Basalt: A Rock-Solid Byzantine-Tolerant Peer Sampling for Very Large Decentralized Networks

Alex Auvolat, Yérom-David Bromberg, Davide Frey, Djob Mvondo, François Taïani

BASALT: A Rock-Solid Byzantine-Tolerant Peer Sampling for Very Large Decentralized Networks

Alex Auvolat
alex@adnab.me
Deuxfleurs
France

Yérom-David Bromberg
david.bromberg@irisa.fr
Univ Rennes, Inria, CNRS, IRISA
France

Davide Frey
davide.frey@inria.fr
Univ Rennes, Inria, CNRS, IRISA
France

Djob Mvondo
barbe-thystere.mvondo-djob@inria.fr
Univ Rennes, Inria, CNRS, IRISA
France

François Taïani
francois.taiani@irisa.fr
Univ Rennes, Inria, CNRS, IRISA
France

Abstract

Recent large-scale Byzantine-Fault-Tolerant (BFT) algorithms provide scalability at a low cost by exploiting a secure Random Peer Sampling (RPS) service: a service that provides a stream of random network nodes where no attacking entity can become over-represented. Unfortunately, producing good peer samples untainted by Byzantine behavior in a large-scale network is particularly difficult, with existing solutions unable to withstand aggressive attacks. In this paper, we propose a novel RPS algorithm, BASALT, that implements what we have termed a *stubborn chaotic search* over node IDs to counter attackers' attempts at becoming over-represented. Our evaluation based on a theoretical analysis, Monte Carlo simulations, and experiments on a live cryptocurrency network shows that BASALT delivers close-to-optimal protection against malicious behaviors and outperforms state-of-the-art solutions by a wide margin.

CCS Concepts: • **Software and its engineering → Distributed systems organizing principles**; **Peer-to-peer architectures**; • **Theory of computation → Distributed algorithms**; • **Computer systems organization → Dependable and fault-tolerant systems and networks**.

Keywords: Gossip, Peer Sampling, Distributed System, Byzantine tolerance, Eclipse Attacks

ACM Reference Format:
Alex Auvolat, Yérom-David Bromberg, Davide Frey, Djob Mvondo, and François Taïani. 2023. BASALT: A Rock-Solid Byzantine-Tolerant Peer Sampling for Very Large Decentralized Networks. In *24th International Middleware Conference (Middleware '23), December 11–15, 2023, Bologna, Italy.* ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3590140.3629109

Middleware '23, December 11–15, 2023, Bologna, Italy

ACM ISBN 979-8-4007-0177-1/23/12…$15.00
https://doi.org/10.1145/3590140.3629109

1 Introduction

Byzantine-Fault-Tolerant distributed systems, such as cryptocurrencies [27] and smart contract platforms [2], can resist attacks from malicious participants (called *Byzantine* nodes), making it extremely difficult for an attacker to mislead honest participants. Guaranteeing this property in large networks is, however, particularly challenging due to the limited knowledge that each node can have of the rest of the system.

To address this challenge, epidemic Byzantine-Fault-Tolerant algorithms [18, 19, 30] exploit stochastic peer-to-peer exchanges [16, 23] to sample small sets of random peers repeatedly. They then use these samples to estimate the overall system state and ensure coordination and agreement between correct (i.e., non-Byzantine) nodes with high probability, despite malicious attacks by Byzantine nodes.

Epidemic BFT approaches critically depend on the availability of *good* network samples that contain as few Byzantine nodes as possible. Providing such samples is the role of a so-called *Byzantine-tolerant*, or *secure*, *random peer sampling* (RPS) service. When such a service is available, these algorithms have the potential to yield much higher throughput than PoW systems at a fraction of the cost [30].

Unfortunately, classical RPS algorithms [5, 21, 28, 33, 34] are not resilient to malicious behavior: Byzantine nodes can easily disrupt their execution by flooding honest nodes with the Byzantine identifiers present in the system. Left unchecked, this strategy has the potential to isolate honest nodes in a so-called *Eclipse attack* [20, 31] (i.e., an attack in which attackers completely isolate an honest node from other honest nodes by monopolizing all its in- and out-coming connections), or, even worse, to partition the system.

Current deployments of epidemic BFT algorithms, such as the AVA cryptocurrency platform [1], therefore rely on a Proof-of-Stake mechanism [17] to ensure that nodes are sampled in a secure way, i.e., that the cost for an attacker of

biasing samples in their favor is very high. Proof-of-Stake has, however, several known limitations [36]. In essence, Proof-of-Stake couples system membership with the economic investments of participants (in the form of token staking). We argue that such an abstraction is too restrictive and, in fact, not required. Particularly in the case of epidemic BFT algorithms, we show that the required Byzantine-tolerant random peer sampling service can be implemented more directly without resorting to Proof-of-Stake to ensure security.

In this paper, we revisit the problem of secure peer sampling in large-scale decentralized systems and propose BASALT, a novel Byzantine-tolerant random peer sampling algorithm. BASALT is specifically designed to resist Eclipse attacks and exhibits a significant improvement over state-of-the-art solutions such as Brahms [11] and SPS [22].

Similarly to Brahms [11], BASALT exploits what we have termed a *stubborn chaotic search*, a greedy epidemic procedure [35] towards random nodes that are implicitly defined using min-wise independent permutations [13] in a way that makes it extremely hard for malicious nodes to manipulate the decisions of correct ones. A stubborn chaotic search needs to be fed a regular stream of candidate nodes and produces a uniform sampling of the nodes seen so far. To produce these candidates, Brahms maintains a separate dynamic view that is periodically updated using epidemic push and pull exchanges with other nodes. Crucially, however, in Brahms this separate view receives only limited feedback from the chaotic search: node IDs flow primarily from the candidate view to the chaotic search (whose result is stored in entities called *samplers* in Brahms). Surprisingly, this apparently anodyne separation turns out to weaken substantially Brahms' robustness.

Building on this insight, BASALT eliminates this separate view and uses the current state of the chaotic search (i.e., the nodes seen so far whose IDs best match the current random hash functions) to drive the epidemic dissemination of node IDs. This design choice has a far-reaching impact on BASALT's performance and robustness, as our theoretical analysis and experimental evaluation show.

In the following, we comprehensively analyze BASALT and show that BASALT provides samples in which the proportion of malicious nodes is getting closer to its theoretical optimum. We complement our theoretical model with Monte Carlo simulations that confirm our analysis and demonstrate BASALT's advantage over the state-of-the art. Finally, we demonstrate the feasibility and concrete benefits of our technique by deploying a random peer sampling service derived from BASALT within a live cryptocurrency network [1, 30].

2 The BASALT approach

A random peer sampling (RPS) service [21, 33, 35] is a distributed algorithm that produces a random stream $(p_i)_{i\geq 0}$ of node identifiers, drawn uniformly from all nodes present in the network. RPS algorithms are designed to execute in large-scale peer-to-peer systems and typically assume that individual nodes only have a partial knowledge of other nodes in the system. (The node identifiers that a given node p knows at some point are usually called p's *view* in this context.) RPS algorithms are typically implemented using periodic stochastic exchanges of views between peers, an approach known as *gossiping*.

By default, traditional RPS algorithms usually tolerate churn (the continuous coming and leaving of nodes into and from a system) and crashes but often collapse when faced with malicious behavior. By contrast, a *secure* random peer sampling service is faced with the double task of *(i)* ensuring the largest possible diversity of peers in the stream $(p_i)_{i\geq 0}$, while *(ii)* limiting as much as possible the appearance of malicious nodes [7, 11, 22].

2.1 System Model

We assume a very large system composed of message-passing nodes that can either be honest (a.k.a. correct) or malicious (a.k.a. Byzantine). Byzantine nodes may deviate arbitrarily from the prescribed protocol to manipulate the decisions taken by correct nodes, for instance, to isolate correct nodes or to increase malicious nodes' representation in the peer sampler's output. We also assume Byzantine nodes may collude with one another. In particular, we assume a given Byzantine node knows all other Byzantine nodes and can use their identifiers in the messages it generates. We write Q the number of correct nodes in the system.

We consider a complete communication network where any node p can send a message to any other node q as soon as p knows q's identity (which is essentially what the Internet and the TCP/IP protocol stack provides). We assume a weak form of synchrony, ensuring that some fraction of the messages sent to a node by other non-malicious nodes arrive within a certain delay. Byzantine nodes may collude (share information, coordinate their behaviors) and may send arbitrary messages to an arbitrarily large number of correct nodes per time unit. They cannot, however, completely block the communication between two correct nodes or read the local memory of correct nodes.

