Fast Byzantine Agreement

Nicolas Braud-Santoni*
EPFL, Switzerland
nicolas.braud-santoni@ens-cachan.fr

Rachid Guerraoui
EPFL, Switzerland
rachid.guerraoui@epfl.ch

Florian Huc
EPFL, Switzerland
florian.huc@gmail.com

ABSTRACT

This paper presents the first probabilistic Byzantine Agreement algorithm whose communication and time complexities are poly-logarithmic. So far, the most effective probabilistic Byzantine Agreement algorithm had communication complexity $\tilde{O}\left(\sqrt{n}\right)$ and time complexity $\tilde{O}(1)$. Our algorithm is based on a novel, unbalanced, almost everywhere to everywhere Agreement protocol which is interesting in its own right.

Categories and Subject Descriptors

C.2.4 [**Computer-Communication Networks**]: Distributed Systems

Keywords

Byzantine Agreement, randomized algorithm

1. INTRODUCTION

The Byzantine Agreement problem.

Given a system of size n in which a Byzantine adversary controls at most t nodes (referred thereafter as *Byzantine nodes*), the Byzantine Agreement problem [LSP82] is about having all non-Byzantine nodes (or *correct* nodes) reach an agreement. The constraints imposed on the agreement are that all the correct nodes have to agree on a single output, which the adversary cannot impose to be *bad*.

Here *bad* can be subject to various interpretations:

- when the output is a single bit, it is required to be the input of one of the correct nodes;
- when the output is a string of $O(\log n)$ random bits, the adversary should not be able to bias *too many* bits of the output[1].

Like [PR10, BOPV06, BO83, Rab83], we consider the latter case.

*On leave from ENS de Cachan, Brittany campus, France.
[1]Here, "too many" will be made precise later on.

Deterministic setting.

Lamport, Shostak and Pease [LSP82, PSL80] have shown that the Byzantine Agreement problem cannot be solved without transferable authentication (or some other form of non-equivocation) when $n \leq 3t$. Furthermore, they gave a solution whose time complexity is $t+1$. Later, Fischer and Lynch, in [FL82], proved that $t+1$ is a lower bound on the time complexity in the worst case, while a lower bound on message complexity of $\Omega\left(n^2\right)$ was given in [DR85].

The result of [LSP82, PSL80] was improved in [GM98], in which the authors proposed an algorithm using $t+1$ steps, and whose computation and message complexities are polynomial. The case where *transferable signatures* are available was considered in [DR85]; under this condition, algorithms to solve the Byzantine Agreement problem exist without any restriction on the number of Byzantine nodes t [LSP82, PSL80]. The authors also proposed an algorithm using $O\left(n+t^2\right)$ messages, and an optimality proof.

Randomized algorithms.

To circumvent these lower bounds on time and communication complexities, randomization is crucial. Randomized Byzantine Agreement algorithms with constant expected time complexity were first proposed in [Rab83, DPPU86]. In [PR10], an algorithm with constant expected time complexity and $\tilde{O}\left(n^2\right)$ communication[2] complexity was proposed, under the assumption that communication channels are *private*. This algorithm works for asynchronous systems, but tolerates only $t < n/4$ malicious nodes.

In [HKK08], the authors proved an $\Omega\left(\sqrt[3]{n}\right)$ lower bound on both message (for at least one node) and time complexities for Byzantine Agreement algorithms in synchronous systems, under some restrictive assumptions.

Almost-everywhere agreement.

A relaxed version of the Byzantine Agreement problem, namely the *almost-everywhere Byzantine Agreement* problem, was introduced by Dwork et *al.* in [DPPU86]. This problem is a relaxation of the classical Byzantine Agreement problem, in the sense that it only requires that all but an $O\left(\log^{-1} n\right)$ fraction of the nodes agree on a common output.

This problem was first efficiently solved in [KSSV06], in which the authors proposed an algorithm whose message and computation complexity are poly-logarithmic in n for each node, and so is the number of rounds required. Later, an efficient *almost-everywhere reduction* (construction of Byzantine Agreement from *almost-everywhere Byzantine Agreement*) was proposed in [KS09, KLST11], using $\tilde{O}\left(\sqrt{n}\right)$ bits per node and poly-logarithmic time. This yields an algorithm for Byzantine Agreement with the same complexity, which, up to our knowl-

[2]$\tilde{O}$ is the same as O up to a poly-logarithmic factor.

edge, was the most efficient Byzantine Agreement protocol in terms of communication complexity until the present paper.

Our contribution.

We propose a new *almost-everywhere to everywhere* algorithm with amortized communication complexity $\tilde{O}(1)$ per node, denoted **AER**. Its time complexity is constant for synchronous executions, with non-rushing Byzantine nodes (defined in Section 2.1), and $O\left(\frac{\log n}{\log\log n}\right)$ for asynchronous ones. Composed with an *almost-everywhere agreement protocol* (along the lines of [KSSV06]), this yields the most effective protocol for Byzantine Agreement to date, using poly-logarithmic communication and time; this novel protocol is denoted **BA**.

The high-level idea underlying **AER** is the following: each node starts with a candidate string. The hypothesis is that more than half of the nodes are both correct and have the same candidate string, *i.e.* the correct one. Each node starts to diffuse its candidate string during a first phase (called push phase, Section 3.1.1). Then in a second phase (pull phase, Section 3.1.2), the bogus strings are discarded so that each node keeps only the correct string. The originality of our protocol, compared to previous ones, is that we relax load-balancing and we introduce new sampler properties so as to reduce communication complexity.

Comparison with existing protocols.

