BASALT: A Rock-Solid Foundation for Epidemic Consensus Algorithms in Very Large, Very Open Networks

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

▶ **To cite this version:**

BASALT: A Rock-Solid Foundation for Epidemic Consensus Algorithms in Very Large, Very Open Networks

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

Univ Rennes, Inria, CNRS, IRISA, Rennes, France

{alex.auvolat,davide.frey}@inria.fr,{david.bromberg,francois.taiani}@irisa.fr

ABSTRACT

Recent works have proposed new Byzantine consensus algorithms for blockchains based on epidemics, a design which enables highly-scalable performance at a low cost. These methods however critically depend on a secure random peer sampling service: a service that provides a stream of random network nodes where no attacking entity can become over-represented. To ensure this security property, current epidemic platforms use a Proof-of-Stake system to select peer samples. However such a system limits the openness of the system as only nodes with significant stake can participate in the consensus, leading to an oligopoly situation. Moreover, this design introduces a complex interdependency between the consensus algorithm and the cryptocurrency built upon it.

In this paper, we propose a radically different security design for the peer sampling service, based on the distribution of IP addresses to prevent Sybil attacks. We propose a new algorithm, BASALT, that implements our design using a stubborn chaotic search to counter attackers' attempts at becoming over-represented. We show in theory and using Monte Carlo simulations that BASALT provides samples which are extremely close to the optimal distribution even in adversarial scenarios such as tentative Eclipse attacks. Live experiments on a production cryptocurrency platform confirm that the samples obtained using BASALT are equitably distributed amongst nodes, allowing for a system which is both open and where no single entity can gain excessive power. **Keywords**: Gossip, Peer Sampling, Distributed System, Byzantine tolerance, Consensus

1 INTRODUCTION

Blockchain-based systems, such as cryptocurrencies [19] and smart contract platforms [3], are said to be *Byzantine Fault Tolerant* (BFT for short), i.e. they are able to resist to attacks from malicious participants (called *Byzantine* nodes), making it arbitrarily hard for instance for an attacker to forge false transactions or revoke already committed transactions. In particular, decision power over the blockchain's state must be spread over various network participants in order to prevent an attacker from obtaining full control over the system.

The breakthrough made by Bitcoin [19] allowed for Byzantine fault-tolerance to be achieved in a truly open network, using a Proof-of-Work system that requires participants to solve computationally intensive crypto-puzzles. The difficulty of these crypto-puzzles limits the influence of individual nodes, but encourage a race for computing power, with Bitcoin reported to consume as much electricity as Austria in 2020 [2]. Moreover, the throughput and latency of Proof-of-Work (PoW) systems are restricted by the time between blocks, which must be long enough to ensure security.

Epidemic BFT algorithms. A particularly interesting area of research in alleviating these issues with Proof-of-Work consists in a new family of BFT algorithms [12, 13, 21] that exploits epidemic mechanisms to provide large-scale protection against Byzantine behaviour. Epidemic algorithms allow for extremely fast dissemination of information in very large networks by means of stochastic peer-to-peer exchanges [9, 17]. Epidemic BFT algorithms exploit this property by repeatedly sampling small sets of random peers in the network, which they then use to estimate the overall system's state, and ensure coordination and agreement between correct (i.e. non-Byzantine) nodes.

Epidemic BFT approaches critically depend on the availability of *good* network samples, in the sense that the proportion of Byzantine nodes in a sample should be kept as low as possible, and sampled nodes should be as varied 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 [21].

Secure random peer sampling. Unfortunately, classical RPS algorithms [15, 20, 23] are not resilient to malicious behavior: Byzantine nodes can easily disrupt their execution by flooding honest nodes with Byzantine identifiers. Left unchecked, this strategy has the potential to isolate honest nodes in a so-called *Eclipse attack* [14, 22], or to partition the system. Moreover, a scheme where peers are sampled with uniform probability is vulnerable to so-called *Sybil attacks* [10] where a malicious entity creates arbitrarily many network node identifiers that it controls, thus gaining unlimited influence on the network.

Current deployments of epidemic BFT algorithms, such as the AVA cryptocurrency platform [1], rely on a Proof-of-Stake mechanism 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. However, Proof-of-Stake has several known limitations [25]. In essence, Proof-of-Stake consists in building an abstraction of a closed (permissioned) system, where system membership can however evolve dynamically according to the various parties' economic investments (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 directly in a much more open fashion, without resorting to Proof-of-Stake to ensure security.

Content of this paper. 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 exhibits close to optimal Byzantine fault tolerance, thus significantly improving on the state-of-the-art [8, 16]. BASALT is designed to operate in Internet-scale permissionless systems while

resisting to Eclipse and Sybil attacks. At the core of BASALT lies what we have termed a *stubborn chaotic search*, a greedy epidemic procedure [24] towards random nodes that are implicitly defined in a way that makes it extremely hard for malicious nodes to manipulate the decisions of correct ones. This procedure is parametrized by a target distribution on nodes based on their IP addresses, which we define to defend against Sybil attacks by institutions that own large contiguous portions of the IP address space.

We comprehensively analyze BASALT under a theoretical model based on the *power f* of the attack, which captures the (ideal) probability of sampling malicious nodes as defined by the target distribution. We show that BASALT provides samples in which the proportion of malicious nodes is very close to f, its theoretical optimum, and that f is acceptably small in several real-world scenarios including institutional attacks and botnet attacks. We complement our theoretical model with Monte Carlo simulations that confirm our analysis. Finally, we demonstrate the feasibility and concrete benefits of our technique by deploying BASALT within a live cryptocurrency network using a prototype implementation of BASALT for AvalancheGo [4], the reference engine powering the AVA cryptocurrency network [1, 21]. Our experiments on the AVA network confirm that the samples obtained using BASALT are equitably distributed amongst nodes, allowing for a system which is both open and where no single entity can gain excessive power. Our prototype is publicly available, fully functional, and compatible with the existing AVA network without requiring any protocol changes.

2 PROBLEM STATEMENT

A random peer sampling (RPS) service can be defined as a service that produces a continuous stream $(p_i)_{i \geq 0}$ of random nodes selected in the network. As stated above, 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 in $(p_i)_{i \geq 0}$.

2.1 System Model

We assume a very large system composed of 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 in order 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 write Q the number of correct nodes in the system.

We consider a communication network where any node can send a message to any other node, and assume that more than a fixed 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 block completely the communication between two correct nodes, or read in the local memory of correct nodes.

Nodes are granted each a unique identifier, which we assume to be their IP address. We will use the same notation to refer to a node and to its identifier. We assume that Byzantine nodes may not spoof the IP addresses of other nodes, which can be prevented using a handshaking mechanism [11].

2.2 Sybil Attacks

Random peer sampling is often considered under the assumption of a *closed*, or *permissioned* system (e.g. [8, 15]), where the whole set of nodes is known and the proportion of malicious nodes is equal to (or bounded by) a small fixed fraction φ. In such a situation, a perfect random peer sampler could be defined as one that samples all nodes uniformly, thus returning a fraction φ of malicious nodes in the samples it produces.