We assume each node possesses a unique identifier. We will use the same notation to refer to a node and to its identifier. For simplicity's sake, we also assume that communication channels are reliable. Let us remark that unreliable communication channels with a given probability of failure (but not controlled by the attacker) would only influence our algorithm if they impact correct nodes more than Byzantine nodes. In this case, such channels could be modeled through what we have called the *attack force* of Byzantine nodes (see below).

2.2 Brahms and SPS

Brahms [11] and SPS [22] are two existing RPS algorithms designed to withstand Byzantine attacks. Both algorithms are

Table 1. Parameters of the BASALT algorithm.

Environment parameters		
n	Number of nodes	1000, 10 000
f	Fraction of Byzantine nodes	10%, 30%
Q	Number of correct nodes	$= (1-f)n$
F	Attack force (described in Sec. 4)	≥ 0
Algorithm parameters		
v	View size	50 to 200
τ	Exchange interval	1 time unit
ρ	Sampling rate (peers per time unit)	~ 1
k	Replacement count	up to $v/2$
Brahms parameters		
ℓ	View and sampler vector size	equal to v
α, β, γ	Relative contribution of the push, pull, and sampling to the view, cf. Eq. (2)	1/3

based on a classical RPS strategy [21, 33, 35], that is extended to correct for the over-representation of malicious nodes.

More concretely, Brahms and SPS both maintain a fixed-size view of node identifiers that is periodically updated using epidemic pull-push exchanges: in round k a node p randomly picks a set of exchange partners from its view ($\mathcal{V}_p^k$) to which it sends its own view (*push*), or request a view (*pull*). p then constructs its new view ($\mathcal{V}_p^{k+1}$) using the identifiers obtained through these exchanges. A classical strategy to construct $\mathcal{V}_p^{k+1}$ (that is not resilient to Byzantine behavior) consists in selecting n identifiers among those received, often extended by p's previous view:

$$\mathcal{V}_p^{k+1} \leftarrow \mathsf{rand}(\ell, \mathcal{V}_{\mathsf{push}}^k \cup \mathcal{V}_{\mathsf{pull}}^k \cup \mathcal{V}_p^k), \tag{1}$$

where ℓ is the fixed size of the view, $\mathcal{V}_{\mathsf{push}}^k$ and $\mathcal{V}_{\mathsf{pull}}^k$ are the set of identifiers obtained through push and pull exchanges during round k, and $\mathsf{rand}(\ell, S)$ randomly and uniformly selects ℓ elements from a set S.

Brahms and SPS modify the above update rule to prevent malicious nodes from flooding correct nodes with Byzantine identifiers:

SPS tries to build some statistical knowledge on node behavior, taking inspiration from social network analysis. Nodes that exhibit extreme indegree values are suspected and blacklisted. To detect extreme indegree values, nodes gather statistics on the identifiers they encounter, which necessitates some warming period. As a result, this mechanism is unable to cope with attacks where malicious nodes send so many messages that correct nodes do not have the time to gather sufficient statistics to block them before becoming isolated.

Brahms maintains besides $\mathcal{V}_p$, a vector $\mathcal{S}_p$ of entities termed *samplers* that realize a set of min-wise independent permutations on the stream of node IDs obtained from $\mathcal{V}_p$. (The following explanation assumes for simplicity that $\mathcal{V}_p$ and $\mathcal{S}_p$ have the same length ℓ.)

Brahms updates $\mathcal{V}_p$ in asynchronous rounds using push and pull exchanges. More concretely, in each round a node p

- pushes its own ID to a fixed number of nodes selected randomly from its current view $\mathcal{V}_p$;
- pulls the view $\mathcal{V}_q$ from another fixed number of random nodes from $\mathcal{V}_p$;
- collects the IDs obtained from push exchanges in a set $\mathcal{V}_{\mathsf{push}}$;
- collects the views pulled through pull exchanges in a set $\mathcal{V}_{\mathsf{pull}}$.
- The view of the next round is obtained by combining a random selection of $\mathcal{V}_{\mathsf{push}}$, $\mathcal{V}_{\mathsf{pull}}$, and $\mathcal{S}_p$. More concretely, Brahms balances the relative contribution of pushes and pulls by sampling the sets $\mathcal{V}_{\mathsf{push}}^k$ and $\mathcal{V}_{\mathsf{pull}}^k$ independently when constructing $\mathcal{V}_p^{k+1}$. It also re-injects earlier samples to limit the risk of isolation, i.e.
$$\mathcal{V}_p^{k+1} \leftarrow \begin{cases} \mathsf{rand}(\alpha\ell, \mathcal{V}_{\mathsf{push}}^k) \cup \\ \mathsf{rand}(\beta\ell, \mathcal{V}_{\mathsf{pull}}^k) \ \cup \\ \mathsf{rand}(\gamma\ell, \mathcal{S}_p^k), \end{cases} \tag{2}$$
where X_p^x for $x \in \{k, k+1\}$ is the value of the variable X_p at the round x, and α, β and γ are parameters of the protocol, such that $\alpha, \beta, \gamma \in [0, 1]$ and $\alpha + \beta + \gamma = 1$.
- Finally the node IDs contained in $\mathcal{V}_{\mathsf{push}}^k$ and $\mathcal{V}_{\mathsf{pull}}^k$ are fed to the samplers of $\mathcal{S}_p$ to implement a random chaotic search using min-wise independent permutations.

Brahms further assumes that Byzantine nodes can only send a limited number of push messages per time unit and blocks the view update if a peer receives more than a certain number of push messages during a given time slot.

2.3 The BASALT algorithm

BASALT takes inspiration from Brahms in that it also uses min-wise independent permutations based on random ranking functions to implement what we have termed a stubborn chaotic search. BASALT, however, completely eliminates the gossip view $\mathcal{V}_p$ of Brahms and uses the current state of the stubborn chaotic search to drive gossip pull/push exchanges and ensure the discovery of new node identifiers. This design choice considerably limits the nuisance power of Byzantine nodes and allows BASALT to remove any limits on the communication power of Byzantine nodes, which are allowed in our model to trigger exchanges much more often than correct nodes through the attack power parameter. Table 1 shows an overview of the parameters of our algorithm and its environment (the *Algorithm parameters* are known to the participants, while the *Environment parameters* are not). Algorithm 1 shows its pseudocode. For the sake of clarity, in the following, we use the generic term *node* to refer to protocol participants, but we use the term *peer* to refer to a node's (potential) neighbor.

BASALT nodes implicitly identify a dynamic target random graph by defining target neighbors using a set of random ranking functions. Then, each node greedily attempts to converge towards this implicit definition by repeatedly exchanging neighbor lists with other peers, discovering at each step peers that better match its ranking functions.

Identifying neighbors through ranking functions. Each node maintains a view, view[·], composed of v *slots*. For each slot, $i \in \{1, \ldots, v\}$, a random seed seed[i] defines a corresponding random *ranking* function, $\text{rank}_{\text{seed}[i]}(\cdot)$ (line 5 of Alg. 1, and Fig. 1). A node's i-th out-neighbor in the target graph is the (correct or malicious) node p that minimizes $\text{rank}_{\text{seed}[i]}(p)$. (As a simplifying shortcut, we will say that p is *closest to* seed[i], or *best matches* seed[i].) The remainder of the algorithm uses stochastic exchanges of node identifiers to search for this i-th out-neighbor. We call this search *chaotic* as it is driven by stochastic exchanges, and *stubborn* because the choice of the seed seed[i] by a correct node cannot be influenced by Byzantine nodes.

This "target" neighbor is not known by the local node, which instead stores in view[i] the identifier that has so far produced the smallest value of $\text{rank}_{\text{seed}[i]}(\text{view}[i])$ amongst those seen since selecting seed[i]. At startup, each node selects the best matching peers, view[i], from a set of bootstrap peers (line 6).[1] Nodes then periodically exchange the current contents of their views (lines 7-9) to discover new peers that can serve as better matches for the slots in their views. Specifically, every τ time units (exchange interval), each correct node selects a random peer from its view and sends it a *pull* request (line 8) to which the recipient, if correct, replies by sending the contents of its current view (line 11). Then, the node selects another peer from its view and sends it a *push* message containing its current view (line 9). On receiving the reply to its pull request, or a *push* message sent at line 9, a node greedily updates any slot view[i] that can be brought closer to its corresponding seed using one of the received identifiers (lines 20-23, where m at line 20 is the size of the sample passed to the function updateSample). The peer to which a push message was sent does the same on its side.