The complexity of push-pull protocols[3], like [KLST11], is dictated by the complexity of the first phase and the size of the candidate lists it produces.

To yield a more efficient protocol (in communication), we propose a solution to limit the total number of candidate strings in the whole system, and a way to diffuse them at a lower cost. Moreover, we introduce a way for each node to filter push requests. However, a Byzantine adversary can seize control of several *Input Quorums*[4], associated to a few nodes, and force these nodes to verify an *almost-linear* number of strings: as such, **AER** is not *load-balanced*.

AER has additional properties that are quite distinctive:

- this algorithm remains *correct and efficient* under asynchrony;
- unlike many randomized protocols, success is guaranteed when there is no Byzantine fault;
- against a *non-rushing* adversary, the algorithm terminates in *constant* expected time.

BA is, up to our knowledge, the first Byzantine Agreement protocol with poly-logarithmic complexity in both time and communication. Figure 1 compares with the state of the art, under various models:

S(N)R synchronous model, with (non-)*rushing* adversary;
APC asynchronous model, with *private communication channels*.

Roadmap.

We introduce our model and some background notions in Section 2. Section 3 describes our protocol and Section 4 describes its analysis. We discuss some future work in Section 5.

[3]Push-pull protocols are defined more precisely in Section 2.3
[4]Quorums are defined in Section 2.2

Figure 1: Comparison with other protocols

(a) « almost-everywhere to everywhere »

	[KLST11]	**AER**	**AER**
Model	SR	SNR	Async
Time	$O\left(\log^2 n\right)$	$O(1)$	$O\left(\frac{\log n}{\log\log n}\right)$
Bits	$\tilde{O}\left(\sqrt{n}\right)$	$O\left(\log^2 n\right)$	$O\left(\log^2 n\right)$
Load-Balanced	Yes	No	No

(b) Byzantine Agreement

	[BOPV06]	[KLST11]	**BA**	[PR10]	[KS13]
Model	SR	SR	SR	APC	Async
Time	$O(\log n)$	Polylog	Polylog	$O(1)$	$\tilde{O}\left(n^{2.5}\right)$
Bits	$n^{O(\log n)}$	$\tilde{O}\left(\sqrt{n}\right)$	Polylog	$\Omega\left(n^2 \log n\right)$	?
n	$4t+1$	$3t+1$	$3t+1$	$4t+1$	$500t$

2. PRELIMINARIES

2.1 Model

We consider the model of Lamport, Shostak & Peace's original Byzantine Agreement paper [LSP82], further used in [KS09, KLST11]: a fully-connected network of n nodes with an adversary controlling t nodes.

Unlike these papers, we assume the network to be asynchronous, except in Lemma 8 and Lemma 9. We also require that each node possesses a *private* random number generator.

An event is said to occur *with high probability* (*w.h.p.*) iff it occurs with probability greater than $1 - O\left(n^{-3}\right)$.

Network.

The network is *fully-connected*, and communication channels are authenticated - the identity of the sender is known to the recipient; *transferable authentication* is not required, nor any weaker form of *non-equivocation*. However, we require the network to be *reliable*: a message sent (to a non-faulty node) will be eventually delivered. In the synchronous case, we also assume that a message sent during round r will be delivered during round $r+1$.

Adversary.

A node controlled by the adversary is called *Byzantine* and can deviate arbitrarily from the algorithm. Furthermore, the adversary has full knowledge of the network and can coordinate the nodes it controls.

In particular, the actions of Byzantine nodes can depend on any messages sent, especially those sent by correct nodes. Two variations are considered:

- A *rushing* adversary knows the messages sent by the correct nodes at a given time step before choosing which messages to send.
- A non-rushing adversary chooses the messages it sends at a given time step independently of the messages sent by the correct nodes during the same time step.

The results we present assume some arbitrary $\varepsilon > 0$ is fixed, and $(3+\varepsilon)\cdot t < n$; however other bounds on t are used and mentioned for related work. It is also assumed that $1/2+\varepsilon$ fraction of the nodes are both correct and know a common string g_{string}, taken with uniform probability. This is equivalent to the assumption that all but a $1/4$ fraction of the correct nodes know g_{string}. Such an assumption can be ensured by the use of the protocol presented in [KSSV06], as mentioned in the introduction.

Finally, as in [LSP82], we consider a non-adaptive adversary: corrupt nodes are chosen before the algorithm is executed.

Complexity.

We consider two metrics for the complexity of the algorithm:

- Time complexity is the number of steps taken before all correct nodes return an agreement value.
- Communication complexity is the total number of exchanged bits (in worst case) divided by the number of nodes. This is called *amortized*[5] complexity, and is the same as *worst case* complexity for *load-balanced* algorithms.

Furthermore, we use the $\tilde{O}$ notation, which is the same as O, up to a poly-logarithmic factor.

2.2 Samplers

Like [KLST11, KKK+, CD89], our work relies on the notion of *samplers*. The intuition is as follows:

- if nodes choose deterministically the nodes they contact, either there are a linear number of them (thus $\Omega(n)$ communication complexity) or there are few enough for the adversary to corrupt a majority;
- if uniformly-random sets (called *quorums*) are chosen, the byzantine nodes can follow the algorithm, but contact many disjoint sets, which would then need communicate amongst *quorums*: again, this yields unreasonably high *worst-case* complexity.

Samplers are a middle ground, in the sense that the choice of *quorums* is directed by both deterministically-known information (like the identity of a node), and random sources (either a local RNG[6] or g_{string}, which is known *almost-everywhere*).

Definitions.