This assumption is however not adapted to an open network such as the public Internet, which is more akin to a *permissionless* (open) system. In such a setting, an attacker may control nodes with many times more IP addresses than there are correct nodes, which may then be used to perform a Sybil attack, leading to an increased influence of the attacker in the peer sample's output. In particular, a RPS that samples peers uniformly based on their IP addresses is particularly vulnerable to such attacks.

Drawing on the classification from [14], we will consider two paradigmatic scenarios where an attacker attempts a Sybil attack using many IP addresses:

(i) **Institutional attacks**, launched by an institution or an organization that owns large IP address blocks; and

(ii) **Botnet attacks**, where many infected machines are controlled by an attacker.

The crucial difference between these two attacks is that in an institutional attack the attacker may control many IP addresses located in a limited number of continuous address blocks, whereas in a botnet attack the attacker may control a smaller number of addresses in the whole IP address space. These properties allow us to implement efficient defenses by biasing our sample selection to limit the influence of any given entity (Section 3.3). From a practical perspective, these two attacks represent the two extremes of a continuous spectrum, as most actual attacks will usually fall somewhere in the middle, a point we return to in our evaluation.

We do not consider network-level attacks such as BGP hijacks in our attack model, however we discuss these attacks and potential defenses in Section 7.

3 THE BASALT ALGORITHM

BASALT leverages three main components. First it employs a novel sampling approach, termed *stubborn chaotic search*, that exploits ranking functions to define a dynamic target random graph (i.e. a set of v target neighbors for each node) that cannot be controlled by Byzantine nodes. Second, it adopts a *hit-counter* mechanism that favors the exploration of new peers even in the presence of Byzantine nodes that flood the network with their identities. Finally, it incorporates hierarchical ranking functions that ensure that nodes sample their peers from a variety of address prefixes. The first two mechanisms ensure that the number of Byzantine nodes in a node's view cannot be increased arbitrarily by attackers. This offers protection from general Byzantine behaviors including those resulting from botnet attacks, as defined above. The third mechanism ensures that nodes sample their peers from a variety of address prefixes, thereby

Table 1: Parameters of the BASALT algorithm and of its environment.

	Environment parameters	
n	Number/equivalent number of nodes	1000, 10000
f	Fraction/equivalent fraction of malicious nodes	10%, 30%
Q	Number of correct nodes	$= (1-f)n$
F	Attack force (described in Sec. 4.2.1)	≥ 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 $v/2$
	Theoretical model variables	
t	Time	
$c(t)$	Number of correct node identifiers seen	$0 \leq c(t) \leq Q$
$b(t)$	(Equivalent) number of malicious node identifiers seen	$0 \leq b(t) \leq fn$
$B(t)$	Probability of sampling a Byzantine node	$= \frac{b(t)}{b(t)+c(t)}$

countering institutional attacks where the attacker controls a limited number of entire address prefixes.

Table 1 shows an overview of the parameters of our algorithm and of its environment, while 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 neighbors or potential neighbors.

3.1 Stubborn Chaotic Search

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. In the following, we first detail the use of ranking functions to identify target neighbors. Then we discuss how nodes update these ranking functions to make the random graph dynamic.

Identifying neighbors through ranking functions. Each node maintains a view, view[·], composed of v *slots*. For each slot, $i \in \{1, \ldots, v\}$, it chooses a random seed, noted seed[i] (line 5 of Algorithm 1, and Fig. 1) that defines a corresponding random *ranking* function, $\text{rank}_{\text{seed}[i]}(\cdot)$. We then define a node's i-th out-neighbor in the target graph as the (correct or malicious) node p that minimizes $\text{rank}_{\text{seed}[i]}(p)$. The function $\text{rank}_{\text{seed}[i]}(\cdot)$ can be selected to implement specific sampling distributions. For instance, using a simple hash function $\text{rank}_{\text{seed}[i]}(p) = h(\langle \text{seed}[i], p \rangle)$ (where angle brackets represent a tuple) leads to a uniform sampling function, since each peer identifier has the same probability of producing the lowest rank. In Section 3.3, we present how a hierarchical ranking function allows BASALT to foil institutional attacks. For simplicity, we use the shortcut of saying that a peer p better matches seed[i] than a peer p' if $\text{rank}_{\text{seed}[i]}(p) < \text{rank}_{\text{seed}[i]}(p')$.

When selecting seed[i], a node cannot know the corresponding target identifier. Rather, it 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 at lines 7-9 in order to discover new peers that can serve as better matches for the slots in their views. Specifically, every τ time units

---
[1]We discuss the influence of the composition of this bootstrap set in Section 4.2.2.
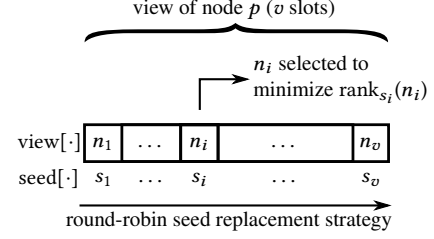
Figure 1: The mechanism of BASALT

Algorithm 1: The BASALT algorithm

1 **algorithm parameters**
2   *see Table 1*

3 **initialization**
4   **for** $i \in 1, \ldots, v$ **do**
5   seed[i] ←rand_seed(); view[i] ← ⊥; hits[i] ← 0
6    $r \leftarrow 1$; updateSample(*bootstrap_peers*)

7 **every** τ **time units**
8    $p \leftarrow$ selectPeer(); **Send** ⟨PULL⟩ **to** p
9    $q \leftarrow$ selectPeer(); **Send** ⟨PUSH, view[·]⟩ **to** q

10 **on receive** ⟨PULL⟩ **from** p
11   **Send** ⟨PUSH, view[·]⟩ **to** p
12 **on receive** ⟨PUSH, $[p_1, \ldots, p_v]$⟩ **from** p
13   updateSample($[p_1, \ldots, p_v, p]$)

14 **every** k/ρ **time units**
15   **for** $i = 1, \ldots, k$ **do**
16      $r \leftarrow (r \bmod v) + 1$
17   **Sample** view[r]
18   seed[r] ←rand_seed()
19   updateSample(view[·])

20 **function** updateSample($[p_1, \ldots, p_m]$)
21   **for** $i \in 1, \ldots, v$, $p \in [p_1, \ldots, p_m]$ **do**
22   **if** p = view[i] **then**
23   hits[i] ← hits[i] + 1
24   **else if** view[i] = ⊥ **or** $\text{rank}_{\text{seed}[i]}(p) < \text{rank}_{\text{seed}[i]}(\text{view}[i])$ **then**
25   view[i] ← p; hits[i] ← 1
26 **function** selectPeer()
27    $i \in \text{argmin}_{j=1}^{v}(\text{hits}[j])$
28   hits[i] ← hits[i] + 1
29   **return** view[i]

(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). When it receives the reply to the pull request, the node greedily updates any slot view[i] that can be brought closer to its corresponding seed, seed[i], using one of the received identifiers (lines 24-25). The peer to which a push message was sent does the same on its side.