We implement $\text{rank}_{\text{seed}[i]}(\cdot)$ using a uniform hash function h $\text{rank}_{\text{seed}[i]}(p) = h(\langle \text{seed}[i], p \rangle)$ (where angle brackets represent a tuple). Using a hash function effectively implements a random permutation over node identifiers and realizes a uniform sampling of these identifiers.

Making the graph dynamic. To continuously generate fresh samples, nodes periodically return k identifiers from their view to the application and then reset the corresponding seeds to random values (lines 14-18), before updating the associated view entries view[r], to the identifiers from the current view that best match the new seeds (line 19). These

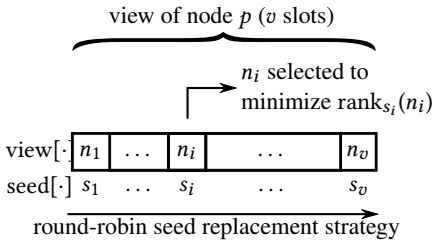


Figure 1. The neighbor selection mechanism of BASALT

Algorithm 1: The BASALT algorithm

```
1  algorithm parameters
2  |  see Table 1
3  initialization
4  |  for i ∈ 1,...,v do
5  |  |  seed[i] ←rand_seed(); view[i] ← ⊥
6  |  r ← 1; updateSample(bootstrap_peers)
7  every τ time units
8  |  p ←selectPeer(); Send ⟨PULL⟩ to p
9  |  q ←selectPeer(); Send ⟨PUSH, view[·]⟩ to q
10 on receive ⟨PULL⟩ from p
11 |  Send ⟨PUSH, view[·]⟩ to p
12 on receive ⟨PUSH, [p_1,...,p_v]⟩ from p
13 |  updateSample([p_1,...,p_v, p])
14 every k/ρ time units
15 |  repeat k times
16 |  |  r ← (r mod v) + 1
17 |  |  Sample view[r]
18 |  |  seed[r] ←rand_seed()
19 |  updateSample(view[·])
20 function updateSample([p_1,...,p_m])
21 |  for i ∈ 1,...,v and p ∈ [p_1,...,p_m] do
22 |  |  if view[i] = ⊥ or
   |  |    rank_seed[i](p) < rank_seed[i](view[i]) then
23 |  |  |  view[i] ← p;
24 function selectPeer()
25 |  i ← random value from [1..v]
26 |  return view[i]
```

k slots are selected in a round-robin fashion every k/ρ time units, yielding ρ random samples per time unit on average.

Parameter ρ controls the number of random samples per time unit, and so the number of slots whose seeds are refreshed at each time unit. With a view size of v, each slot is refreshed on average every v/ρ time units. The value of v/ρ must be large enough with respect to the exchange interval, τ, so that the view slots can converge before being refreshed. Parameter k controls, on the other hand, the number of slots that are reset simultaneously. A large value of k causes the

[1] See Sec. 3.3.1 for a discussion of the influence of this bootstrap.

Table 2. Theoretical model variables used in Section 3

t	Time	
$c(t)$	Number of correct node identifiers seen	$0 \leq c(t) \leq Q$
$b(t)$	Number of malicious node identifiers seen	$0 \leq b(t) \leq fn$
$B(t)$	Probability of sampling a Byzantine node	$= \frac{b(t)}{b(t)+c(t)}$

algorithm to explore many slots in parallel, thereby obtaining more diverse samples that help the k slots converge faster together. A small value of k (e.g. $k = 1$), instead, causes the exploration to occur with most slots in a quasi-converged state, limiting the population of candidates considered to optimize a node's view. We return to the impact of these parameters in our evaluation.

3 Theoretical Analysis

We now use a theoretical continuous model to estimate the value of $B(t)$, the probability at a time t that a given slot of a correct process contains a Byzantine peer identifier, as a function of f, the fraction of Byzantine nodes in the network (see Table 2 for an overview of the variables we will use for our analysis).

3.1 Parameters and Assumptions

Scenario parameters. In order to derive a continuous model of BASALT's behavior, we consider a theoretically perfectly uniform hashing function, i.e., a hash function that ensures that the hash values of nodes controlled by the attacker have the same variety as those of correct nodes. We note n the total number of network nodes (i.e., the network size). The product fn denotes the number of Byzantine nodes, and $Q = (1 - f)n$ denotes the number of correct nodes.

Assumptions. BASALT employs a simple hash function $\text{rank}_{\text{seed}[i]}(p) = h(\langle \text{seed}[i], p \rangle)$ to rank potential neighbors. To approximate the system's behavior, we will reason using the mean values of $c(t)$, the number of correct identifiers considered by a node for a slot, over all nodes and slots. We will assume that the values of individual nodes tend to concentrate around their means in practice with high probability, as is usually the case in such stochastic systems.

3.2 Deriving $B(t)$

Our theoretical analysis revolves around the probability $B(t)$ of selecting a Byzantine node in a given slot of a node p at time t, for which we use the following result.

Theorem 3.1. *The probability $B(t)$ of selecting a Byzantine node in a given slot of a node p at time t is equal to*

$$B(t) = \frac{b(t)}{b(t) + c(t)}, \tag{3}$$

where $c(t)$ is the number of correct identifiers seen at time t by p since the last reset of the slot, and $b(t)$ is the number of Byzantine peer identifiers seen by p over the same period.

Proof. The probability $B(t)$ of selecting a Byzantine node in a given slot of a node p at time t depends on two sets of identifiers: the set of correct identifiers seen at a time t by p on this slot since the last reset, and the set of Byzantine identifiers seen by p over the same period.

We define $C(t)$ as the set (of size $c(t)$) of correct node identifiers seen by a node p at time t since the last reset of a given slot. Similarly, $\mathcal{B}(t)$ is the set (of size $b(t)$) of Byzantine node identifiers seen by p over the same interval.

We fix one node p selected randomly amongst $C(t) \cup \mathcal{B}(t)$, and write $\text{selected}(p)$ the event that p is selected by the ranking function $\text{rank}_S(\cdot)$:

$$\text{selected}(p) \equiv \left(p = \text{argmin}_{q \in C \cup \mathcal{B}} \text{rank}_S(q)\right). \tag{4}$$

With this notation we have $B(t) = \Pr\left(p \in \mathcal{B} \mid \text{selected}(p) \right)$.

Byzantine and honest nodes are indistinguishable from the point of view of $\text{rank}_S(\cdot)$, which means here that the events $p \in C(t)$ and $\text{selected}(p)$ are independent. This independence implies that

$$\begin{aligned} B(t) &= \Pr\left(p \in \mathcal{B}(t) | \text{selected}(p)\right) \\ &= \frac{\Pr\left(p \in \mathcal{B}(t) \wedge \text{selected}(p)\right)}{\Pr\left(\text{selected}(p)\right)} \\ &= \frac{\Pr\left(p \in \mathcal{B}(t)\right) \times \Pr\left(\text{selected}(p)\right)}{\Pr\left(\text{selected}(p)\right)} \\ &= \Pr\left(p \in \mathcal{B}(t)\right) \\ &= \frac{|\mathcal{B}(t)|}{|C(t)|+|\mathcal{B}(t)|} = \frac{b(t)}{c(t) + b(t)}. \end{aligned} \tag{5}$$

□

In this analysis, we consider the worst-case scenario in which correct nodes have been flooded with all existing Byzantine identifiers, and we use the following result.

Corollary 3.2. *In a worst-case scenario in which correct nodes have been flooded with all existing Byzantine identifiers, the following holds*

$$B(t) = \frac{b_{\max}}{b_{\max} + c(t)}, \tag{6}$$

where $b_{\max} = fn$ is the total number of Byzantine identifiers.

Proof. This result directly follows from Theorem 3.1, and the fact that $f(x) = \frac{x}{a+x}$ is increasing over $]-a, +\infty[$. □

3.3 Analysis of the Core Mechanism

We first analyze the risk of a node becoming isolated (i.e., of an Eclipse attack succeeding) under this worst-case scenario before moving on to studying the convergence properties of BASALT assuming no node is ever isolated.