First, we introduce some notations:

- $[n]$ denotes the set of integers from 1 to n.
- $[n]^d$ is the set of size d strings, with elements in $[n]$.
- $\mathscr{D}$ is the *agreement domain*, of cardinal $D = n^c$
- $\mathscr{R}$ is the domain of *random labels*, used in our algorithm. Its cardinal, R, is polynomial.

DEFINITION 1 ([KLST11]). *A function $\mathbb{S} : X \to Y$ is a (θ, δ)-sampler if for any set $S \subseteq Y$ at most a δ fraction of the inputs x have $\frac{|\mathbb{S}(x)\cap S|}{|S|} > \frac{|S|}{n} + \theta$.*

We define $H(i,x) = \mathbb{S}(i \cdot n + x)$ for $i \in \mathscr{D}$ and $x \in [n]$:
The separation in two variables will be used to define *push quorums* and *pull quorums* (Sections 3.1.1 and 3.1.2).

Furthermore, we denote by $H^{-1}(i,x)$ the set of nodes y such that $x \in H(i,y)$. A node x is said to be *overloaded* by H (for some constant a) if there is a $i \in \mathscr{D}$ such that $\left|H^{-1}(i,x)\right| > a \cdot d$

Lemmata.

The following lemma is about the existence of samplers:

LEMMA 1 ([KLST11]). *For every constant c, for $s = n$, $\delta = D^{-1}$ and any $\theta > 0$, there is a (θ, δ)-sampler $H : \mathscr{D} \times [n] \to [s]^d$ with $d = O\left(\frac{\log(1/\delta)}{\theta^2}\right)$ such that for all $i \in \mathscr{D}$, no $x \in [n]$ is overloaded.*

We further use the following lemma:

LEMMA 2 (SECTION 4.1). *For $\mathscr{R}$ with cardinality polynomial in n, there exists $d = O(\log n)$ such that there is a mapping $J : [n] \times \mathscr{R} \to [n]^d$ such that for any subset of $[n]$ of size $(1/2+\varepsilon)\cdot n$, whose elements are called* good nodes*, we have:*

1. *At most n elements of $[n] \times \mathscr{R}$ are mapped to a set containing a minority of* good *nodes.*
2. *For any $L \subset [n] \times \mathscr{R}$, s.t. $\forall x \in [n], |L \cap (\{x\} \times \mathscr{R})| \leq 1$ and $|L| = O\left(\frac{n}{\log n}\right)$, with $L^{\perp} = \{y \mid \exists r \in \mathscr{R} \text{ s.t. } (y,r) \in L\}$:*

$$\sum_{(x,r)\in L} \left| J(x,r) \setminus L^{\perp} \right| > \frac{2d|L|}{3}$$

Property 1 comes from [KLST11], and forbids the Byzantine adversary from corrupting too many potential quorums. Property 2 is a novel property that prevents the Byzantine adversary from "cornering" a set of nodes and isolate it from the rest of the network. It is used in Algorithms 2 and 3.

2.3 Pulls and Pushes

In our context, it is convenient to model communication as *pushes* and *pulls*, two ways according to which a node x can get information.

A push occurs when x receives information from other nodes without asking for it, whereas in a pull, x sends a request to one or more other nodes, and receives information as a consequence. Notice that nodes may ignore pushes and pull requests. Pushes have the advantage of requiring less communication. On the other hand, a node receiving a push has not selected the sender, enabling Byzantine nodes to flood the network.

In this work, we design filters according to which push requests are accepted, preventing the Byzantine nodes from harming **AER**:

- When pull requests are initiated by x, it yields some guarantees on the nodes to which pull requests are addressed.
  Here, they are selected randomly, ensuring *w.h.p.* (with high probability) a majority of correct nodes.
- As in [KS09], *pull requests* are filtered to prevent Byzantine nodes from triggering too many replies (poor worst case complexity).

3. ALMOST EVERYWHERE TO EVERYWHERE

The almost-everywhere to everywhere problem [KS09] consists of propagating a piece g_{string} of information that is detained by many nodes to all nodes of the system.

The main contribution of this paper is an algorithm which solves this problem *w.h.p.* with amortized communication complexity $\tilde{O}(1)$ under the hypothesis that g_{string} is $c \log n$ bits long for some large enough constant c, and that $2/3 + \varepsilon$ of its bits are uniformly random.

Together with the algorithm presented in [KSSV06], **AER** yields a Byzantine Agreement protocol, noted **BA**, with amortized complexity $\tilde{O}(1)$.

3.1 Overview of our protocol

Preconditions.

AER enables all the nodes to agree on a common string called g_{string}, under two assumptions. First, all nodes must share three sampling functions: I, H and J:

- I defines the Push Quorums used to diffuse candidate strings. Lemma 1 yields a (θ, δ)-sampler $I : \mathscr{D} \times [n] \to [n]^d$ for $\theta =$

[5]Note that the amortization is over the set of nodes, not time. However, *average* complexity would refer to averaging over possible (random) runs of the algorithm.
[6]Random Number Generator, assumed uniformly-random and private

$O(1)$, $\delta = n^{-c}$ and $d = O(\log n)$ such that no $x \in [n]$ is overloaded;
- H define Pull Quorums with the same properties;
- J generates Poll Lists during the pull phase. It has the properties described in Lemma 2.

Secondly, each node x knows a candidate s_x:

- s_x can be equal to g_{string}, random, set to a default value, or even chosen by the adversary.
- However, more than half of the nodes must be correct and know g_{string}, *i.e.* they have $s_x = g_{\text{string}}$. Under the assumption that $t < (1/3 - \varepsilon) \cdot n$, it is equivalent to assume that $3/4$ of the correct nodes know g_{string}.
- g_{string} is $c \log n$ bits long for some large enough constant c, and $2/3 + \varepsilon$ of its bits were chosen uniformly at random.