Making the graph dynamic. To generate a dynamic random graph and enable nodes to continuously generate fresh samples from the

network, nodes regularly reset some of their seeds to new random values. This periodic operation, at lines 14-19, first provides the application with k peer identifiers representing a random sample of the network (line 17), and then resets the k corresponding slots by selecting new seeds seed[r] (line 18). These k slots are selected in a round-robin fashion every k/ρ time units. This yields ρ random samples per time unit on average as indicated in Table 1. It then sets the corresponding entries, view[r], to the identifiers from the current view that best match the new seeds (line 19). When the algorithm returns view[r] as a sample to the application, view[r] effectively results from a random selection amongst all the peer identifiers received since the last reset of seed[r]. A node has no way of knowing if it has found the peer p that best matches seed[r] globally (i.e. its target neighbor in the random graph), but selecting which seeds to reset in a round-robin fashion and by sampling view[r] just before resetting seed[r], the algorithm ensures that a maximum number of identifiers have been seen for each seed when returning the corresponding sample. This optimizes the randomness of the sample for a given budget of peer exchanges and view size.[2]

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, this means that each slot is refreshed on average every v/ρ time units. The value of v/ρ must therefore be large enough with respect to the exchange interval, τ. Parameter k controls, instead, the number of slots that are reset at the same time. A large value of k causes the 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. This increases the probability of contacting peers that have already been contacted recently, thereby leading to slower convergence for the k unconverged slots. Our experiments by simulation confirm that BASALT better resists the presence of Byzantine peers using a batch sampling-and-replacement strategy where k can be as high as $v/2$ (in which case k/ρ, like v/ρ, must also be at least several exchange intervals, τ), rather than replacing seeds one by one (i.e. setting $k = 1$).

3.2 Hit Counter Hardening Mechanism

The graph-generation mechanism described above prevents Byzantine peers from influencing the target graph, and thus the views of correct nodes once the network has converged. However, depending on the speeds at which correct nodes discover other correct or Byzantine peers, their intermediate views may suffer from a bias in favor of Byzantine nodes. If this happens, the algorithm will tend to select malicious nodes to push to and pull from (lines 8 and 9), further slowing down convergence.

BASALT mitigates this issue by introducing a hit-counter mechanism that effectively makes the protocol harder to attack. Each node maintains a *hit counter* variable, hits[i], for each slot $i \in \{1, \ldots, v\}$ in its view. A node sets hits[i] to 1 when initializing the slot, as well as whenever updating view[i] with a peer that better matches the

corresponding seed (line 25). Every time a node receives another peer's view that also contains peer view[i], it increases hits[i] by one (line 23).

When deciding which neighbors to contact, a node always selects one of the peers with the lowest value of hits[i] (line 27). Finally, the node increases the hit counter of the selected peer by 1 to make it less likely to be selected the next time (line 28).

This mechanism has no impact on honest nodes as they should each appear as often in expectation. However, it creates a trade-off for (possibly colluding) malicious nodes that try to be over-represented, as nodes attempting to appear more often will automatically be contacted less. We further discuss this aspect and the possibility of attacks on the hit counter mechanism in Section 4.3.

3.3 Hierarchical ranking

Central to Algorithm 1, the function rank$_{seed[i]}(\cdot)$ induces a specific sampling distribution of node identifiers. For instance, using simply a hashing function for rank$_{seed[i]}(\cdot)$ yields uniform node sampling. Unfortunately, uniform sampling makes it relatively easy for institutional attackers to gain an overwhelming influence in the system, by taking the control of large IP address blocks to implement Sybil attacks (Section 2.2). For instance, as we will see in Section 4.4, controlling one of the largest ISPs would grant an attacker about 10^8 IPv4 addresses, a number large enough to thwart most existing decentralized BFT systems. Such attacks, are, however, heavily concentrated in a limited number of address ranges by design (~5700 address blocks in the above example). The key idea for countering them therefore consists in not sampling peers uniformly, but using ranking functions that induce some diversity in the sampled node identifiers, and thus reduce the probability of sampling Sybil peers.

Ranking functions and target distributions. Formally, let S be a uniform random variable on 256-bit integers, which corresponds to the sampling of a seed. Let X be the random variable corresponding to the *best matching peer sample* for S, defined as:

$$X = \text{argmin}_{p \in \mathcal{N}} \text{rank}_S(p) \tag{1}$$

where $\mathcal{N}$ denotes the set of all network nodes. Depending on the definition of rank$_S(p)$, X can implement a specific probability distribution on network nodes. This allows us to define the attacker's *power*, f, as the probability of X being a malicious node given a specific ranking function rank$_S(\cdot)$. If rank$_S(\cdot)$ is a simple hashing function, rank$_S(p) = h(\langle S, p \rangle)$, nodes are selected uniformly, and f corresponds to the fraction φ of malicious nodes in the system. In the more general case, this probability is no longer equal to φ but, as we will see in Section 4.1, it plays the same role in our analysis, thus we will also call f an *equivalent fraction* of malicious nodes.

Selecting the ranking function. In order to counter institutional Sybil attacks, we need to select a ranking function that minimizes f by giving malicious nodes a low probability of being selected as best-matching peers (i.e. chosen by the distribution X). BASALT adopts a ranking function that spreads sampled peers amongst different subnets by exploiting the structure of IP addresses. IP addresses can indeed usually be decomposed in two parts, a prefix, that designates the *subnet* to which the address belongs (linked to a given Internet service provider), and a local part that identifies a node within that subnet.

---

[2]Other than using more memory, increasing the view size has a non-negligible networking cost as one would typically keep an open TCP connection ready for each peer of the current view. Alternatively, the samples' randomness could be increased by keeping a log of recently closed connections and re-injecting these peer identifiers when selecting new seeds.

A first grouped ranking function. To illustrate this intuition, suppose that $G(p)$ corresponds to the prefix of a given length of the IP address, or a country code determined from the address. The following ranking function (based on a lexicographical ordering on values) can be used to sample uniformly amongst the different values of property $G(p)$, and then uniformly amongst all the peers that have the selected value of $G(p)$:

$$\text{rank}_S(p) = \langle\, h(\langle S, G(p)\rangle),\, h(\langle S, p\rangle)\,\rangle$$

Using such a ranking function makes an attack against BASALT harder. In order to gain a power of f in the network, a malicious entity would need to control a large number of nodes at least in a fraction f of all the values of $G(p)$ where network nodes exist. For instance, consider an attacker that owns a full IP address block. Uniform node sampling gives the attacker a power of $f = \frac{q}{n}$, where q is the size of the IP block and n the total number of nodes in the network. In the group-based sampling model, since an address block is usually associated with a single group (a single country, a single IP address prefix), the attacker only has a power of $f = \frac{1}{|G|}\frac{q}{g}$, where $|G|$ is the number of different groups, and g is the number of nodes present in the particular group of the attacker's address block. This attacking power is trivially bounded by $\frac{1}{|G|}$, and the only way to increase it consists in taking control of many IP addresses in other groups, making such an attack much more costly.

BASALT*'s hierarchical ranking function.* In BASALT, we take the grouping approach described above one step further. We adopt a hierarchical ranking function that descends the address hierarchy by sampling uniformly at levels /8, then /16, then /24, and then finally at the level of individual addresses, defined as follows:

$$\text{rank}_S(p) = \left\langle h(\langle S, p_0^8\rangle), h(\langle S, p_0^{16}\rangle), h(\langle S, p_0^{24}\rangle), h(\langle S, p\rangle) \right\rangle \quad (2)$$

where p_0^i corresponds to the prefix constituted of the i most significant bits of p's IP address. The efficiency of this ranking function in countering institutional Sybil attacks is demonstrated numerically in Section 4.4.