3.3.1 Bounding the Probability of Isolation. Isolation, which is synonym to a successful Eclipse Attack, is essential in our context for two reasons: first because, once isolated, an honest node is fully at the mercy of the Byzantine nodes that surround it (i.e., it has been *eclipsed*, hence the name of the attack); second, because as long as an honest node retains some connection to other honest nodes, BASALT ensures (thanks to its *stubbornness*, see Section 2.3) that it might recover a view with a balanced mix of honest and Byzantine identifiers (which we analyze formally just below when discussing non-isolated executions). Isolation occurs in two ways: when a node joins the network, or when it replaces all correct peers in its view with Byzantine peers.

Isolated joining node. We assume a worst-case scenario where the unfortunate joining node receives all Byzantine identifiers as soon as it joins. At time ϵ after joining, we thus have $b(\epsilon) = b_{\max}$ and $c(\epsilon) = (1 - f_0)I$, where f_0 is the fraction of Byzantine nodes in the bootstrap sample and I is the size of the bootstrap sample. Since we defined $B(t)$ as the probability of a given slot in the view being occupied by a Byzantine peer, we can write the probability that a node has only Byzantine neighbors as $B(t)^v$.

$$B(t)^v = \left(\frac{b_{\max}}{b_{\max} + c}\right)^v = \left(\frac{1}{1 + (1 - f_0)\,\frac{I}{fn}}\right)^v \tag{7}$$

We can reduce this probability exponentially by increasing v, by increasing I or by assuming a lower f_0. For instance, supposing $f_0 = 50\%$ of malicious nodes in our bootstrap peer list, by taking a view size of $v = 200$ and a bootstrap peer list size 25% of the number of malicious nodes in the network ($I = \frac{1}{4}fn$), $B(t)^v$ becomes smaller than 10^{-10}. Supposing, for instance, a network of size $n = 10000$ with a fraction $f = 0.1$ of Byzantine nodes, this only requires a bootstrap set of size $I = 250$ nodes, of which only 125 are required to be correct.

Convergence to isolated state. The second way for a node to become isolated results from resetting the seeds for the slots that still contain correct peers to new seeds that select Byzantine nodes. When such a reset occurs, the probability that all of the non-reset slots are already owned by Byzantine peers is equal to

$$B(t)^{v-k} = \left(\frac{b_{\max}}{b_{\max} + c(t)}\right)^{v-k}. \tag{8}$$

To bound this quantity, let us consider the value of $c(t)$ at the time of a reset, depending on the value of $c(t)$ at the time of the previous reset. Consider for this a single node of the network and make the hypothesis that other network nodes are well-converged. As we discuss in Sections 3.3.2 and 4, this implies that the fraction of Byzantine nodes in their views approaches f with appropriate algorithm parameters. We write c_0 the value of $c(t)$ at the previous reset.

This problem can be interpreted as the coupon collector's problem, in which correct peer identifiers play the role of coupons, and the reception of random correct identifiers (with replacement) that of trials. More precisely, assuming c_0 correct peer identifiers are already known amongst Q, we seek to learn Δc new **distinct** correct peer identifiers. Applying the standard analysis used to solve the coupon collector's problem [26], the expected number of uniformly distributed (non-distinct) correct peer identifiers that must be received is:

$$\frac{Q}{Q - c_0} + \frac{Q}{Q - c_0 - 1} + \cdots + \frac{Q}{Q - c_0 - \Delta c + 1} \tag{9}$$

The number of correct peer identifiers received between the two resets is at least the following expression:

$$\frac{k}{\rho}\frac{v}{\tau}\frac{c_0}{fn + c_0}(1 - f) \tag{10}$$

where $\frac{k}{\rho}$ is the duration of the considered time slice, v is the number of peer identifiers exchanged at each exchange step, τ is the time between two exchange steps, $\frac{c_0}{fn+c_0}$ is the probability that the exchange was conducted with a correct peer, and $(1 - f)$ is the probability that each of the peers of the returned view is correct.

We bound the value of (9) as follows:

$$(9) \leq \Delta c \frac{Q}{Q - c_0 - \Delta c} \tag{11}$$

Moreover, we have (9) $\geq$ (10). Thus:

$$\Delta c \frac{Q}{Q - c_0 - \Delta c} \geq \frac{k}{\rho}\frac{v}{\tau}\frac{c_0}{fn + c_0}(1 - f)$$

thus

$$\Delta c Q \tau \rho (fn + c_0) \geq kvc_0(Q - c_0 - \Delta c)(1 - f)$$

thus

$$\Delta c \geq \frac{kvc_0(1 - f)(Q - c_0)}{Q\tau\rho(fn + c_0) + kvc_0(1 - f)}. \tag{12}$$

Suppose, for instance, a network of $n = 10000$ nodes with a proportion $f = 0.1$ of malicious nodes, with algorithm parameters $v = 100$ and $k = 50$. In this system, taking $\tau = 1$ and $\rho = 1$, and supposing that the node we are considering has just joined the network and knows only of $c_0 = f_0\frac{1}{4}fn = 125$ correct node identifiers, we obtain that $\Delta c \geq 467$, i.e. $c(t)$ at the next reset is expected to be at least 592.

In Equation (8) $B(t)^{v-k}$ is smaller than 10^{-10} as soon as the number $c(t)$ of correct node identifiers seen is larger than 585. In other words, the node's probability of becoming isolated at the next reset is negligible.

3.3.2 Non-Isolated Execution. Now that we have shown that the probability of a node becoming isolated can be made arbitrarily low, we make the following assumption in the rest of the analysis.

Assumption 1. *No node is isolated, and all nodes have at least some correct neighbors.*

Deriving a continuous model. Let's note $C(t) = 1 - B(t)$ the probability of a given slot of a given node to contain the identifier of a correct peer.

When Assumption 1 holds, and in the worst-case scenario discussed above (Byzantine nodes have propagated all their identities to all correct nodes) $b(t)$ is constant equal to $b_{\max} = fn$, which allows us to write the evolution of $c(t)$ over time as a differential equation, as the sum of contributions resulting from the various parts of the system:

- *Pull exchange*: every τ rounds, a node pulls from one peer in its view, which replies by sending v node identifiers. With probability $B(t)$, the node contacts a Byzantine peer. In this case, it does not learn any new identifier (by our worst-case assumption). With probability $C(t)$, the node contacts a correct peer. In this case, each returned identifier will itself be correct with probability $C(t)$, and if correct, it will have a probability $\frac{c(t)}{(1-f)n}$ of being already known ($(1-f)n$ being the total number of correct nodes). Thus we can express the effect of pull exchanges on $c(t)$ as:
$$\frac{dc}{dt} = \frac{C(t)^2 v}{\tau}\left(1 - \frac{c(t)}{(1-f)n}\right).$$
- *Push exchange*: every τ rounds, a node pushes to a random node in its view. This push has a $C(t)$ probability of being sent to a correct node. In this case, we can apply the same reasoning as above and derive the same contribution to $\frac{dc}{dt}$.
- *Sampling and view renewal*: every ρ rounds, a node resets one of its v slots and forgets the identifiers collected for this slot. Let us write $c(t)$ as $c(t) = \frac{1}{v}\sum_{i=1}^{v} c_i(t)$, where $c_i(t)$ is the number of correct nodes taken into account by slot i. Then a single c_i is set to zero every ρ rounds. On average, this yields the following contribution to $\frac{dc}{dt}$:
$$\frac{dc}{dt} = -\rho\frac{c(t)}{v}.$$

By summing all three above contributions, we obtain our final differential equation:

$$\frac{dc}{dt} = \frac{2C(t)^2 v}{\tau}\left(1 - \frac{c(t)}{(1-f)n}\right) - \rho\frac{c(t)}{v}. \tag{13}$$

Solving the continuous model. To solve Eq. 13, we can express $\frac{dB}{dt}$ as $\frac{dB}{dt} = -\frac{b_{\max}}{(b_{\max}+c)^2}\frac{dc}{dt}$, and by substituting $\frac{dc}{dt}$ from Eq. 13, we obtain:

$$\frac{dB}{dt} = B(1-B)\left(\frac{\rho}{v} - \frac{2v(1-B)(B-f)}{\tau f(1-f)n}\right) \tag{14}$$

To study the constant regime of this system, we write $\frac{dB}{dt} = 0$ and exclude the solutions $B = 0$, which is not compatible with $b_{\max} = fn$, and $B = 1$, which corresponds to the case where Byzantine nodes take over the whole network. We also simplify by setting $\tau = 1$ as its role is symmetrical to that of ρ. We obtain after a few steps:

$$(1-B)(B-f) = \frac{\rho f(1-f)n}{2v^2}. \tag{15}$$

The equation exhibits two roots $B_1 < B_2$.

$$B_{1,2} = \frac{1}{2}\left(1 + f \mp \sqrt{(1-f)^2 - 2\frac{\rho f(1-f)n}{v^2}}\right) \tag{16}$$

When the quantity on the right-hand side of Eq. 15 approaches zero, B_1 approaches f from above, while B_2 approaches 1 from below. Since $\frac{dB}{dt} > 0$ for $B < B_1$ and $B > B_2$, while $\frac{dB}{dt} < 0$ for $B_1 < B < B_2$, B_1 corresponds to a stable equilibrium, while B_2 corresponds to an unstable one. So we focus our analysis on B_1.