The protocol.

AER proceeds in two phases:

Push This first phase consists of several *pushes* which provide each node x with a list L_x of at most $O(n)$ candidates for g_{string}, containing g_{string} *w.h.p.* This phase has a message complexity of $\tilde{O}(n)$, and the sum of the sizes of the lists is $O(n)$, amortizing to $O(1)$ per node.

Pull Each node x sends a *pull* query to a set of sample nodes to verify each candidate $s \in L_x$, to find g_{string}. We will show how to answer the pull queries, so as to identify g_{string} without having high worst case complexity.

3.1.1 Push

To each pair of string s and node x, the sampler I assigns a set of $O(\log n)$ nodes. x may receive pushes for s from nodes in $I(s,x)$: the Push Quorum of x according to s. If more than half of the nodes of $I(s,x)$ push for s, s is added to x's candidates list, L_x.

Since nodes do not react to the reception of messages by sending messages, this phase is *impervious to flooding*: the adversary cannot increase the communication complexity of this phase by sending many candidate strings to all nodes.

However, flooding may increase the number of candidate strings for each node. The filter defined by I prevents this: Lemma 4 states that the sum of the size of the candidate list is $O(n)$, while Lemma 3 implies that the number of messages sent during the first phase by *any good node* is $\tilde{O}(1)$.

In Figure 2a, a node x adds a string s_1 to its list of candidate strings (which originally contains only s_x, its initial candidate string), and ignores another string s_2. Indeed x receives s_1 from more than half of the nodes of $I(x, s_1)$, while it receives s_2 from less than half of the nodes of $I(x, s_2)$.

3.1.2 Pull

Checking a string s involves simultaneously a *Poll List* $J(x, r_x)$ (where r_x is taken at random) which is deemed *authoritative*, and *Pull Quorums* of the form $H(\cdot, s)$. They can be seen as proxies, used to *forward and filter* x's pull requests, so as to prevent it from flooding the network.

In Figure 2b, a node x performs a pull request to verify a string $s \in L_x$. An arrow to a quorum represents a message sent to all the nodes of the quorum, and similarly, an arrow from a quorum means that all the correct nodes of the quorum send the message.

Algorithm 1: Sending *pull* requests

```
Data: $L_x$, list of candidates for node $x$
has_decided, a boolean denoting whether $x$ has decided or not
Result: String agreed upon, w.h.p.
Bit complexity: $O(|L_x| \cdot \log n)$ messages, of $O(\log n)$ size
Time complexity: 1 if all messages are sent in a single round.
                 $|L_x|$ otherwise
for $s \in L_x$ do
    $r_{x,s} \leftarrow$ UniformRand()
    Send Poll $(s, r_{x,s})$ to all nodes in $J(x, r_{x,s})$
    Send Pull $(s, r_{x,s})$ to all nodes in $H(s, x)$
Upon event ⟨Unicast.deliver | $w$ [Answer($s$)]⟩ do
    if $w \in J(x, r_{x,s})$ and $w$ hasn't sent another Answer($s$) message yet then
        $count_s$++
        if $count_s > 1/2 |J(x, r_{x,s})|$ then
            has_decided ← true
            $s_{\text{this}} \leftarrow s$
            return $s$
```

Sending queries.

As formalized in Algorithm 1, each node x verifies each string $s \in L_x$ by polling a set of nodes:

- x chooses a random string $r_{x,s}$ to define the Poll List $J(x, r_{x,s})$. A different random string is used *for each candidate string*.
- x sends a pull request (containing $r_{x,s}$) to the Poll List $J(x, r_{x,s})$ and its Pull Quorum $H(s, x)$.

Answering.

This second part corresponds to Algorithms 2 and 3.

- A node $y \in H(s,x)$ forwards a request received from x iff s is its initial candidate string (s_y). The request is forwarded to the nodes in $J(x, r_x)$ through their Pull Quorums.
- A node z in the Pull Quorum of $w \in J(x, r_x)$ ($z \in H(s, w)$) forwards the request to w iff $s = s_z$ and z received the request from more than half of the nodes of $H(s,x)$.
- Finally, a node $w \in J(x, r_x)$ replies to a pull request from x if:
  1. the pull request was received from a majority of $H(s,w)$;
  2. either it one of its pull requests was answered (thus w knows g_{string} *w.h.p.*), and s_w was changed accordingly;
  3. or it currently has received less than $\log^2 n$ pull requests.

If x receives answers from a majority of nodes in $J(x, r_x)$, s is deemed to be the global string.

4. ANALYSIS OF AER

In this section, we prove that **AER** brings all correct nodes to agree on g_{string} *w.h.p.*, with poly-logarithmic time and space complexity; but first, we must prove Lemma 2.

