Handel: Practical Multi-Signature Aggregation for Large Byzantine Committees

Olivier Bégassat, Blazej Kolad, Nicolas Gailly, Nicolas Liochon
ConsenSys / PegaSys R&D
firstname.lastname@consensys.net
v1.01

Abstract

This paper presents Handel, a Byzantine fault tolerant aggregation protocol that allows for the quick aggregation of cryptographic signatures over a WAN. Handel has both logarithmic time and logarithmic network complexity and needs minimal computing resources. We implemented Handel as an open source Go library with a flexible design to support any associative and commutative aggregation function. We tested Handel with a BLS multi-signature scheme for BN 256 on 2000 AWS instances running two nodes per instance and located in 10 AWS regions. The 4000 signatures are aggregated in less than 900 milliseconds with an average per-node communication cost of 56KB.

1 Introduction

Many large scale decentralized systems need to efficiently establish agreement between thousands of untrusted nodes in a network. For instance, Algorand [22], Dfinity [27] and Tendermint [9] exchange digital signatures on a common message.

A common practical solution is to employ committees. For example, Dfinity uses committees for random number generation and block notarization, both by means of threshold signatures. However, there are issues with using committees in a Byzantine context. One such problem is that of committee creation. Members must be randomly selected, but if the randomness can be manipulated, Byzantine nodes may take control of future committees. In the adaptive adversaries model [11], committee elections must be private and unpredictable. In Dfinity, the randomness and the IP addresses are public, making committee members high profile targets for attackers. A second problem arises from the fact that committees are far more vulnerable than the network as a whole. Indeed, the amount of resources needed to compromise a committee is, by construction, orders of magnitude lower than that needed to compromise the totality of the network. Hence, we argue that using committees comes at the cost of lowering the security of any decentralized system.

Handel is a building block to achieve agreement at scale without committees. As with most Byzantine fault tolerant systems, nodes digitally sign a message to cast a vote. The proportion of signatures required is decided by the application built on top of Handel. When applied to the problem of signature aggregation, the outcome of a Handel execution is a constant-size multi-signature that represents the set of signers' individual signatures. Many practical multi-signature schemes exist. In our evaluation, we used the BLS multi-signature scheme [6, 7].

Aggregating at scale is a well known problem [26, 32, 44, 34]. Cappos et al. proposed San Fermín [12], an efficient and scalable data aggregation protocol. Its time and communication complexity scale logarithmically with the number of nodes through a tree-based communication overlay and a parallel aggregation mechanism. However, San Fermín is *not* Byzantine fault tolerant and relies heavily on timeouts: data sent after a timeout is ignored.

This works well in permissioned environments with homogeneous networks, but is inadequate for fully decentralized systems: short timeouts and high network latency lead to valid contributions being ignored, while long timeouts slow the protocol down. To the best of our knowledge, no Byzantine fault tolerant protocol compares to San Fermín's efficiency. Handel provides **Byzantine Fault Tolerance** and **Versatility** (natively handles heterogeneous network latencies and computer capacities) while replicating the **Speed** (thousands of contributions aggregated in seconds) and **Efficiency** (minimal CPU and network consumption) of San Fermín.

We stress that Handel solves not only signature aggregation at scale, but also *generic data aggregation*. We will thus refrain from using the word *signature* and switch to the more generic term *contribution*.

1.1 High level presentation of Handel

Handel borrows the binary tree-based overlay structure from San Fermín [12]. Thus, every node partitions the set of its peers into pair-wise disjoint peer sets labeled with a level (a nonnegative integer). Peer sets grow exponentially in size with increasing levels, providing the basis for logarithmic time complexity. However, and contrary to San Fermín, all nodes *may eventually* contact all other nodes, contributing to Byzantine fault tolerance.

Rather than contact all peers at a given level simultaneously, nodes contact them one by one. This happens periodically and on several levels in parallel. All communication is one-way and consists of sending levelwise relevant contributions. Nodes can receive and send incomplete contributions. Verification of incoming contributions is triaged: only the most relevant contributions are verified, which lessens CPU resource consumption. At each round of outgoing communication, nodes send their current best aggregate contributions. Completed levels are communicated sooner and more aggressively resulting in faster completion times. Handel uses a scoring and windowing mechanism to prioritize incoming contributions to defend against DOS attacks.