With respect to B_1, the right-hand side of Eq. 15 represents the difference between the proportion of malicious peers in nodes' views, $B(t)$, and their overall proportion in the network, f. Ideally, we want to keep this quantity as small as possible, making B only slightly larger than f.

To this end, we observe that the term $\frac{\rho f(1-f)n}{2v^2}$ shrinks proportionally to the square of the view size, v^2. Thus, choosing a large enough view size allows the network to converge to a state where Byzantine nodes control only slightly more peers in the view than their overall proportion in the network. Moreover, in order to obtain the same stable state value of B, the view size v should grow proportionally to $\sqrt{\rho}$ and $\sqrt{n}$ (in both cases, all other parameters remaining constant).

4 Experimental Evaluation

We complement our theoretical analysis with Monte Carlo simulations that illustrate BASALT's dynamic behavior. This section focuses on simulating a permissioned system with a known fraction of malicious nodes. We show that BASALT consistently produces samples with fewer malicious peers than the state-of-the-art algorithms Brahms [11] and SPS [22] over a wide range of scenarios. We also show that BASALT converges faster on metrics quantifying the random connectivity of the graph generated by the algorithm, such as the clustering coefficient and mean path length. These metrics are relevant for information dissemination and may thus influence the convergence time of epidemic agreement algorithms.

4.1 Experimental Setting

We evaluate the tested algorithms by simulating a system with n nodes, of which a fraction f implement the following malicious behavior:

- A malicious node that receives a pull request returns a view composed of v nodes selected uniformly at random amongst the malicious nodes.
- Regularly, a malicious node sends a push request to randomly selected correct peers, containing similarly a view of v uniformly random malicious peers.

This represents a worst-case situation since attackers cannot influence the choices correct nodes make nor read their local memory (lines 16-18 of Alg. 1). They are therefore limited to maximizing in expectation their representation in the views of correct nodes, which the above strategy implements.

We define the *force* of the attack, F (distinct from the fraction of Byzantine nodes f), as the ratio between the number of push requests sent by a Byzantine node and the number of push requests sent by a correct node in a given time interval. For example, if a Byzantine node sends push requests at the same rate as correct nodes, a force of F corresponds to sending requests to F distinct correct nodes rather than to only one. Alternatively, the force of the attack can also model a situation in which Byzantine nodes send requests more often or where the network loses more messages (due to corruption or timeouts) from correct nodes than from Byzantine ones. (The extreme scenario of Sec. 3 in which correct nodes are flooded with all existing Byzantine identifiers corresponds to an arbitrarily large value of F.)

We do not simulate node churn but consider instead an extreme scenario in which all nodes have just joined the system—this can be seen as an ultimate churn event in which all nodes are replaced.

4.2 Parameter Values

We fix the exchange interval to $\tau = 1$ (1 simulation time step). In practical systems, this exchange interval is typically of the order of a few tens of seconds, up to a few minutes, with one minute being used in the Avalanche network for instance [4]. In the following, we therefore assume that 1 simulation time step represents 10 seconds, in order to balance network costs, reactivity to Byzantine attacks, and sample freshness. For our evaluation, we use a base scenario consisting of $n = 10000$ nodes, a proportion $f = 0.1$ of malicious nodes, a view size of $v = 160$, a sampling rate of $\rho = 1$, and an attack force of $F = 10$. Unless stated otherwise, we keep all other parameters fixed to their base value. We then vary either v, ρ, or F in isolation to analyze their impact on BASALT and its competitors. $F = 10$ means we assume malicious nodes use 10 times more network traffic than honest nodes. An attacker must typically trade off its ability to influence correct nodes rapidly (with a high F value) and the risk of being detected (e.g., as a result of DDoS countermeasures, although this aspect is out of the scope of this paper). As we will see in Figure 2b, F only has a marginal effect on the behavior of BASALT. $\rho = 1$ means we assume the peer sampling service is requested by default with the same frequency as exchanges take place. Increasing the ratio ρ/τ provides more peer samples for the same network costs but introduces a vulnerability as nodes have less time to update their views between two samples. All algorithms were implemented in the same simulation framework written in Rust, totaling about 2500 lines of code.[2]

[2]https://github.com/basalt-rps/basalt-sim.

4.3 Competitors

We compare BASALT to two state-of-the-art competitors, Brahms [11] and SPS [22], which we presented in Section 2.

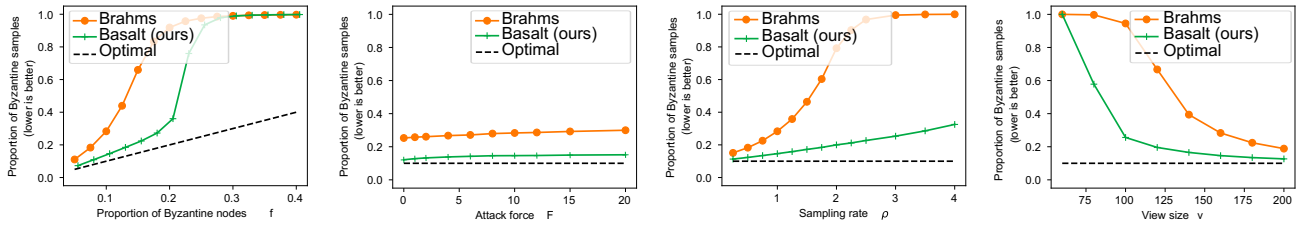
SPS was unable to function at all in the scenarios we tested: for instance, for $n = 1000$, $f = 30\%$, and even with a favorable attack force F of 0, 90% of correct nodes become isolated in the network rapidly using SPS and remain so during the whole simulation. In contrast, both BASALT and Brahms were able to prevent all correct nodes from becoming isolated in this scenario. In light of these results, we have excluded SPS from our comparison charts and concentrate on the comparison of BASALT against Brahms.

In all experiments, we set the size of Brahms' view ($\mathcal{V}_p$) and of its sampler vector ($\mathcal{S}_p$) to be the same, as in the experiments of the initial Brahms paper [11]. We further set this size to be that of BASALT's view, i.e. using the notation of Table 1, we set $\ell = v$, with $\ell = |\mathcal{V}_p| = |\mathcal{S}_p|$ and $v = |\text{view}|$, where $\mathcal{V}_p$ and $\mathcal{S}_p$ denote Brahms' view and sampler vector, and view is BASALT's view. Doing so allows Brahms to store twice as many node identifiers as BASALT. Given the view sizes we consider (of at most 200 identifiers), this overhead is, however, negligible and will not be discussed further.

A key advantage of BASALT is the tight feedback loop it introduces between the gossiping exchanges and the construction of samples using min-wise independent permutations. To compare Brahms and BASALT on similar grounds, we therefore opt for a balanced contribution of the push, pull, and sampling mechanism in Brahms and choose $\alpha = \beta = \gamma = 1/3$ (see Equation (2) in Section 2.2).

Whereas BASALT is multi-shot by design in that it produces a continuous stream of random node identifiers, the original Brahms algorithm is one-shot, in the sense that each node only constructs one single sample of peer ids. To be able to compare both approaches, we therefore make Brahms multi-shot and add to it a mechanism that resets some of the hash functions regularly, using the same round-robin strategy as BASALT. More concretely, we add a task to Brahms that is identical to that of lines 14-18 in Algorithm 1, in which line 18 is replaced by the corresponding resetting operation for Brahms ($\mathcal{S}_p[i]$.init() using the notation of the original Brahms paper [11]). Without such a mechanism, Brahms would always return the same fixed set of samples, limiting its usability as a random peer sampling algorithm, and preventing a direct comparison to BASALT.