4.1 Proof of Lemma 2

In this section, we use graph theoretic considerations in order to prove Lemma 2:

Lemma 2. *For $\mathcal{R}$ with cardinality polynomial in n, there exists $d = O(\log n)$ such that there is a mapping $J : [n] \times \mathcal{R} \to [n]^d$ such that*

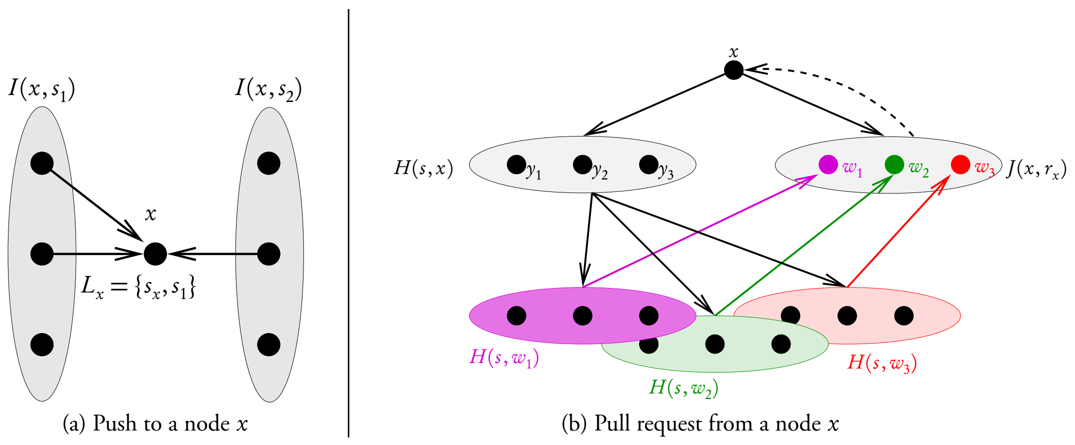

Figure 2: Push and Pull phases of **AER**

Algorithm 2: Routing *pull* requests

```
Data: The current node (called this) believes $g_{\text{string}}$ to be $s_{\text{this}}$
$FwCount_{s,x}$ and $Fw_2Count_{s,x}$, sets of counters, initialized at 0
Upon event $\langle Unicast.deliver \mid x\,[Pull(s,r)]\rangle$ do
    if $s=s_{this}$ and $this\in H(s,x)$ then
        // Keep track of senders to prevent flooding
        for $w\in J(x,r)$ do
            Send $Fw_1(x,s,r,w)$ to $H(s,w)$
Upon event $\langle Unicast.deliver \mid y\,[Fw_1(x,s,r,w)]\rangle$ do
    if $this\in H(s,w)$, $y\in H(s,x)$, $s=s_{this}$ and $w\in J(x,r)$ then
        $FwCount_{s,x}++$
        if $FwCount_{s,x}>1/2|H(s,x)|$ then
            Send $Fw_2(x,s,r)$ to $w$
            $FwCount_{s,x}\leftarrow-\infty$     // Forward only once
```

Algorithm 3: Answering *pull* requests

```
Data: The current node (called this) believes $g_{\text{string}}$ to be $s_{\text{this}}$
$Count_s$, set of counters, initialized at 0
$Polled$, set of pairs $(n,s)$ corresponding to received poll requests
Upon event $\langle Unicast.deliver \mid z\,[Fw_2(x,s,r)]\rangle$ do
    if $Count_s>\log^2 n$ then
        Wait for $has\_decided$
    if $this\in J(x,r)$, $z\in H(s,this)$ and $s=s_{this}$ then
        $Fw_2Count_{s,x}++$
        if $Fw_2Count_{s,x}>1/2|H(s,this)|$ and $(x,s)\in Polled$ then
            $Count_s++$
            Send $Answer(s)$ to $x$
            $Fw_2Count_{s,x}\leftarrow-\infty$     // Forward once
Upon event $\langle Unicast.deliver \mid x\,[Poll(s,r)]\rangle$ do
    if $this\in J(x,r)$ then
        $Polled\leftarrow Polled\cup\{(x,s)\}$
        if $Fw_2Count_{s,x}>1/2|H(s,this)|$ then
            // Necessary in the asynchronous case
            $Count_s++$
            Send $Answer(s)$ to $x$
            $Fw_2Count_{s,x}\leftarrow-\infty$     // Forward once
```

for any subset of $[n]$ of size $(1/2+\varepsilon)\cdot n$, whose elements are called good nodes*, we have:*

1. *At most n elements of $[n]\times\mathscr{R}$ are mapped to a set containing a minority of* good *nodes.*
2. *For any $L\subset[n]\times\mathscr{R}$, s.t. $\forall x\in[n], |L\cap(\{x\}\times\mathscr{R})|\le 1$ and $|L|=O\left(\frac{n}{\log n}\right)$, with $L^{\perp}=\{y\mid\exists r\in\mathscr{R}$ s.t. $(y,r)\in L\}$:*

$$\sum_{(x,r)\in L}\left|J(x,r)\setminus L^{\perp}\right|>\frac{2d|L|}{3}$$

Henceforth, these properties will be called Properties 1 and 2. In [KLST11], Lemma 4 state that Property 1 happen with probability at least $1/2$.

In the following subsection, we show that Property 2 holds *w.h.p.* It follows that, for all n big enough, there is a digraph satisfying both properties; this proves Lemma 2.

We use a graph-based formulation adapted from [Zuc97], which is a powerful tool for proving the existence of a family of samplers.

4.1.1 Graph-theoretic formulation

Model.

We consider *random digraphs* on vertex set $V=[n]\cup([n]\times\mathscr{R})$. We call $\mathscr{R}$ the set of labels. We call a vertex of $[n]\times\mathscr{R}$ a labeled vertex, and a vertex from $[n]$ an unlabeled one.