4 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 attacker's power. The use of f allows us to apply the same analysis to institutional and botnet attacks in a unified reasoning.

4.1 Parameters, Notations and Assumptions

4.1.1 Scenario parameters and node distribution. We first consider an ideal 'uniform' botnet attack, in which Byzantine identifiers follow the same distribution as those of honest nodes. This situation corresponds for instance to a scenario in which a botnet indiscriminately targets the same kind of nodes (e.g. personal machines) as those making up the rest of the system. In this case, the attacker's power f that we introduced in Section 3.3 is simply equal to the fraction of Byzantine nodes in the network, and is independent of BASALT's hierarchical ranking function. To analyze this attack, 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.

In the case of an institutional attack, the attacker's power f depends on the distribution of the address blocks it controls, and represents the probability of selecting a Byzantine identifier using the ranking function $\text{rank}_S(p)$ in the hypothetical case that all identifiers in the network are known (Eq. 1). In this scenario, we define n as an *equivalent network size*, defined as $n = \frac{Q}{1-f}$, where Q still denotes the number of correct nodes. These definitions satisfy the equality $Q = (1 - f)n$, as in the (uniform) botnet attack, and will allow us to apply the same analysis seamlessly.

The two above scenarios represents the extreme cases of a wider spectrum of attacks. In particular, botnet attack might not be uniform, as the identifiers controlled by a botnet might be biased towards certain blocks (e.g. in the case of botnet built by targeting certain organization, or specific vulnerabilities) that differ from those of honest nodes. In such hybrid cases, the reasoning for institutional attacks applies.

4.1.2 Notations. 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, noted $C(t)$, and the set of Byzantine identifiers seen by p over the same period, noted $\mathcal{B}(t)$.

One key observation is that, for a fixed $C(t)$, $B(t)$ increases as p hears of new Byzantine identifiers and $\mathcal{B}(t)$ grows, i.e. $C(t) = C(t') \wedge \mathcal{B}(t) \subseteq \mathcal{B}(t') \implies B(t) \leq B(t')$, so that for a given $C(t)$, $B(t)$ is maximum when the node p has learned all Byzantine identifiers circulating in the system.

In the following analysis, we therefore assume a worst case scenario in which correct nodes have been flooded with all existing Byzantine identifiers. (We discuss the actual implementation of this worst case scenario in Section 4.2.1.) For botnet attacks, we have assumed that correct and Byzantine identifiers follow the same distribution, implying that

$$C(t) = \frac{c(t)}{b_{\max} + c(t)}, \quad (3)$$

where $b_{\max}$ is the total number of Byzantine identifiers, i.e. $b_{\max} = fn$. (See Appendix A for a detailed derivation.)

For institutional attacks, we assume that the distribution of correct nodes is independent of the sampling distribution introduced by rank(), and we approximate $C(t)$ using the same form as Eq. 3, where $b_{\max}$ becomes an *equivalent* number of Byzantine identifiers. Considering the case when p knows all correct nodes ($c(t) = Q$), and having defined $n = \frac{Q}{1-f}$, we derive $b_{\max} = f \times \frac{Q}{1-f}$, and hence $b_{\max} = fn$ in this case as well, where n is now the equivalent network size introduced above.

In both attacks, the probability of selecting a Byzantine node, $B(t)$, becomes therefore driven by the number of correct identifiers known to p, $c(t) = |C(t)|$, and the same system equation can be used to study both cases (modulo the redefinition of n for institutional attacks).

4.1.3 Assumptions. For simplicity, we study a version of BASALT without the hit counter-based hardening mechanism, and later discuss its impact in Section 4.3. Algorithm 2 shows the pseudocode corresponding to the hit counter-less version being analyzed. To approximate the system's behavior, we will reason using the mean values of $c(t)$ over all nodes and slots, and assume that the values

---

Algorithm 2: Simplification of Algorithm 1 for theoretical analysis of Section 4.2

---

1 function updateSample($[p_1, \ldots, p_v]$)
2   **for** $i \in 1, \ldots, v$ **do**
3   **for** $p \in [p_1, \ldots, p_v]$ **do**
4   **if** view$[i]$ = $\perp$ **or**
             $h(\langle \text{seed}[i], p \rangle) < h(\langle \text{seed}[i], \text{view}[i] \rangle)$ **then**
             view$[i] \leftarrow p$
5 function selectPeer()
6 $i \leftarrow$rand$(1, \ldots, v)$; **return** view$[i]$

---

of individual nodes tend to concentrate around their means in practice with high probability, as is usually the case in such stochastic systems.

4.2 Analysis of the Core Mechanism

We first discuss in more detail the worst-case attack on the BASALT algorithm. We then study the risk of a node becoming isolated under this attack model (i.e. of an Eclipse attack succeeding), before moving on to studying the convergence properties of BASALT assuming no node is ever isolated.

4.2.1 Identifying the Worst-Case Attack. To identify the worst case attack, we observe that attackers cannot influence the choices correct nodes make (at line 6 of Algorithm 2, and at lines 16-18 of Algorithm 1); thus they can only manipulate the peer-sampling process by increasing their representation in the views of correct nodes, i.e. the value of $B(t)$. The fact that $B(t)$ grows with the set of Byzantine identifiers the node is aware of, $\mathcal{B}(t)$, suggests that the worst case scenario arises when Byzantine nodes flood the network with their identifiers in order to increase $\mathcal{B}(t)$ as much as possible. We model this attack scenario as follows:

- 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.

We define the *force* of the attack, F (distinct from the attacker's power, f), as the ratio between the number of push requests sent by a Byzantine node and 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 from correct nodes than from Byzantine ones.

The worst case corresponds to an arbitrarily large value of F, arising when correct nodes receive all the identifiers of Byzantine peers in any arbitrarily small (but non-empty) time interval. This means that apart from the initial state, $\mathcal{B}(t)$ is constant, and its effect can be captured by the term $b_{\max} = fn$ to compute the probabilities of selecting a correct (resp. Byzantine) peer in a slot of a node's view. We recall that fn represents the total/equivalent number of Byzantine nodes depending on the attack considered. The analysis that follows

shows that even in this case, BASALT causes B to converge to a value that is only slightly larger than the attacker's power, f. The experimental results of Section 5 analyze instead the actual performance with finite values of F.

4.2.2 Bounding the Probability of Isolation. We start by showing that nodes have a low probability of being isolated. Isolation can happen in two ways: either when a node joins the network for the first time, or when it evicts all correct peers from its view and replaces them with Byzantine peers.

Isolated joining node. In the first case, the unfortunate joining node receives all of the identifiers of Byzantine nodes as soon as it joins. At time ϵ after joining we have $\mathcal{B}(t)$ becomes maximal 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 a Byzantine peer, we can write the probability that a node has only Byzantine neighbors as $B(t)^v$.

$$B(t)^v = \left(\frac{b_{\max}}{b_{\max} + c} \right)^v = \left(\frac{1}{1 + (1 - f_0) \frac{I}{fn}} \right)^v \quad (4)$$

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 ($I = \frac{1}{4} fn$), this probability 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 peers. When such a reset occurs, the probability that all of the non-reset slots are already owned by Byzantine peers is equal to $B(t)^{v-k} = \left(\frac{b_{\max}}{b_{\max} + c(t)} \right)^{v-k}$. When the number of correct nodes seen locally, $c(t)$, is large enough, this probability is negligible.