For similar reasons, we deactivate the blocking mechanism that causes Brahms to stall when receiving too many push replies. This blocking mechanism protects Brahms under the assumption that Byzantine nodes can only send a limited number of push messages per time unit. By varying the attack force F we push Brahms beyond its design envelope in order to compare it against BASALT. In this context, keeping



(a) Varying f (malicious nodes) **(b)** Varying the attack force F **(c)** Varying the sampling rate ρ **(d)** Varying the view size v

Figure 2. Our algorithm (crosses, green) consistently provides samples that contain fewer Byzantine nodes than our competitor, Brahms, in a variety of situations. Results are shown for $n = 10000$ nodes. Each graph varies one parameter at a time (indicated in the legend), while the other parameters are set to their base value. Base values: proportion $f = 0.1$ of malicious nodes, view size of $v = 160$, sampling rate of $\rho = 1$, and attack force of $F = 10$.

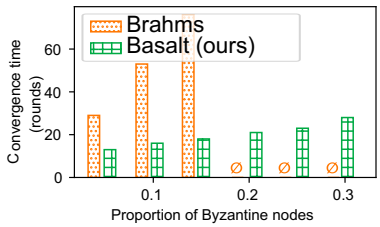


Figure 3. Time to convergence within 25% of optimal proportion of Byzantine samples, for $n = 1000$, $v = 100$ (on the right part, Brahms does not converge within experiment time)

the blocking mechanism would stall Brahms entirely and prevent it from returning a continuous stream of samples, thus precluding any comparison.

In terms of communication costs, we adopt a system perspective and allow Brahms and BASALT to perform two datagram exchanges per round on each node. We assume each datagram fits into a single network packet, corresponding to a maximum transmission unit (MTU) of 1500 bytes on the Internet. Supposing $\ell = v = 200$ (the maximum view size in our experiments) and node identifiers of size 4 bytes this communication budget allows nodes executing Brahms and BASALT to perform one push and one pull exchange in each round. Although all exchanges have the same network cost measured in packets, note that Brahms makes the specific design choice to limit pushed IDs to a peer's own ID: this is because pushing more IDs could spread more malicious IDs from the pulled and pushed sections of the view. By contrast, BASALT sends its full view of size v. During a pull exchange, both algorithms send back their full view ($\mathcal{V}_p$ for Brahms, and view for BASALT).

4.4 Proportion of Byzantine Samples

In our first experiment, we measure the average number of Byzantine nodes present in correct nodes' samples after 200 simulation time steps. For this experiment, we simulate a network of $n = 10000$ nodes. We fix base parameter values of $f = 10\%$ of malicious nodes, a sampling rate of $\rho = 1$, a view size of $v = 160$ and an attack force of $F = 10$. We then individually vary the parameters f, ρ, v, and F. Figure 2 shows how this proportion evolves for the two evaluated algorithms, as one of the parameters f, ρ, v, and F varies.

These results indicate that BASALT provides close to optimal proportions of Byzantine samples up to 20% of Byzantine nodes, whereas Brahms fails to contain the attack in this domain (Fig. 2a). However, above 20% of Byzantine nodes, although BASALT performs better than Brahms, it progressively fails to resist to the Byzantine attack. Further, BASALT is almost insensitive to F, whereas Brahms shows an increasing proportion of Byzantine samples when F increases (Fig. 2b). For low values of ρ, both Brahms and BASALT are able to converge to high-quality samples (Fig. 2c), however, such a setting does not provide much utility as the algorithm is unable to frequently return new samples to the application. Increasing the sampling rate ρ results, however, in more disruption, as view slots have a higher risk of being reset before they converge to their target peer. This disruption causes Brahms to collapse for higher values of ρ: the network becomes fully disconnected, and the views of correct nodes end up completely polluted by malicious peers.

Figure 2d shows how the algorithms behave for various view sizes. For small view sizes, all algorithms are unable to keep the network in a connected state, and correct nodes all end up isolated. The plots show that BASALT can keep the network connected using smaller views than Brahms.

4.5 Evaluating Convergence Speed

In this second experiment, we study the speed at which the algorithms converge to good network states, where they provide samples with low proportions of malicious nodes. Figure 3 shows the time that Brahms and BASALT take to converge to proportions of Byzantine samples that are within 25% of the optimal proportion, for $n = 1000, v = 100, F = 10$ and $\rho = 1$ and for varying proportions of Byzantine nodes in the network. We show that the convergence time of BASALT remains

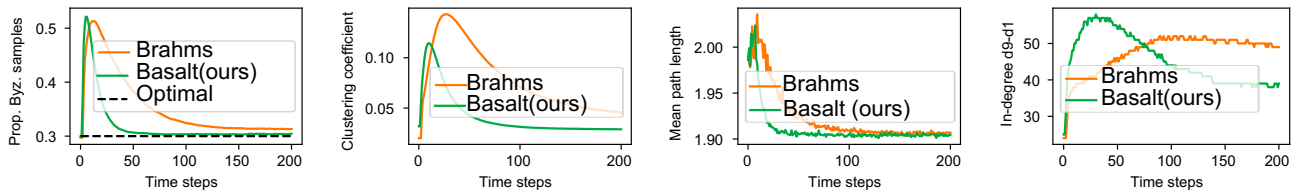


Figure 4. Algorithm convergence on several graph quality metrics, for $n = 10000$, $f = 10\%$, $F = 1$, $\rho = 0.5$, $v = 160$. On all metrics, lower is better: we see that BASALT converges much more rapidly than Brahms.

low for up to 30% of Byzantine nodes, whereas Brahms takes much longer to converge (starting at 20% of Byzantine nodes, it did not converge within the experiment's time).

Figure 4 shows the evolution of several metrics through time, starting with the number of Byzantine nodes in the view, in our experiment for $n = 10000$, in a favorable situation with $f = 10\%$, $\rho = 0.5$ and $F = 1$. The left-most plot shows that BASALT converges much faster than Brahms to a good network state. The remaining plots show metrics for graph quality, where the algorithms exhibit a similar convergence behavior: clustering coefficient, mean path length, and the concentration of in-degrees measured by the difference between the last and the first decile. The clustering coefficient is computed by averaging the local clustering coefficient of correct nodes in a graph where malicious nodes are assumed to be all connected to one another. The mean path length is measured in a graph where malicious nodes are assumed to have no connection in either direction, which models the situation where they do not cooperate in transmitting information between correct nodes.

4.6 Node Isolation vs. Sampling Rate

We have seen earlier (Fig. 2c) that both Brahms and BASALT are sensitive to increased sampling rates and return more malicious samples when the sampling rate, ρ, is high, with Brahms failing completely for too large values of ρ.

To investigate this effect further, we run both algorithms for various values of v and ρ, and plot the maximum value of ρ that can be used for a given v without causing a network partition. More precisely, a run for a given set of parameters v, ρ is successful if, starting from half of the allocated simulation time, no correct node is ever isolated by the malicious peers. Otherwise, it fails. We plot the successful runs with the highest values of ρ for a given v. The results of this experiment are shown in Figure 5 for $N = 10000$, $f = 10\%$ and $F = 10$. The areas delineated in Figure 5 correspond to the parameter sets that give successful runs. Our results show that for similar view sizes, BASALT achieves a higher sampling rate than Brahms, thus providing more utility to the application.

5 Live Deployment

To demonstrate the applicability of BASALT to a running system, we have applied BASALT to the AvalancheGo engine [3],

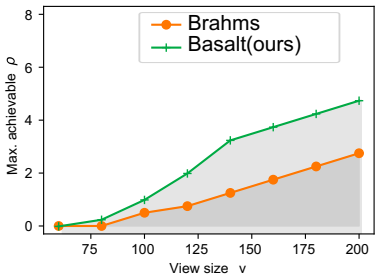


Figure 5. Maximum achievable sampling rate ρ (see Sec. 4.6) for 10000 nodes, $f = 10\%$

the main implementation of the AVA network [1] which uses the Avalanche consensus algorithm[3]. We picked AVA, as it is the main cryptocurrency network that uses an epidemic, sampling-based consensus, which is the target use case of BASALT. Our implementation, a 500-line patch to the Go source code of AvalancheGo, replaces peer sampling based on proof-of-stake by a peer sampling mechanism derived from BASALT.