We use ∂L, a notion similar to the *border* of a subgraph, defined as follow.
For $L\subseteq[n]\times\mathscr{R}$ such that $\forall x\in[n], |L\cap(\{x\}\times\mathscr{R})|\le 1$:

$$\partial L=E_G\cap\left(L\times\left([n]\setminus L^{\perp}\right)\right).$$

In other words, ∂L is the set of edges from the labeled vertices in L to the unlabeled vertices in $[n]\setminus L^{\perp}$. Given this, we use a metric

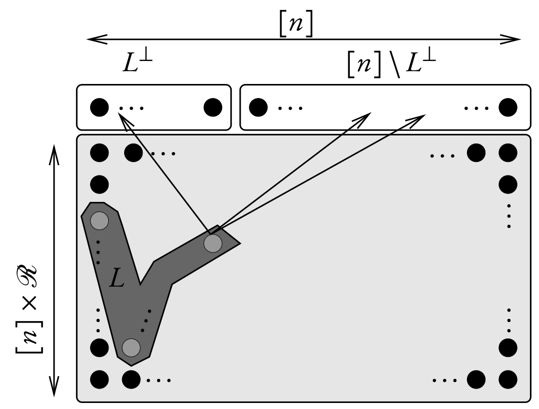

Figure 3: Our digraph model

closely related to the *isoperimetric number* [Bol88]:

$$i_\alpha = \min\left\{ \frac{|\partial L|}{|L|} \,\middle|\, 1 \le |L| \le \alpha n \right\}$$

The digraph G we consider (which is illustrated in the following figure) is taken uniformly such that:

1. The vertices of G in $[n]\times\mathscr{R}$ have exactly d out-neighbors (counting with multiplicity) in $[n]$ for $d = \log n$.
2. $E \subseteq ([n]\times\mathscr{R})\times[n]$: a vertex (labeled with a point in $\mathscr{R}$) is connected to unlabeled vertices.
3. E is uniformly random; it implies that edges are chosen independently.

Let $P(u,s)$ be the probability that there is a set $L \subset [n]\times\mathscr{R}$ with $|L| = u$ and $\forall x \in [n], |L \cap (\{x\}\times\mathscr{R})| \le 1$ such that $|\partial L| = s$.

Result.

$P(u,s) = o(2^{-n})$ for $0 < u \le \frac{n}{\log n}$ and $s < 2/3 \cdot d \cdot u$; hence, G satisfies Property 2 with probability $1 - o(n^2 2^{-n})$.

4.1.2 Proof

Since edges are taken uniformly, we can fix the set $L \subset [n]\times\mathscr{R}$ and have:

$$P(u,s) \le \binom{n}{u} \cdot \mathbb{P}[|\partial U| = s | P]$$

Enumeration.

We consider u vertices with d edges, which makes $n^{d\cdot u}$ possibilities for the edges.

There are $R^u \cdot \binom{d\cdot u}{s}(n-u)^s u^{du-s}$ sets E such that our property holds:

- We must choose u labels with values in $\mathscr{R}$:
  there are R^u possibilities.
- We must choose s edges (amongst the $d\cdot u$ in U) that will be in ∂U.
- These s edges go to vertices in $[n]\setminus L^\perp$:
  there are $(n-u)^s$ possibilities.
- The remaining vertices are connected to nodes in P:
  u^{du-s} possibilities.

Upper bound on the probability.

This yields the exact value of $\mathbb{P}[|\partial L| = s]$:

$$\mathbb{P}[|\partial L| = s] = \frac{R^u \cdot \binom{d\cdot u}{s}(n-u)^s u^{du-s}}{n^{d\cdot u}}$$

We can now upper-bound $P(u,s)$:

$$\begin{aligned}
P(u,s) &\le \binom{n}{u}\cdot\frac{R^u\cdot\binom{d\cdot u}{s}(n-u)^s u^{du-s}}{n^{d\cdot u}} \\
&\le \binom{n}{u}\binom{du}{s}\cdot R^u\cdot\left(\frac{u}{n}\right)^{du}\cdot\left(\frac{n}{u}-1\right)^s \\
&\le \left(\frac{ne}{u}\right)^u R^u\cdot\left(\frac{u}{n}\right)^{du}\cdot\left(\frac{due}{s}\right)^s\left(\frac{n}{u}-1\right)^s \\
&\qquad \text{using inequality } \binom{n}{x} \le \left(\frac{n\cdot e}{x}\right)^x \\
&\le \left(\frac{neR}{u}\right)^u\cdot\left(\frac{u}{n}\right)^{du}\cdot\left(\frac{d(n-u)e}{s}\right)^s \\
&\le \left(\frac{neR}{u}\right)^u\cdot(\log n)^{\frac{-dn}{\log n}}\cdot\left(\frac{dne}{s}\right)^s \text{ since } 0 < u \le \frac{n}{\log n} \\
&\le \left(\frac{neR}{u}\right)^u\cdot\log^{-n} n\cdot\left(\frac{dne}{s}\right)^s \text{ because } d = \log n \\
&\le (eR\log n)^{\frac{n}{\log n}}\cdot\log^{-n} n\cdot({}^3\!/_2\cdot de)^{{}^2\!/_3\cdot n} \\
&\qquad \text{because } \left(\frac{\alpha}{x}\right)^x \text{ increases until } \frac{\alpha}{e} \text{ and } s \le {}^2\!/_3\cdot d\cdot u \\
&\le \left[\frac{(eR\log n)^{\frac{1}{\log n}}\cdot({}^3\!/_2\cdot e)^{{}^2\!/_3}}{\sqrt[3]{\log n}}\right]^n \\
&\le \left[\frac{O(1)}{\sqrt[3]{\log n}}\right]^n \text{ because } (R\log n)^{\frac{1}{\log n}} \text{ is bounded} \\
P(u,s) &= o\left(2^{-n}\right)
\end{aligned}$$

□

4.2 Push phase

Lemma 3. *The communication complexity of the Push is $O(s\log n)$ on each node ($s = O(\log n)$ being the size of g_{string}).*