Let us now study 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. For this analysis, we look at a single node of the network and make the hypothesis that other network nodes are well-converged. As we discuss in Sections 4.2.3 and 5, 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. The expected number of correct peer identifiers received during the period between the two resets is lower bounded by $\frac{k}{\rho} \frac{v}{\tau} \frac{c_0}{fn+c_0}(1 - f)$. If we write Δc the corresponding increase in $c(t)$, i.e. the number of *distinct* correct peer identifiers received during this time period, we obtain:

$$\Delta c \geq \frac{kvc_0(1 - f)(Q - c_0)}{Q\tau\rho(fn + c_0) + kvc_0(1 - f)} \quad (5)$$

(see Appendix B for the full derivation).

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. $B(t)^{v-k}$ is smaller than 10^{-10} as soon as the number $c(t)$ of correct node identifiers seen is more than 585. In other words, the probability that the node becomes isolated at the next reset is negligible. This guarantee can be made even stronger by increasing the view size v. Moreover, if the node is already in a better-converged state with a relatively large c_0, the probability of becoming isolated during a reset becomes even smaller.

4.2.3 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. In particular, $B(t)^v$ is negligible at all times t.*

Deriving a continuous model. When Assumption 1 holds, and in the worst-case scenario discussed above (Byzantine nodes have propagated all their identities to all correct nodes) $\mathcal{B}(t)$ is constant, and its effect captured by the term $b_{\max}$, 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 receives only Byzantine peer identifiers that it is already aware of (by the worst-case assumption $b_{\max} = fn$). With probability $C(t)$, the node contacts instead 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 variation of $c(t)$ over time as a result of a pull operation as follows:
$$\frac{dc}{dt} = \frac{1}{\tau}\left[C(t)^2 v\left(1 - \frac{c(t)}{(1-f)n}\right)\right].$$

- *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:
$$\frac{dc}{dt} = \frac{1}{\tau}\left[2C(t)^2 v\left(1 - \frac{c(t)}{(1-f)n}\right)\right] - \rho\frac{c(t)}{v}. \tag{6}$$

Solving the continuous model. We now solve Equation 6 under Assumption 1 and show that the network converges to a state where the proportion B of Byzantine peers in nodes' views is small even for arbitrarily large values of the attack force, F. To this end, 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 Equation 6, we obtain:
$$\frac{dB}{dt} = B(1-B)\left(\frac{\rho}{v} - \frac{2v(1-B)(B-f)}{\tau f(1-f)n}\right) \tag{7}$$

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 with that of ρ. We obtain after a few steps:
$$(1-B)(B-f) = \frac{\rho f(1-f)n}{2v^2}. \tag{8}$$

The equation exhibits two roots $B_1 < B_2$.
$$B_{1,2} = \frac{1}{2}\left(1 + f \mp \sqrt{(1-f)^2 - 2\frac{\rho f(1-f)n}{v^2}}\right) \tag{9}$$

When the quantity on the right-hand side of Equation 8 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 Equation 8 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 globally well mixed 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, for fixed values of f and n, the view size v should grow proportionally to the square root of the sampling rate $\sqrt{\rho}$, while, for fixed values of f and ρ, it needs to increase proportionally to $\sqrt{n}$.

4.3 Analysis of the Hardening Mechanism

BASALT's hit counter-based hardening mechanism allows nodes to detect which peers have appeared more often in incoming messages, and prioritize other peers for network exploration. In the case of a standard attack, where malicious peers flood their own identifiers, the hit counter favors the choice of correct peers over malicious ones.

However, the fact that we analyzed a simplified version of BASALT without the hardening mechanism raises the legitimate question of whether the hit counter may degrade the security of the approach by enabling some other attack. To answer this question, let us consider a malicious node or a coalition of malicious nodes that want to influence the sampling operations performed by a correct node.

We start by observing that malicious nodes can neither write nor read the local memories of correct nodes. So they cannot influence

Table 2: Institutional attack: power f of the adversary, defined as the equivalent fraction of malicious nodes, for different sampling methods, supposing biggest Internet service provider as an adversary ($\sim 10^8$ IP addresses, distributed over 5739 blocks), on the IPv4 network. Data sourced from the GeoLite2 Block/ASN dataset.

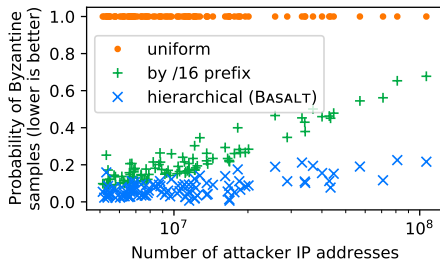| # of honest IPs (Q)	100	1000	10000
Uniform	99.9999%	99.999%	99.99%
By /8 prefix	49%	28%	27%
By /16 prefix	95%	64%	17%
By /24 prefix	99.98%	99.8%	98%
Hierarchical	**47%**	**21%**	**10%**

Figure 2: Institutional attack: probability of sampling a Byzantine node with different ranking functions, calculated for the 100 biggest Internet ASes supposing that each of them is the attacker, and that 1000 honest nodes are uniformly spread amongst remaining IP address space. Data sourced from the GeoLite2 Block/ASN dataset. Horizontal axis: number of IP addresses controlled by the AS. The sampling probability is calculated from the equivalent fraction by using Equation 9, with a view size of $v = 100$ and a sampling rate of $\rho = 1$.

sampling operations directly: their only strategy consists in trying to convince a target correct node that some of the correct peers in its view are malicious by increasing their hit counters. To this end, malicious nodes can repeatedly advertise the identifiers of correct peers. But again, they cannot guess which correct peers are in the target correct node's view slots. So their only option consists in advertising a possibly large random set of correct peers in the hope that some of them will be in the correct node's view.

But even this turns out to be counterproductive. If malicious nodes advertise a large number of correct identifiers, it is indeed possible that they may increase the hit counter of some entries in the target's view. But they do so at the cost of increasing their target's value of $c(t)$, i.e. the number of correct peers known to it. Since we already assumed the worst case scenario of $b_{\max} = fn$, the increase in $c(t)$ can only decrease $B(t)$ thereby making the attack counterproductive.

4.4 Numerical Analysis: Institutional Attacks

To illustrate the robustness of BASALT's hierarchical ranking function against an institutional attack, we calculate the power f (here an equivalent fraction of malicious nodes) of a real-world attacker using data from the GeoLite2 Block/ASN dataset [5], and use the equilibrium formula of Equation 9 (B_1) to compute the proportion

of Byzantine nodes that BASALT would return in such a scenario. We assume that the attacker is an internet autonomous system (AS), that exploits all the IP addresses it owns to attack BASALT, and that a certain number of honest nodes (100, 1000, 10000) are uniformly spread amongst the remaining currently active IP addresses. Table 2 shows the power f of such an attacker, supposing that the attacker is the Internet AS with the largest number of currently active addresses (106 million in the dataset we used, spread over 5739 blocks). This calculation shows that the hierarchical sampling method reduces the power of the attacker down to 21% when only 1000 honest nodes run BASALT, where it would have been above 99.99% (i.e. full control on the network) using uniform sampling. Figure 2 shows the corresponding probability that BASALT will return Byzantine nodes by applying Equation 9 for the 100 biggest Internet ASes, assuming 1000 uniformly spread honest nodes.

