▷ *bv-delivered bvs*
      broadcast(CO[r_i], w) $\rightarrow$ $msgs_i$
    start-timer(τ_i)
    wait until ($bvs_i[r_i] \neq \emptyset \wedge$ timer expired)
  ▷ Phase 2:
    if (CO[r_i], w) $\in msgs_i$ from $p_{c_i} \wedge$
      $w \in bvs_i[r_i]$) **then** $aux_i \leftarrow \{w\}$
    else $aux_i \leftarrow bvs_i[r_i]$
    sig_i = sign(aux_i, r_i, i) ▷ *sign the messages*
    broadcast(ECHO[r_i], $aux_i[r_i]$, sig_i) $\rightarrow$ $msgs_i$
    wait until $vals_i$ = values($msgs_i, bvs_i, aux_i$) $\neq \emptyset$
  ▷ Decision phase:
    if $vals_i = \{v\}$ **then** ▷ *if one value, adopt it*
      $e_i \leftarrow v$
      if $v = (r_i \mod 2)$ **then** ▷ *if parity matches*
        if no previous decision by p_i **then** decide(v)
    else
      $e_i \leftarrow (r_i \mod 2)$ ▷ *otherwise, adopt parity bit*
    $\ell_i[r_i]$ = justify($vals_i, e_i, r_i, bvs_i, msgs_i$)

Rules:
1. Every message that is not properly signed by the sender is discarded.
2. Every message that is sent by bv-broadcast without a valid ledger after Round 1, except for messages containing value 1 in Round 2, are discarded.
3. On first discovering a ledger l that conflicts with a certificate, send ledger l to all processes.

bv-broadcast(MSG, val, l, i, bvs):
  broadcast(BVAL, $\langle val, l, i\rangle$) $\rightarrow m$ ▷ *bcast message*
  After round 2, and in round 1 if $val = 0$, discard
    all messages received without a proper ledger.
  upon receipt of (BVAL, $\langle v, \cdot, j\rangle$)
    if received $(t_0 + 1)$ messages (BVAL, $\langle v, \cdot, \cdot\rangle$) and
      (BVAL, $\langle v, \cdot, \cdot\rangle$) not yet broadcast **then**
      Let $l \neq \emptyset$ be any ledger in these messages.
      broadcast(BVAL, $\langle v, l, j\rangle$)
    if received $(2t_0 + 1)$ times (BVAL, $\langle v, \cdot, \cdot\rangle$) **then**
      Let $l \neq \emptyset$ be any ledger in these messages.
      $bvs \leftarrow bvs \cup \{\langle v, l, j\rangle\}$

values($msgs$, b_set, aux_set): ▷ *check messages*
  if $\exists S \subseteq msgs$ where the following conditions hold:
    (i) $|S|$ contains $(n - t_0)$ distinct ECHO[r_i] msgs
    (ii) aux_set is equal to the set of values in S.
     then return(aux_set)
  if $\exists S \subseteq msgs$ where the following conditions hold:
    (i) $|S|$ contains $(n - t_0)$ distinct ECHO[r_i] msgs
    (ii) Every value in S is in b_set.
     then return(V = the set of values in S)
  else return($\emptyset$)

justify($vals_i, e_i, r_i, bvs_i, msgs_i$): ▷ *compute ledger*
  if $e_i = (r_i \mod 2)$ **then**
    if $r_i > 1$ **then**
      return $\ell[r_i]_i = l$ s.t. (EST[r_i], $\langle v, l, \cdot\rangle$) $\in bvs_i$
    else return $\ell[r_i]_i = \emptyset$
  else return $\ell[r_i]_i = (n - t_0)$ signed messages
    from $msgs_i$ containing only value e_i
  if $vals_i = \{(r_i \mod 2)\} \wedge$ no previous decision
    by p_i in previous round **then**
    $cert_i = (n - t_0)$ signed messages from $msgs_i$
      containing only value e_i
    broadcast($e_i, r_i, i, cert_i$) ▷ *broadcast certificate*