Proof. By the definition of I (Lemma 1), no node is overloaded. Hence, each correct node y sends its candidate string to $O(\log n)$ nodes, namely the nodes x such that $y \in I(s_y, x)$.
Therefore, during the Push, the number of messages sent *by any correct nodes* is $O(\log n)$, each containing a string having the same size as g_{string}. □

Lemma 4. *The sum of the sizes of the candidates list of the correct nodes is $O(n)$.*

Proof. A node y accepts a candidate string s iff more than half of the nodes of $I(s,y)$ sent it. By assumption, ${}^1\!/_2{+}\varepsilon$ fraction of the nodes are both correct and have g_{string} as initial string. Therefore, since I is a sampler, at most $O(n)$ quorums have a majority of nodes sending a string different from g_{string}. These quorums inject at most $O(n)$ wrong strings (in total) to the lists of candidate strings of the correct nodes. Therefore the sum of the size of the candidates list is $O(n)$. □

Lemma 5. *There is a constant c such that, when g_{string} is $c\log n$ bits long, and ${}^2\!/_3 + \varepsilon$ of the bits are uniformly random,* w.h.p. *each node of the system has g_{string} in its candidate list at the end of the* Push *phase.*

Proof. Looking at the Input Quorums, since I is a sampler, whatever are the nodes corrupted by the adversary and the knowledgeable nodes, at most $O(n)$ of the quorums do not have an correct and knowledgeable majority. If a node x does not have g_{string} in its candidate list L_x, it means that $I(g_{\text{string}}, x)$ does not have an correct and knowledgeable majority; we call such an Input Quorum *bad*. It is sufficient to prove that the probability of $I(g_{\text{string}}, x)$ to be bad is negligible.

g_{string} has length $c \log n$ for some constant c (that we choose thereafter), and the Byzantine nodes generate a $1/3 - \varepsilon$ fraction of its bits. Among all the choices, $O(n^c)$ of them lead to good Input Quorums for all of the nodes of the network, and $O(n)$ lead to at least one bad Input Quorum.

The probability for a chosen string to match a specific string leading to a bad Input Quorums is upper bounded by $2^{-2/3 \cdot c \log n}$. By the union bound over the n such strings, we get that the probability to choose one of them is less than $n^{-c'}$ for some constant $0 < c' < 2/3c$.

From this, we conclude that each correct node has g_{string} in its candidate list with probability greater than $1 - n^{-c'}$. □

4.3 Pull phase

It now remains to analyze the pull phase.

Lemma 6. *Any polling request (for g_{string}) is answered in $O\left(\frac{\log n}{\log\log n}\right)$ time steps.*

Proof. A node receives a message from an correct node in its poll list iff it is not overloaded or if it has already received an answer from a majority of nodes in its pull request.

Now, notice that to overload a node x whose original string g_{string}, the adversary must have $O\left(\log^2 n\right)$ of its nodes to send a pull request to x. This pull request will be considered by x iff it is for g_{string}. Therefore, the adversary must send $O\left(\log^2 n\right)$ pull requests corresponding to g_{string}. But then, *w.h.p.*, the quorum associated to g_{string} are all correct, which implies that the adversary can send pull requests at most once for each node it controls, as otherwise the pull requests will not be forwarded by the associated quorums. Therefore, the adversary can overload $O\left(\frac{n}{\log n}\right)$ nodes with request associated to g_{string} (each node can send $O(\log n)$ requests, and a node is overloaded if it receives more than $\log^2 n$ requests).

The adversary can also overload a node x whose original string is $s \neq g_{\text{string}}$ and that was chosen by the adversary. But to overload such a node, the adversary need to corrupt the quorum $H(x,s)$ and $H(a,s)$ for $\log^2 n$ nodes a controlled by the adversary, as otherwise the pull requests would not be forwarded. Therefore, in this case also, the adversary can overload at most $O\left(\frac{n}{\log n}\right)$ nodes.

The adversary knows[7] the nodes to which pull requests are addressed, before choosing its own pull requests. Therefore, it can overload all the nodes x' to which a given node x has sent pull requests. It can further overload all the nodes to which all x' have sent their pull requests and so on.

From the properties of J, we know that each set L of at most $\frac{n}{\log n}$ nodes send at least $2/3 \cdot d\,|L|$ pull requests to nodes outside L. This implies that by overloading $\frac{n}{\log n}$ nodes, the adversary can at most overload a sequence of length $O\left(\frac{\log n}{\log\log n}\right)$ in such a way. Therefore, in $O\left(\frac{\log n}{\log\log n}\right)$ steps, each node receives answer from its pull requests. □

[7]In the asynchronous (or synchronous rushing) model; the case of the (less general) synchronous model with non-rushing adversary is addressed in Lemma 8.

Lemma 7. *Any node decides on g_{string}* w.h.p.

Proof. There are two ways for a node not to decide on g_{string}:

1. its poll request for g_{string} isn't answered;
2. a poll request for another string s was answered first.

Lemma 6 ensures that, *w.h.p.*, 1. does not happen.