5 EXPERIMENTAL EVALUATION

We complement our theoretical analysis with Monte Carlo simulations that illustrate BASALT's dynamic behaviour. In this section, we focus on simulating a permissioned system with a known fraction of malicious nodes. We do not simulate the IP address distribution and use the uniform ranking function. As explained above, our observations can be transposed to a permissionless setting, where the attacker's power defined by the hierarchical ranking function plays the role of an equivalent fraction of malicious nodes. We show that BASALT consistently produces samples with fewer malicious peers than the state-of-the-art algorithms Brahms [8] and SPS [16] 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 have an influence on the convergence time of epidemic agreement algorithms.

5.1 Experimental Setting

We evaluate the tested algorithms by simulating a system with n nodes, of which a fraction f implement the malicious behaviour described in Section 4.2.1. We do not simulate message loss or variable link latencies, as our model parameter F (the attack force) already integrates the possibility of message loss (see Section 4.2.1), and variable link latencies can also be modeled as losing messages that arrive after a certain delay. 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. We vary the two parameters v, the view size, and ρ, the sampling rate, of the algorithm, as well as the force of the attack, F. We fix the exchange interval to $\tau = 1$ (1 simulation time step). Unless stated otherwise, we use $F = 10$ and $\rho = 1$. All algorithms were implemented in a same simulation framework written in Rust, totaling about 2500 lines of code [3].

We compare BASALT (Algorithm 1) and its variant without the hardening mechanism (Algorithm 2, BASALT-simple) to two state-of-the-art competitors: Brahms [8] and SPS [16].

SPS was unable to function at all in the tested scenarios: for instance for $n = 1000$, $f = 30\%$, and even with an attack force F of 0,

---

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

(a) Varying the fraction f of malicious nodes

(b) Varying the attack force F

(c) Varying the sampling rate ρ
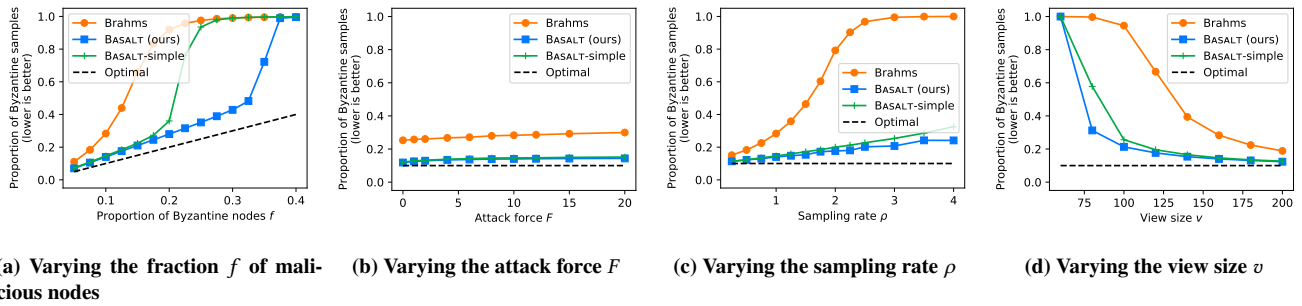
(d) Varying the view size v

Figure 3: Our algorithm (boxes, blue) consistently provides samples that contain fewer Byzantine nodes than our competitor, Brahms, in a variety of situations. A proportion of 1 of Byzantine samples, as exhibited in Fig 3a for the highest values of f, corresponds to a situation where malicious nodes are able to cause a network partition. Results shown for a network size of 10000 nodes, with a base proportion $f = 0.1$ of malicious nodes. Base values for other parameters are $v = 160$, $\rho = 1$, $F = 10$. BASALT corresponds to the complete version of our algorithm, whereas BASALT-simple corresponds to Algorithm 1 without the hardening mechanism (modifications of Algorithm 2).

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. We have thus decided to exclude SPS from our comparison charts, and concentrate on the comparison of BASALT against Brahms.

To compare Brahms and BASALT on similar grounds, we add to the Brahms algorithm a mechanism that resets some of the hash functions regularly, using the same round-robin strategy as BASALT. Without such a mechanism Brahms would always return the same fixed set of samples, limiting its usability as a random peer sampling algorithm. As we will show just below, adding a reset rate to Brahms makes it less resilient to malicious nodes. In terms of communication overhead, Brahms and BASALT have the same cost. Indeed, both algorithms send a set of peer identifiers of size v when replying to a pull request. For push requests, BASALT uses larger messages since Brahms does not send the view with a push message, only the sending node's identifier, whereas we send the whole view of size v. However, supposing $v = 200$ (the maximum in our experiment) and node identifiers of size 4 bytes (such as IPv4 addresses), the size of the communicated information is smaller than one MTU (maximum transmission unit, i.e. maximum size of a single packet, which is about 1500 bytes on the Internet), thus the same number of Internet packets need to be sent by both algorithms.

5.2 Proportion of Byzantine Samples

In our first experiment, we measure the number of Byzantine nodes present in correct nodes' samples on average 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 vary the parameters f, ρ, v and F individually. Figure 3 shows how this proportion evolves for the three algorithms evaluated, as one of the parameters f, ρ, v and F varies.

Plot 3a shows how the algorithms behave when the proportion of Byzantine nodes in the system varies. BASALT provides close to optimal proportions of Byzantine samples even with many Byzantine nodes, whereas Brahms fails to contain the attack in this domain.
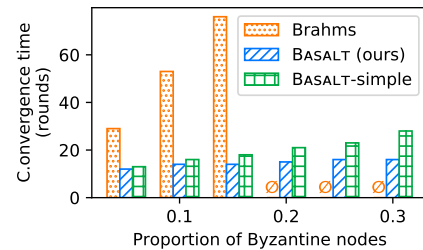

Figure 4: Time to convergence within 25% of optimal proportion of Byzantine samples, for $n = 1000$, $v = 100$ (on the right part, Brahms does not converge within experiment time)

Plot 3b shows the sensitivity of the algorithms to the force of the attack F. These plots show that BASALT is almost insensitive to F, whereas Brahms shows an increasing proportion of Byzantine samples when F increases.

Plot 3c shows how the algorithms behave for various values of the sampling rate ρ. For low values of ρ, both Brahms and BASALT are able to converge to high quality samples, 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 of the views, where 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. This plot also shows how the hit-counter variant helps BASALT attain better states when ρ is high.

Plot 3d 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.