Our implementation is fully compatible with the existing AVA network. To show that BASALT can help reduce the risk of an attack, we ran a 10-hour experiment where we launched 100 "adversarial" Avalanche nodes on the public AVA network (corresponding to about 20% of all active nodes) in an attempt to bias sampling in their favor using an Eclipse attack against one of our nodes. (This attack was transparent to the other AVA nodes and did not disrupt the functioning of the network in any way.) Samples were measured at witness nodes running the sampling mechanism derived from BASALT, and a sampling algorithm using a full knowledge of the network. The approach based on BASALT was able to deliver samples containing 17.5% of malicious nodes on average, a proportion comparable to that of a full-knowledge protocol (18.4%) and very close to the true proportion of Byzantine nodes (18.8%).

[3]Our code is publicly available at https://github.com/basalt-rps/avalanchego-basalt. Our implementation is forked from the official AvalancheGo repository [30]. Our changes are identified by "Basalt RPS Authors".

6 Discussion and Related Work

Random peer sampling. Random peer sampling in non-adversarial settings is a well-studied problem [21, 33, 35]. Similarly, defenses against Eclipse attacks in structured peer-to-peer overlays, such as DHTs, have been extensively researched [24, 31, 32]. By contrast, and somewhat surprisingly, very few works have sought to develop Byzantine-tolerant RPS protocols.

As discussed in Section 2.2, Brahms [11] and SPS [22] are two existing RPS algorithms designed to withstand Byzantine attacks. Both algorithms are based on a classical RPS strategy that is extended to correct for the over-representation of malicious nodes. SPS takes inspiration from social network analysis and builds some statistical knowledge on node behavior. In particular, nodes that exhibit extreme indegree values are suspected and blacklisted. Unfortunately, this detection mechanism necessitates some warming period, and as a result, this mechanism is unable to cope with attacks where malicious nodes send so many messages that correct nodes do not have the time to gather sufficient statistics to block them before becoming isolated. Brahms maintains two views: a traditional gossip view ($\mathcal{V}_p$) on one hand, and a vector of "samplers" ($\mathcal{S}_p$) on the other hand, in which each sampler implements a min-wise independent permutation. This design departs markedly from that of BASALT whose view consists only of the sampler's view. In each gossip round, Brahms' gossip view is updated using the result of push/pull exchanges, and by reusing some of the IDs from the sampler vector. The relative contributions of each mechanism (pull, push, and samplers) are controlled by three parameters (α, β, γ). The identifiers that pass through the gossip view are then fed into the sampler vectors to construct the final random sample using a stubborn chaotic search similar to that of BASALT. In contrast to BASALT, however, Brahms assumes that Byzantine nodes are limited in their sending rate (in practice, this rate is supposed to be similar to that of honest nodes, corresponding to an attack force of $F = 1$), and implements a blocking mechanism when this rate is exceeded. Brahms also makes the specific design choice to limit pushed IDs to a peer's own ID: this is because pushing more IDs could spread more malicious IDs from the pulled and pushed sections of the view. BASALT's view, on the other hand, is entirely managed by the sampler.

In a recent contribution, Pigaglio et al proposed RAPTEE [29] an improvement over Brahms that leverages the presence of devices that support Intel SGX [15], a Trusted Execution Environment, present on some Intel CPUs. RAPTEE allows SGX nodes to run a standard peer-sampling protocol, while using Brahms only to interface with non-SGX nodes. This strategy provides increased resilience to Byzantine samples at the cost of slightly slower convergence. As their approach appears completely orthogonal to ours, it would be interesting to explore whether applying a RAPTEE-like technique to BASALT would result in similar improvements.

More generally, the inherent limitations of statistical filtering techniques were characterized formally in [6], in terms of bounds on what we have termed the *attack force* of malicious nodes, and on the local memory available to honest nodes. Following upon this analysis, the approach of [7] used Count-Min Sketches [14] to approximate an ideal RPS from a biased stream of identifiers.

Our protocol circumvents these limitations by eschewing filtering altogether. Instead, BASALT directly exploits a minimization of hash functions, which makes it capable of effectively handling the above Byzantine flooding attacks.

Verifiable Random Functions. The committee creation mechanism used in Algorand [17], although related to peer sampling, is not directly applicable as an RPS service. Algorand's committee creation uses Verifiable Random Functions (VRF) [25] to determine which nodes should participate in a round's consensus, with a probability that is proportional to each user's stake. When successfully executed by a node p, the VRF algorithm of Algorand provides p with a proof that p is entitled to participate in the round's committee. The VRF algorithm, however, only provides each selected node with a proof of its selection but does not prescribe how this proof should be distributed to the rest of the system. Algorand uses a gossip network (similar to that of Bitcoin) to propagate this information, thus implicitly assuming a mechanism providing random peer sampling. The prototype presented in the original Algorand paper ([17], Section 9) implements this peer sampling through a global and static address book, known to all participants. This address book links the IP addresses of participants with their public key. The Algortand authors indicate in the same section that such an approach to gossiping is susceptible to Sybil attacks (discussed just below) and leave the design of a Sybil-resistant gossip network to future work.

Sybil Attacks. Under working parameters chosen appropriately using Equation (16), BASALT guarantees that the proportion of Byzantine identifiers returned in each sample is close to that of the proportion of Byzantine identifiers present in the whole system. This property is essential to defend against Eclipse Attacks. However, on its own, it does not protect against an attacker capable of generating arbitrary large numbers of identifiers, a situation known as *a Sybil attack*. Mitigating Sybil attacks in our setting would require limiting the ability of an attacker to obtain new identifiers, for instance, by only accepting identifiers that are tied to some limited pool of resources as formalized in [10] (which is essentially what Proof-of-Work and Proof-of-Stake do). Such a mechanism, which limits an attacker's ability to control a large portion of the "voting rights" in a system, is orthogonal to BASALT, which focuses primarily on *Eclipse Attacks* and is out of the scope of this work.

A possible avenue to add Sybil resilience of BASALT could be to use the approach proposed in HAPS [12]. HAPS is a RPS protocol that addresses Sybil attacks in which attackers are concentrated in a few IP blocks (sometimes termed "institutional attacks") by using random walks on a carefully crafted probabilistic tree. Due to its design, however, it is not immediately clear how HAPS could be extended to counter attackers that are spread out, which BASALT does thanks to its stubborn chaotic search.

Network Attacks. Recent works on blockchains have also brought to light the risk of attacks at a more fundamental level than those we consider in this paper. Network adversaries are malicious entities that gain control of part of the routing infrastructure (Internet autonomous systems, or ASes), in which case they can intercept and modify all the traffic that they are routing, or attack the routing algorithm itself by advertising Internet prefixes that they do not own, thus attracting traffic that should have gone through another path, a so-called *BGP hijack* [9].

Most ISPs implement BGP filtering, and many organizations closely monitor BGP announcements for changes and potential hijacks to counter them as soon as possible. As a result, BGP hijacking attacks are necessarily limited to one or a few IP prefixes, and even then, they cannot extend for long periods. How to counter such attacks within an RPS protocol remains an open question. One venue could consist in spreading connections over a variety of IP prefixes by using a specially crafted rank function.

However, network attacks might also be used to target specific nodes, to remove them from the global network and make them believe false information about the network's state, resulting in a network-level Eclipse attack. Defenses have been proposed against Eclipse attacks at the network level: for instance, the SABRE network [8] proposes to use additional communication channels, in the form of a network of specialized nodes that are all connected to one another using dedicated links, and that are located close to end users so that they can provide a safe service directly to them, even in the case of a hijack.

For sampling-based methods that use BASALT, SABRE could provide a security mechanism that detects network attacks and stops all activity in case they happen, for instance by detecting a discrepancy between a node's local state and the state of SABRE nodes. This mechanism, however, cannot be used to allow eclipsed nodes to make progress in such a situation as it does not provide the secure random peer sampling service itself. Finding mechanisms to allow nodes that are eclipsed by a network attack to continue functioning normally when running a sampling-based algorithm is, to the best of our knowledge, still an open problem.

7 Conclusion

We have presented a new algorithm for Byzantine-tolerant random peer sampling for very large networks that uses stubborn chaotic search to limit the disruption power of Byzantine nodes. Such an algorithm can be used to implement sampling-based consensus algorithms such as Avalanche. Contrary to sampling algorithms based on Proof-of-Stake, BASALT allows any user to join the consensus without having to own any cryptocurrency tokens. We expect that in the future the line of research around Byzantine fault-tolerant algorithms based on epidemics will continue to see new developments motivated by gains in performance, and thus, we believe that the kind of RPS mechanism that BASALT provides is likely to rise in importance as these systems gain prominence.