1.2 Deployment scenarios

We describe several deployment scenarios for Handel. The first scenario is the regular deployment of Handel as described by the protocol where the IP addresses of the signers are revealed, with some practical defenses. The second scenario uses anonymity networks such as Tor [18] to hide signers' IP addresses. The last scenario uses ring signatures [41] to securely and anonymously map IP addresses to fresh public key pairs, thereby not exposing the link between the IP address of a signer and its long term signing key.

1.3 Implementation & Evaluation

We implemented Handel as an open-source Go library [5]. It includes many openness points to allow its users to plug different signature schemes or even other forms of aggregation besides signature aggregation. We implemented extensions to use Handel with BLS multi-signatures using the BN256 curve [15, 2]. We ran large-scale tests and evaluated Handel on 2000 AWS nano instances located in 10 AWS regions and running two Handel nodes per instance. Our results show that Handel scales logarithmically with the number of nodes both in communication and resource consumption. Handel aggregates 4000 BN256 signatures with an average of 900ms completion time and an average of 56KBytes network consumption.

1.4 Our contributions

Our contributions are the following: the protocol, (section 3), a detailed analysis of this protocol in our threat model (section 4.4), an open source implementation (section 6), and evaluation results in a large scale context (section 7).

2 Definitions, Threat Model, Goals

This section presents some *terminology* and *concepts*; we describe the *threat model* and the *design goals* Handel achieves.

2.1 Contributions

Handel's purpose is to facilitate the aggregation of pieces of data we call **contributions**. Formally, con-

sider a nonempty set C whose elements are called **contributions**, a set Pub of **public data**, a **verification function** $Ver^{Pub} : C \to \{0,1\}$ that verifies contributions against public data, a **(partial) aggregation function** $Aggr : C \times C \nrightarrow C$ that performs aggregation of contributions and a **weight function** $w : C \to \mathbb{N}$ to compare contributions. A contribution $c \in C$ is **valid** against the public data Pub if and only if $Ver^{Pub}(c) = 1$. In practice, not all pairs of contributions (c, c') may be meaningfully aggregated, hence we insist aggregation be a *partial* function (i.e. defined on a subset of $C \times C$). Define a pair (c, c') of contributions to be **aggregatable** if (c, c') is in the domain of the partial function $Aggr$, and an **aggregate contribution** to be a contribution in the image of $Aggr$. Contributions that aren't aggregate contributions are called **individual contributions**. We shall assume that the (partial) aggregation function satisfies both *commutativity* and *associativity* conditions. The weight function measures how many individual contributions are contained in a given contribution.

For concreteness, let us describe these components in the case of BLS signature aggregation [7]. Suppose there are N participants signing off on a message m. We define C to be $E \times (\{0,1\}^N \setminus \{[0,\dots,0]\})$ where E is the set of points of the elliptic curve used for signatures. Thus, a contribution is a pair $(\sigma, [\epsilon_1, \dots, \epsilon_N])$ where $\sigma \in E$ is a point on the curve, and $[\epsilon_1, \dots, \epsilon_N]$ is an ordered list of bits (not all zero). This ordered list of bits is used to to keep track of whose signatures are (supposedly) included in σ: participant i's signature ought to be "accounted for in σ" iff $\epsilon_i = 1$. The public data is an ordered list $Pub = [pk_1, \dots, pk_N]$ of N public keys. The verification function is defined on $c = (\sigma, [\epsilon_1, \dots, \epsilon_N])$ by

$$Ver^{Pub}(c) = \text{IsOne}\left(e(\sigma, g)^{-1} \cdot e\Big(H(m), \prod_{i=1}^{N} pk_i^{\epsilon_i}\Big)\right)$$

where e is the pairing is used in the BLS scheme (with values in the multiplicative group $\mathbb{F}^\times$ of some finite field), $g \in E$ is the chosen generator used to construct public keys from secret keys, and $\text{IsOne} : \mathbb{F}^\times \to \{0,1\}$ is constant equal to 0 except for $\text{IsOne}(1) = 1$. Thus a contribution is valid if and only if it is the product of the individual signatures on m (hashed onto the curve as $H(m) \in E$) of the public keys represented in the bitset $[\epsilon_1, \dots, \epsilon_N]$.