5.3 Evaluating Convergence Speed

In this second experiment, we study the speed at which the algorithms converge to good network states, where they provide samples
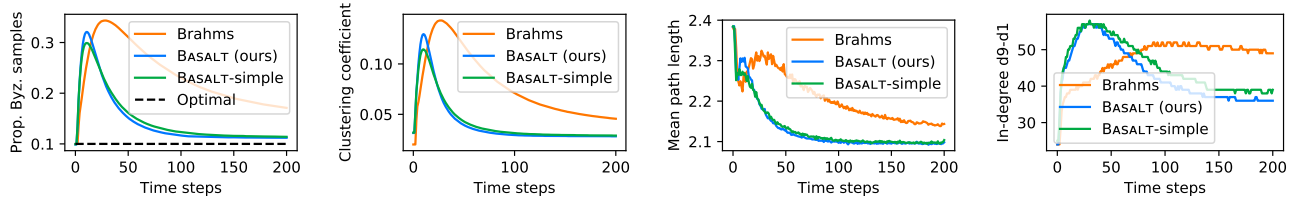
Figure 5: Algorithm convergence on several graph quality metrics, for $n = 10000$, $f = 10\%$, $F = 1$, $\rho = 0.5$, $v = 160$. On all metrics, lower is better: we see that BASALT converges much more rapidly than Brahms.
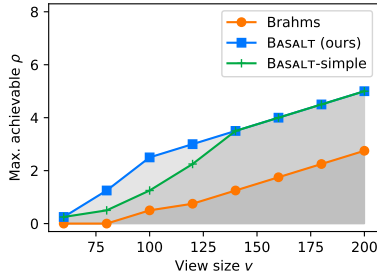

Figure 6: Maximum achievable sampling rate ρ (see Section 5.4) for 10000 nodes, $f = 10\%$

with low proportions of malicious nodes. Figure 4 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 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 5 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 $F = 1$. These plots show that BASALT converges much faster than Brahms to a good network state (Brahms does not converge according to the previous criterion within the time of the experiment). The other plots show metrics for graph quality, where the algorithms exhibit a similar convergence behaviour: 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.

5.4 Node Isolation vs. Sampling Rate

We have seen earlier (Plot 3c) 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 is failed. We plot the successful runs with highest values of ρ for a given v. The results of this experiment are shown in Plot 6 for $N = 10000$, $f = 10\%$ and $F = 10$. The areas delineated in Plot 6 correspond to the parameter sets that give successful runs. Our results show that for similar view sizes, BASALT achieves much higher sampling rate than Brahms, thus providing more utility to the application.

6 LIVE DEPLOYMENT

We implemented BASALT in the AvalancheGo engine [4], the main implementation of the AVA network [1] which uses the Avalanche consensus algorithm[4]. 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-lines patch to the Go source code of AvalancheGo, replaces peer sampling based on stake in a proof-of-stake system by peer sampling based on BASALT, including the hierarchical ranking function described in Section 3.3.

Our implementation integrates seamlessly with the AVA protocol and is fully compatible with the existing network. Our implementation supports managing current outgoing connections according to the BASALT algorithm, instead of keeping connections open to all reachable network nodes as done by the original AvalancheGo implementation[5].

To show that BASALT can be applied as a sampling method that reduces the risk of an institutional attack, we ran a 10-hour experiment where we launched 100 "adversarial" Avalanche nodes on the public AVA network (corresponding to about 20% of total active nodes) in an attempt to bias sampling in their favor using a Sybil attack against one of our nodes. The nodes we launched all had IP addresses located in the same /24 prefix, owned by our research institution. Samples were measured at witness nodes running the BASALT sampling algorithm, as well as the non-hierarchical variant of BASALT and a sampling algorithm based on full network knowledge. Results shown in Table 3 show that using BASALT, the

---

[4]Our code is publicly available at https://github.com/basalt-rps/avalanchego-basalt. Our implementation is forked from the official AvalancheGo repository [21]. Our changes are identified by "Basalt RPS Authors".

[5]Unfortunately, we had to disable this behaviour as it led to too many connection attempts and some nodes appeared to have banned our IP addresses as a consequence. A simple modification allows our code to never close connections intentionally: the view maintained by BASALT is only used to sample peers for the Avalanche consensus algorithm, and connections are kept in the background to nodes that have been removed from the view.
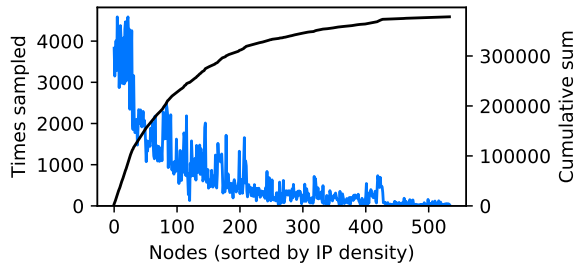
Figure 7: Behaviour of AvalancheGo modified with BASALT running on the public AVA network, 5-hour experiment. On the left, nodes that are alone in their IP prefix are sampled the most frequently. On the right, nodes that belong to IP prefixes where other network nodes are present are sampled less often.

Table 3: Observed proportion of samples that are nodes controlled by the adversary in our live experiment (see Section 6)

Algorithm	Adversary samples
Full knowledge uniform sampling	18.4%
BASALT-uniform	17.5%
BASALT (hierarchical)	**1.13%**
True proportion of Byzantine nodes	18.8%

probability of sampling one of our adversarial nodes is brought to about 1%, meaning that the influence of our nodes in the network is extremely limited.

To show the wider benefit of BASALT, we plot in Figure 7 the number of times the various nodes of the AVA network were sampled in the experiment. Sorting nodes by a density metric which counts the number of other nodes in the same /8, /16 and /24 prefix reveals that nodes which are isolated in their prefix (on the left of the graph) are sampled more often than nodes which share their IP prefixes with other nodes (on the right). However all network nodes have a chance of being sampled, and no single node is sampled exceedingly more often than others. This stands in contrast with Proof-of-Stake-based sampling, where sampling frequency is proportional to the stake invested, a mechanism that gives disproportionate power to rich nodes and totally excludes nodes that are not able to invest any stake in the network.

7 DISCUSSION AND RELATED WORK

Random peer sampling in non-adversarial settings is a well-studied problem [15, 23]. Surprisingly, very few works have sought to develop Byzantine-tolerant RPS protocols.

State-of-the art methods such as Brahms [8] and Secure Peer Sampling [16] (SPS) are based on a classical RPS algorithm, to which is adjoined a mechanism that tries to correct for the over-representation of malicious nodes. In Brahms, the view is not updated if a peer has received more than a certain number of push messages in a given time slot. Albeit vaguely similar to our hit counter mechanism, Brahms' approach can only work if we assume that malicious nodes have limited total firing power and must therefore target their attack on a specific victim node. Otherwise, they would be able to simply flood the whole network with many pushes and halt the peer

sampling algorithm completely. In SPS, nodes try to build some statistical knowledge on node behaviour; however, 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. Our protocol, on the other hand can effectively handle these attacks. Moreover, the majority of these systems do not address risks that exist in real-world networks, such as Sybil attacks. To our knowledge, the only exception is HAPS [6], which is designed specifically to handle Sybil attacks. HAPS, however, only addresses Sybil attacks in which attackers are concentrated in a few IP blocks ("institutional attacks"), by using random walks on a carefully crafted probabilistic tree. Due to its design, 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.

Recent works on blockchains have also brought to lighten the risk of attacks at a more fundamental level than those described in Section 2.2. 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* [18].

Note that BGP hijacking attacks are necessarily limited to one or a few IP prefixes, as large-scale routing attacks would likely bring down large parts of the Internet and would be noticed immediately. By spreading connections over a variety of IP prefixes through its rank function, BASALT builds intrinsic resilience to these attacks as at most only a small fraction of nodes' neighbors will be located in hijacked prefixes. In this way, the global BASALT network is not at risk of being taken down or manipulated by a malicious entity.