The protocol proceeds in asynchronous rounds where processes maintain an estimate. Each round proceeds in two phases, after which a possible decision is taken. In the first phase, each process bv-broadcasts its estimate using a reliable broadcast service that guarantees while $f < n/3$ that: (i) every message broadcast by $t_0 + 1$ honest processes is eventually delivered to every honest process; (ii) every message delivered to an honest process was broadcast by at least $t_0 + 1$ processes. All processes then wait until they receive at least one message, and until an increasing timer expires. A rotating coordinator for each round broadcasts its estimate with a special designation. In the second phase, if a process receives a message from the coordinator, then it chooses the coordinator's value to "echo" it to everyone. Otherwise, it simply echoes all the messages received in the first phase. At this point, each process p_i waits until it receives ECHO messages from enough $(n - t_0)$ distinct processes where every value in those messages was also received by p_i in the first phase. Finally, the processes try to decide. If process p_i has only one candidate value v, then p_i adopts that value v as its estimate. In that case, it can decide v if it matches the parity of the round, i.e., if $v = r_i \mod 2$. Otherwise, if p_i has more than one candidate value, then it adopts as its estimate $r_i \mod 2$, the parity of the round.

Ledgers and certificates. In order to ensure accountability, we need to record enough information during the execution to justify any decision that is made. To this end, we define two types of justifications: (i) a ledger designed to justify adopting a specific value and (ii) a certificate to justify a decision. We attach ledgers to certain messages and discard any message containing an invalid or malformed ledger. We define a ledger for round r and value v as follows. If $v \neq r \bmod 2$, then the ledger consists of the $(n - t_0)$ ECHO messages, each properly signed, received in Phase 2 of round r that contain only value v (and no other value). If $v = r \bmod 2$, then the ledger is simply a copy of *any* other ledger from the previous round $r - 1$ justifying value v. A certificate for a decision value v in round r is $(n - t_0)$ echo messages, each properly signed, received in Phase 2 of round r that contain only value v.

Accountability. We now explain how the ledgers and certificates are used. In every round, when a process uses bv-broadcast to send a message containing a value, it attaches a ledger from the previous round justifying why that value was adopted (except in Round 1 where no ledger is available to justify 1). The bv-broadcast ignores the ledger for the purpose of deciding when to echo a message. When it echoes a message m, it chooses any arbitrary non-empty ledger that was attached to a message containing m. However, every message that does not contain a valid ledger justifying its value is discarded, with the following exception: in Round 2, messages containing value 1 can be delivered without a ledger.

Whenever there is only one candidate value received in Phase 2, a process adopts that value and either (i) decides and constructs a certificate or (ii) does not decide and constructs a ledger. In both cases, this construction simply relies on the signed messages received in Phase 2 of that round. If a process decides a value v in round $r > 1$, or adopts v because it is the parity bit for round $r > 1$, then it also constructs a ledger justifying why it adopted that value v. It accomplishes this by examining all the bv-broadcast messages received for value v and copying a round $r - 1$ ledger. This is always possible since any message that is not accompanied by a valid ledger is ignored.

Proving culpability. When a process decides in round r, it sends its certificate to all the other processes. Any process that decides a different value in a round $r' > r$ can prove the culpability of at least $\lceil n/3 \rceil$ Byzantine processes by comparing this certificate to its logged ledgers. We will say that a certificate (e.g., from p_1) and a ledger (e.g., from p_2) *conflict* if they are constructed in the same round r, but for different values v and w. We now discuss how to find conflicting certificates and ledgers. Assume that process p_i decides value v in round r, and that process p_j decides a different value w in a round $> r$ after the network stabilizes. If p_j does not decide v, then, by looking at the messages received in round $r + 1$ and $r + 2$, it can identify a ledger that conflicts with the decision certificate of p_i and hence can prove the culpability of at least $t_0 + 1$ malicious processes. (Proofs are in the report [3].)

References

1. Vitalik Buterin and Virgil Griffith. Casper the friendly finality gadget. Technical Report 1710.09437v4, arXiv, January 2019. `arXiv:1710.09437v4`.
2. Miguel Castro and Barbara Liskov. Practical Byzantine fault tolerance and proactive recovery. *ACM Transactions on Computer Systems*, 20(4), 2002.
3. Pierre Civit, Seth Gilbert, and Vincent Gramoli. Polygraph: Accountable Byzantine consensus. In *Workshop on Verification of Distributed Systems (VDS)*, June 2019. Available at `https://eprint.iacr.org/2019/587.pdf`.
4. Tyler Crain, Vincent Gramoli, Mikel Larrea, and Michel Raynal. DBFT: Efficient leaderless Byzantine consensus and its applications to blockchains. In *IEEE NCA*, 2018. URL: `http://gramoli.redbellyblockchain.io/web/doc/pubs/DBFT-preprint.pdf`.
5. Andreas Haeberlen, Petr Kouznetsov, and Peter Druschel. PeerReview: Practical accountability for distributed systems. *SOSP*, 2007.