The domain of the (partial) aggregation function $Aggr$ consists of all pairs of contributions $c = (\sigma, [\epsilon_1, \dots, \epsilon_N])$ and $c' = (\sigma', [\epsilon'_1, \dots, \epsilon'_N])$ whose bitsets are "disjoint" (i.e. for all i, $\epsilon_i \epsilon'_i = 0$), and we set

$$Aggr(c, c') = (\sigma\sigma', [\epsilon_1 + \epsilon'_1, \dots, \epsilon_N + \epsilon'_N])$$

Aggregate contributions are precisely those contributions whose bitset contains at least two nonzero bits, while individual contributions are those contributions whose bitset contains precisely one nonzero bit. We define the weight of a contribution $c = (\sigma, [\epsilon_1, \dots, \epsilon_N])$ to be $w(c) = \epsilon_1 + \dots + \epsilon_N \in]0, N[$.

2.2 System Model

We assume there are N participants in the system. Each participant i carries a unique identifier $id_i \in [0, N[$. Moreover, we assume each participant knows the identifier of all other nodes participating in a given round of the protocol. Attackers may choose their respective identifiers in advance.

Multi-signatures. In the context of multi-signatures, each node has a private key sk_i and the corresponding public key pk_i, and the public keys of the participants are known to the participants ahead of time. We assume that m, the message to sign, is known by all participants beforehand as well.

2.3 Threat model

Handel makes several assumptions about the participants and the network:

- **Network:** We assume a *partially synchronous* network as defined in [19] or [17][1]: a fixed (but unknown) bound on the network latency is assumed. In this model, an attacker can read and delay any message (up to a maximum delay) before it reaches its destination [21].
- **Authentication:** We assume point-to-point authenticated channels between each pair of participating nodes.

[1] [17] uses the term *semi-synchronous*

- **Adversarial model:** We assume a static fraction of the participating nodes to be Byzantine. Byzantine nodes may behave arbitrarily during the execution of the protocol and may coordinate amongst themselves. Byzantine nodes are bound to polynomial-time computation, eg. cannot break the cryptography primitives commonly used.
- **Unforgeability of valid contributions:** We ask that the only way to produce valid contributions be by aggregating *distinct* valid individual contributions, and that two such aggregate contributions be equal if and only if they are assembled from the same individual contributions.

2.4 Goals

Handel aims to be a large scale aggregation protocol that outputs an aggregate contribution including *at least* a predefined threshold $T \leq N$ number of contributions. Handel guarantees the following:

- **Termination:** Handel finishes with an aggregate contribution including at least a configuration-defined threshold number of individual contributions.
- **Fairness:** The final aggregate contributions *eventually* contain all honest contributions.
- **Time efficiency:** Handel's completion time scales logarithmically with the number of nodes.
- **Resource efficiency:** Handel's CPU and bandwidth consumption scale poly-logarithmically with the number of nodes.

Handel is not a consensus protocol and does not guarantee uniformity of the results. Indeed, the aggregation of T (different) contributions may result in different *valid* aggregate contributions. Moreover, a node running Handel can output multiple final aggregate contributions, each of them containing more individual contributions than the preceding aggregation. Handel leaves to the application the responsibility of managing these different aggregate contributions.

3 Handel Protocol

This section introduces the techniques used by Handel, then details how Handel works. For those techniques involving parameter values, we explain how we came to choose those values.

3.1 Overview of the techniques

Handel combines multiple techniques to reach its goals. To achieve speed, aggregation is parallelized as nodes organize themselves in a binary tree: this leads to a $O(log(N))$ time complexity. A binary tree with nodes occupying all positions from root to leaves (reflecting hierarchical relationships between nodes) is not fault tolerant: failure of the root node is fatal, failure of intermediate nodes can be very damaging. Thus Handel, much like San Fermín, uses a *tree based overlay network* where nodes are connected to all other nodes. These connections are structured in levels. At the first level, every node n has one level 1 peer n'. That relationship is symmetrical : n' has n as its level 1 peer. At the second level, each such pair of nodes n, n' has two peers m, m', themselves being level 1 peers. Again, this is symmetrical. These four nodes n, n', m, m' have four peers at level 3 and so forth.

Because nodes may be down or slow, all levels are executed concurrently and all nodes optimistically send their current best contributions, even if their previous level is incomplete. A node will often receive multiple contributions for the same level, and update its own aggregate contribution accordingly. This brings fail-silent fault tolerance, but floods the network.