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 (an Eclipse attack). Defenses have been proposed against Eclipse attacks at the network level: for instance, the SABRE network [7] proposes to use additional communication channels, in the form of a network of specialized nodes that are all connected to one another using dedicated channels, 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.

In the case of a blockchain, where the most crucial property to guarantee safety is that all nodes are made aware of new blocks rapidly, the SABRE method is able to help by providing reliable block delivery. For sampling-based methods that use BASALT, SABRE could provide a security mechanism at the application layer to enable detection of network attacks and stop 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.

Finally, one could argue that it will be hard to bootstrap a BASALT network containing enough nodes to effectively counter botnet attacks. We note that this problem is exactly the same as in PoW-based cryptocurrencies, as an attacker that gains > 50% of the network's hashing power can overturn the network in their favor (which is easy to do for smaller cryptocurrencies that don't have a lot of hashing power allocated to them). A PoW-based cryptocurrency network is secured by members investing in providing lots of hashing power, as is the case e.g. for Bitcoin, in order to make a > 50% attack so costly that it is impossible in practice (or simply not worth it compared to the value of the cryptocurrency that could be stolen). A BASALT-based cryptocurrency is similarly secured by participants investing in running as many nodes as possible from many different IP prefixes, which they have an incentive to do in order to keep the system safe. Moreover, BASALT has the advantage that this investment does not require the waste of tremendous quantities of energy.

8 CONCLUSION

We have presented a new algorithm for Byzantine-tolerant random peer sampling on the Internet that uses biased sampling to prevent Sybil attacks. Such an algorithm can be used to implement sampling-based consensus algorithms such as Avalanche. Contrary to sampling algorithms based on Proof-of-Stake, such as those currently in use on the AVA network, BASALT allows the network to be truly open by allowing any Internet 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 we have brought an important contribution to making such methods applicable in large-scale open networks.

A $C(C, \mathcal{B})$ IN A BOTNET ATTACK

The probability $C(C, \mathcal{B})$ depends on the distribution of correct and Byzantine identifiers across the three levels of blocks used in Equation 2. We fix one node p selected randomly amongst $C \cup \mathcal{B}$, and write selected(p) the event that p is selected by the ranking function $\text{rank}_S(\cdot)$:

$$\text{selected}(p) \equiv \left(p = \text{argmin}_{q \in C \cup \mathcal{B}} \text{rank}_S(q) \right). \tag{10}$$

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

In our model, a botnet attack corresponds to the (ideal) case in which Byzantine and honest nodes follow the same distribution across IP blocks. As a result, they are indistinguishable from the point of view of $\text{rank}_S(\cdot)$, which means here that the events $p \in C$ and selected(p) are independent. This independence implies that

$$\begin{aligned} C(C, \mathcal{B}) &= \Pr\left(p \in C \mid \text{selected}(p) \right) \\ &= \Pr\left(p \in C \right) \\ &= \frac{|C|}{|C| + |\mathcal{B}|}. \end{aligned} \tag{11}$$

B DERIVING EQUATION (5)

Based on the result from the coupon collector's problem, the expected number of uniformly distributed (non-distinct) correct peer identifiers that must be received in order to learn Δc new **distinct** correct peer identifiers amongst Q, when c_0 are already known, is:

$$\frac{Q}{Q - c_0} + \frac{Q}{Q - c_0 - 1} + \cdots + \frac{Q}{Q - c_0 - \Delta c + 1} \tag{12}$$

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

$$\frac{k}{\rho} \frac{v}{\tau} \frac{c_0}{fn + c_0}(1 - f) \tag{13}$$

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 (12) as follows:

$$(12) \le \Delta c \frac{Q}{Q - c_0 - \Delta c} \tag{14}$$

Moreover, we have $(12) \ge (13)$. Thus:

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

thus

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

thus

$$\Delta c \ge \frac{kvc_0(1 - f)(Q - c_0)}{Q\tau\rho(fn + c_0) + kvc_0(1 - f)}$$

which is the result of Equation (5).

REFERENCES

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

[2] [n.d.]. Bitcoin Energy Consumption Index - Digiconomist. https://digiconomist. net/bitcoin-energy-consumption. Accessed: 2020-03-05.

[3] [n.d.]. Ethereum.org. https://www.ethereum.org/. Accessed: 2020-02-20.

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

[5] [n.d.]. GeoLite2 ASN CSV Database. https://dev.maxmind.com/geoip/geoip2/geolite2-asn-csv-database/. Accessed: 2020-10-12.

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

[7] Maria. Apostolaki, Marti Gian, MÃ¼ller Jan, and Vanbever Laurent. 2019. SABRE: Protecting Bitcoin against Routing Attacks.. In *NDSS*. 1–15.

[8] 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.

[9] Alan Demers, Dan Greene, Carl Houser, Wes Irish, John Larson, Scott Shenker, Howard Sturgis, Dan Swinehart, and Doug Terry. 1987. Epidemic algorithms for replicated database maintenance. https://dl.acm.org/citation.cfm?doid=41840. 41841

[10] John R Douceur. 2002. The sybil attack. In *International workshop on peer-to-peer systems*. Springer, 251–260.

[11] Toby Ehrenkranz and Jun Li. 2009. On the state of IP spoofing defense. *ACM Transactions on Internet Technology (TOIT)* 9, 2 (2009), 1–29.

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

[13] Rachid Guerraoui, Petr Kuznetsov, Matteo Monti, Matej Pavlovic, and Dragos-Adrian Seredinschi. 2019. Scalable Byzantine reliable broadcast. In *33rd International Symposium on Distributed Computing (DISC 2019)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik.

[14] Ethan Heilman, Alison Kendler, Aviv Zohar, and Sharon Goldberg. 2015. Eclipse attacks on bitcoin's peer-to-peer network. In *24th USENIX Security Symposium (USENIX Security 15)*. 129–144.

[15] 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). https://doi.org/10.1145/1275517.1275520

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

[17] 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. https://doi.org/10.1109/TPDS.2003.1189583

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

[19] Satoshi Nakamoto. 2009. Bitcoin: A peer-to-peer electronic cash system.

[20] Brice NÃ©delec, Julian Tanke, Davide Frey, Pascal Molli, and Achour MostÃ©faoui. 2018. An adaptive peer-sampling protocol for building networks of browsers. *World Wide Web* 21, 3 (May 2018), 629–661. https://doi.org/10.1007/s11280-017-0478-5

[21] Team Rocket. 2018. Snowflake to avalanche: A novel metastable consensus protocol family for cryptocurrencies.

[22] Atul Singh et al. 2006. Eclipse attacks on overlay networks: Threats and defenses. In *In IEEE INFOCOM*. Citeseer.

[23] 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.

[24] Spyros Voulgaris and Maarten van Steen. 2013. VICINITY: A Pinch of Randomness Brings out the Structure. In *Middleware 2013 (Lecture Notes in Computer Science)*, David Eyers and Karsten Schwan (Eds.). Springer Berlin Heidelberg, 21–40.

[25] Fan Zhang, Ittay Eyal, Robert Escriva, Ari Juels, and Robbert Van Renesse. 2017. REM: Resource-efficient mining for blockchains. In *26th USENIX Security Symposium (USENIX Security 17)*. 1427–1444.