Assuming x decides on s, then $J(x, r_x)$ is composed in majority of nodes that are either Byzantine or that decided on s. Taking the first node to decide on s, the sample $J(x, r_x)$ must be composed mostly of Byzantine nodes. *W.h.p.* this doesn't happen, because r_x is uniformly random, J is a sampler, and the adversary chose the Byzantine nodes before r_x was chosen. □

Lemma 8. *Any polling request (for g_{string}) is answered in $O(1)$ time steps* w.h.p.*, if the adversary is non rushing.*

Proof. If the adversary is not rushing, it does not know which nodes an correct node address its pull requests to. Therefore, *w.h.p.*, each correct node pulls a majority of nodes that are both correct and not overloaded. It follows that *w.h.p.* each correct node receives an answer in a constant number of steps. □

4.4 Result

From this we obtain the following two lemmata:

Lemma 9. *For n nodes in a synchronous full information message passing model with a non-adaptive non-rushing Byzantine adversary which controls less than a $1/3 - \varepsilon$ fraction of the nodes, if more than $3/4$ of the correct nodes know a string g_{string} (random enough), there is an algorithm such that* w.h.p.*:*

- *At the end of the algorithm, each correct node knows g_{string}.*
- *The algorithm takes $O(1)$ rounds and $\tilde{O}(n)$ messages are exchanged in total.*

Lemma 10. *For n nodes in a asynchronous full information message passing model with a non-adaptive Byzantine adversary which controls less than a $1/3 - \varepsilon$ fraction of the nodes, if more than $3/4$ of the correct nodes know a string g_{string} (random enough), there is an algorithm such that* w.h.p.*:*

- *At the end of the algorithm, each correct node knows g_{string}.*
- *The algorithm takes $O\left(\frac{\log n}{\log\log n}\right)$ rounds and $\tilde{O}(n)$ messages are exchanged in total.*

Note that, to obtain an amortized communication complexity of $\tilde{O}(1)$, the condition that the algorithm is load-balanced was relaxed.

5. CONCLUSION

We propose an asynchronous *almost-everywhere to everywhere agreement* protocol with poly-logarithmic complexity, which yields the first poly-logarithmic algorithm for Byzantine Agreement. Future work could focus on *almost-everywhere Agreement*, for which no efficient asynchronous protocol is known. Another interesting and challenging question is to find the best complexity that is achievable by a load-balanced algorithm in the general case, and - more generally - characterize the trade-off between load-balancing and communication complexity.

6. REFERENCES

[BO83] M. 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 (PODC'83)*, pages 27--30. ACM, 1983.

[Bol88] B. Bollobas. The isoperimetric number of random regular graphs. *European Journal of Combinatorics*, pages 241--244, 1988.

[BOPV06] M. Ben-Or, E. Pavlov, and V. Vaikuntanathan. Byzantine agreement in the full-information model in $o(\log n)$ rounds. In *Proceedings of the thirty-eighth annual ACM symposium on Theory of computing*, pages 179--186. ACM, 2006.

[CD89] B. Chor and C. Dwork. Randomization in byzantine agreement. *Advances in COmputing Research 5: Randomness and Computation*, pages 443--497, 1989.

[DPPU86] C. Dwork, D. Peleg, N. Pippenger, and E. Upfal. Fault tolerance in networks of bounded degree. In *Proceedings of the eighteenth annual ACM symposium on Theory of computing (STOC'86)*, pages 370--379. ACM, 1986.

[DR85] D. Dolev and R. Reischuk. Bounds on information exchange for byzantine agreement. *Journal of the ACM*, 32(1):191--204, 1985.

[FL82] M.J. Fischer and N.A. Lynch. A lower bound for the time to assure interactive consistency. *Information Processing Letters*, 14(4):183--186, 1982.

[GM98] J.A. Garay and Y. Moses. Fully polynomial byzantine agreement for n> 3t processors in t+ 1 rounds. *SIAM J. Comput.*, 27(1):247--290, 1998.

[HKK08] D. Holtby, B.M. Kapron, and V. King. Lower bound for scalable byzantine agreement. *Distributed Computing*, 21(4):239--248, 2008.

[KKK$^+$] B. Kapron, D. Kempe, V. King, J. Saia, and V. Sanwalani. Fast asynchronous byzantine agreement and leader election with full information.

[KLST11] V. King, S. Lonargan, J. Saia, and A. Trehan. Load balanced scalable byzantine agreement through quorum building, with full information. *Distributed Computing and Networking*, pages 203--214, 2011.

[KS09] V. King and J. Saia. From almost everywhere to everywhere: Byzantine agreement with $\tilde{o}(n^{3/2})$ bits. *Distributed Computing*, pages 464--478, 2009.

[KS13] V. King and J. Saia. Byzantine agreement in polynomial expected time. In *45th Symposium on the Theory of Computing (STOC'13)*, 2013.

[KSSV06] V. King, J. Saia, V. Sanwalani, and E. Vee. Towards secure and scalable computation in peer-to-peer networks. In *47th Annual IEEE Symposium on on the Foundations of Computer Science (FOCS'06)*, pages 87--98, 2006.

[LSP82] L. Lamport, R. Shostak, and M. Pease. The Byzantine Generals Problem. *ACM Transactions on Programming Languages and Systems*, 4(3):382--401, July 1982.

[PR10] Arpita Patra and C. Pandu Rangan. Brief announcement: communication efficient asynchronous byzantine agreement. In *Proceedings of the 29th ACM symposium on Principles of distributed computing (PODC'10)*, pages 243--244, 2010.

[PSL80] M. Pease, R. Shostak, and L. Lamport. Reaching agreement in the presence of faults. *Journal of the ACM*, 27(2):228--234, 1980.

[Rab83] M.O. Rabin. Randomized byzantine generals. In *24th Annual Symposium on Foundations of Computer Science (FOCS'83)*, pages 403--409. IEEE, 1983.

[Zuc97] D. Zuckerman. Randomness-optimal oblivious sampling. *Random Struct. Algorithms*, pages 345--367, 1997.