Acknowledgments

This work was partially supported by Rennes Métropole (Program "*Allocation d'installation scientifique*", AIS, France), along with the French ANR projects ByBloS (ANR-20-CE25-0002-01) and PriCLeSS (ANR-10-LABX-07-81) devoted to the modular design of building blocks for large-scale Byzantine-tolerant multi-users applications. The authors would also like to thank the anonymous reviewers whose careful reading and suggestions helped them improve their paper.

References

[1] 2020. AVA Labs, Build the Internet of Finance. https://www.avalabs.org/. Accessed: 2020-07-21.

[2] 2020. Ethereum.org. https://www.ethereum.org/. Accessed: 2020-02-20.

[3] 2020. Gecko, Official Go implementation of an AVA node. https://github.com/ava-labs/avalanchego. Accessed: 2020-07-21.

[4] 2022. Network Protocol. https://docs.avax.network/specs/network-protocol Accessed February 2023.

[5] André Allavena, Alan Demers, and John E. Hopcroft. 2005. Correctness of a Gossip Based Membership Protocol. In *ACM Symposium on Principles of distributed computing (PODC)*. 292–301.

[6] Emmanuelle Anceaume, Yann Busnel, and Sébastien Gambs. 2013. On the Power of the Adversary to Solve the Node Sampling Problem. *Trans. Large Scale Data Knowl. Centered Syst.* 11 (2013), 102–126.

[7] Emmanuelle Anceaume, Yann Busnel, and Bruno Sericola. 2013. Uniform node sampling service robust against collusions of malicious nodes. In *IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*. 1–12.

[8] Maria. Apostolaki, Marti Gian, Müller Jan, and Vanbever Laurent. 2019. SABRE: Protecting Bitcoin against Routing Attacks.. In *Network and Distributed System Security Symposium (NDSS)*. 1–15.

[9] Maria Apostolaki, Aviv Zohar, and Laurent Vanbever. 2017. Hijacking Bitcoin: Routing Attacks on Cryptocurrencies. In *IEEE Symposium on Security and Privacy (S&P)*.

[10] Sarah Azouvi, Christian Cachin, Duc V Le, Marko Vukolic, and Luca Zanolini. 2022. Modeling Resources in Permissionless Longest-chain Total-order Broadcast. In *Conference on Principles of Distributed Systems (OPODIS)*.

[11] Edward Bortnikov, Maxim Gurevich, Idit Keidar, Gabriel Kliot, and Alexander Shraer. 2009. Brahms: Byzantine resilient random membership sampling. *Computer Networks* 53, 13 (2009), 2340–2359.

[12] Amaury Bouchra Pilet, Davide Frey, and Francois Taiani. 2020. Foiling Sybils with HAPS in Permissionless Systems: An Address-based Peer Sampling Service. In *IEEE Symposium on Computers and Communications (ISCC)*.

[13] Andrei Z Broder, Moses Charikar, Alan M Frieze, and Michael Mitzenmacher. 1998. Min-wise independent permutations. In *ACM Symposium on Theory of Computing (STOC)*. 327–336.

[14] Graham Cormode and Shan Muthukrishnan. 2005. An improved data stream summary: the count-min sketch and its applications. *Journal of Algorithms* 55, 1 (2005), 58–75.

[15] Victor Costan and Srinivas Devadas. 2016. Intel SGX Explained. Cryptology ePrint Archive, Paper 2016/086. https://eprint.iacr.org/2016/086 https://eprint.iacr.org/2016/086.

[16] Alan Demers, Dan Greene, Carl Hauser, Wes Irish, John Larson, Scott Shenker, Howard Sturgis, Dan Swinehart, and Doug Terry. 1987. Epidemic algorithms for replicated database maintenance. In *ACM Symposium on Principles of distributed computing (PODC)*. 1–12.

[17] Yossi Gilad, Rotem Hemo, Silvio Micali, Georgios Vlachos, and Nickolai Zeldovich. 2017. Algorand: Scaling byzantine agreements for cryptocurrencies. In *ACM Symposium on Operating Systems Principles (SOSP)*. 51–68.

[18] Rachid Guerraoui, Petr Kuznetsov, Matteo Monti, Matej Pavlovič, and Dragos-Adrian Seredinschi. 2019. The consensus number of a cryptocurrency. In *ACM Symposium on Principles of Distributed Computing (PODC)*. 307–316.

[19] Rachid Guerraoui, Petr Kuznetsov, Matteo Monti, Matej Pavlovic, and Dragos-Adrian Seredinschi. 2019. Scalable Byzantine reliable broadcast. In *International Symposium on Distributed Computing (DISC)*.

[20] Ethan Heilman, Alison Kendler, Aviv Zohar, and Sharon Goldberg. 2015. Eclipse attacks on bitcoin's peer-to-peer network. In *USENIX Security Symposium (USENIX Security)*. 129–144.

[21] Márk Jelasity, Spyros Voulgaris, Rachid Guerraoui, Anne-Marie Kermarrec, and Maarten van Steen. 2007. Gossip-based Peer Sampling. *ACM Trans. Comput. Syst.*, Article 8 (2007).

[22] Gian Paolo Jesi, Alberto Montresor, and Maarten van Steen. 2010. Secure peer sampling. *Computer Networks* 54, 12 (2010), 2086–2098.

[23] A. Kermarrec, L. Massoulie, and A. J. Ganesh. 2003. Probabilistic reliable dissemination in large-scale systems. *IEEE Transactions on Parallel and Distributed Systems* 14, 3 (March 2003), 248–258.

[24] Jon McLachlan, Andrew Tran, Nicholas Hopper, and Yongdae Kim. 2009. Scalable onion routing with torsk. In *ACM conference on Computer and communications security (CCS)*. 590–599.

[25] Silvio Micali, Michael Rabin, and Salil Vadhan. 1999. Verifiable random functions. In *Annual Symposium on Foundations of Computer Science (FOCS)*. IEEE, 120–130.

[26] Michael Mitzenmacher and Eli Upfal. 2005. *Probability and Computing: Randomized Algorithms and Probabilistic Analysis*. Cambridge University Press.

[27] Satoshi Nakamoto. 2009. Bitcoin: A peer-to-peer electronic cash system. https://assets.pubpub.org/d8wct41f/31611263538139.pdf accessed February 2023.

[28] Brice Nédelec, Julian Tanke, Davide Frey, Pascal Molli, and Achour Mostéfaoui. 2018. An adaptive peer-sampling protocol for building networks of browsers. *World Wide Web* 21, 3 (May 2018), 629–661.

[29] M. Pigaglio, J. Bruneau-Queyreix, Y. Bromberg, D. Frey, E. Riviere, and L. Reveillere. 2022. RAPTEE: Leveraging trusted execution environments for Byzantine-tolerant peer sampling services. In *IEEE 42nd International Conference on Distributed Computing Systems (ICDCS)*. IEEE Computer Society, Los Alamitos, CA, USA, 603–613.

[30] Team Rocket, Maofan Yin, Kevin Sekniqi, Robbert van Renesse, and Emin Gün Sirer. 2019. Scalable and Probabilistic Leaderless BFT Consensus through Metastability. https://doi.org/10.48550/ARXIV.1906.08936

[31] Atul Singh, Tsuen-Wan Ngan, Peter Druschel, and Dan S. Wallach. 2006. Eclipse Attacks on Overlay Networks: Threats and Defenses. In *IEEE Internet Conference on Computer Communications (INFOCOM)*.

[32] Guido Urdaneta, Guillaume Pierre, and Maarten Van Steen. 2011. A Survey of DHT Security Techniques. *ACM Comput. Surv.* 43, 2, Article 8 (Feb. 2011), 49 pages.

[33] Spyros Voulgaris, Daniela Gavidia, and Maarten Van Steen. 2005. Cyclon: Inexpensive membership management for unstructured p2p overlays. *Journal of Network and systems Management* 13, 2 (2005), 197–217.

[34] Spyros Voulgaris and Maarten van Steen. 2013. VICINITY: A Pinch of Randomness Brings out the Structure. In *International Middleware Conference (Middleware)*, Vol. 8275. Springer, 21–40.

[35] Spyros Voulgaris and Maarten van Steen. 2013. VICINITY: A Pinch of Randomness Brings out the Structure. In *International Conference on Middleware (Middleware)*. 21–40.

[36] Fan Zhang, Ittay Eyal, Robert Escriva, Ari Juels, and Robbert Van Renesse. 2017. REM: Resource-efficient mining for blockchains. In *USENIX Security Symposium (USENIX Security)*. 1427–1444.