For this reason, nodes engage in *periodic dissemination*. They do not contact all their peers simultaneously, but rather one by one, periodically and in parallel for *active* levels. New levels are activated every 50ms.

In a Byzantine context, nodes may receive invalid contributions. Contributions should thus be verified before they are aggregated. However, verification can be costly (a few milliseconds [33] for BLS signatures). Moreover, in heterogeneous environments (with some nodes faster than others, and/or a wide range of latencies), some nodes may receive more contributions than they can verify. Furthermore, since nodes may receive multiple contributions from peers in a given level set, some of these contributions may prove redundant or outdated. The ver-

ification of contributions must thus be triaged. For this reason, nodes *score* incoming contributions before verifying them and prune redundant ones. This brings resource efficiency—redundant contributions are not verified—and versatility—slow nodes need not verify all contributions.

However, this creates an attack point: Byzantine nodes could flood the network with high scoring, yet invalid contributions, and thus waste the recipients' CPU resources. In the worse case, honest nodes may find themselves wasting most of their resources verifying invalid contributions. Every Handel node thus prioritizes contributions by means of a *local ranking* of its peers with a *variable window size*. The ranking and score determine the subset of received contributions to be evaluated: simply score the contributions from nodes in the current window of ranked peers. Window size is dynamic: when a verification fails, the window size decreases; when a verification succeeds, the window size increases.

We now describe some mechanisms pertaining to messaging. Messages must include the sender's *individual contribution* alongside the relevant aggregate one. This allows its peers to aggregate individual contributions even if the aggregate contribution is not *aggregatable* with its local aggregation contribution. Next, when a node has a full aggregate contribution of a level, it immediately sends it to multiple nodes in the relevant peer set (*fast path*) instead of relying on periodic dissemination. Complete aggregate contributions are very valuable : if a node possesses one for some level, it does not need to verify any of the contributions from that (or lower) level(s).

3.2 Tree based overlay network

Handel organizes a node's sequence of aggregation steps using a binary tree. For instance, in figures 1 and 2, both n_4 and n_5 aggregate contributions c_4 and c_5 at level 1, and $n_4, \ldots, n_7$ aggregate $c_4, \ldots, c_7$ at level 2.

Formally, every node i defines a partition $C_0^i \sqcup C_1^i \sqcup \cdots \sqcup C_L^i$ of the set of nodes in the network where $\forall l \in [0\ ..\ L]$, $C_l^i = \{$nodes sharing a common prefix of length $L-l$ with node $i\}$ (and thus $C_0^i = \{i\}$). We call the set of peers of a node for a given level its *peer set*. We call C_l^i node i's **level l peer set**. See figure 1 showing how node 5 partitions the set of nodes.

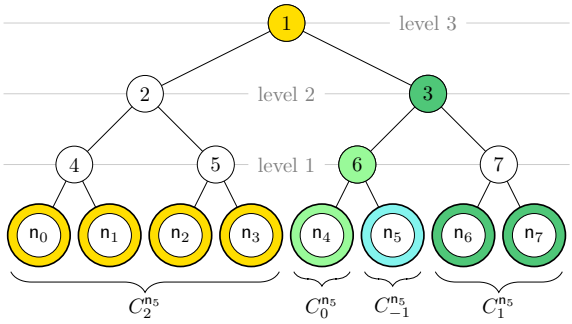

Figure 1: n_5's view of the network.

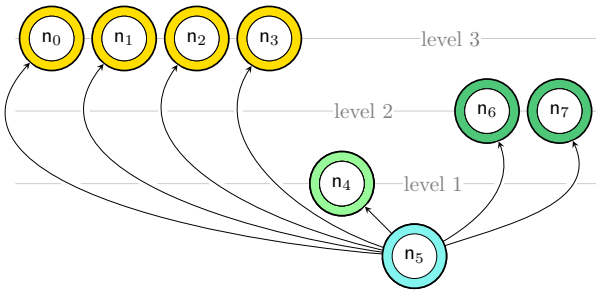

Figure 2: n_5 communication organization.

3.3 Peer ranking

Peer ranking allows nodes to organize communication within a level's peer set. Initially, all nodes are publicly shuffled (see section 4.3). Every node i ranks its level l peers s by tagging them with a Verification Priority (VP) $1 \leq \text{VP}_i(s) \leq 2^l$. Contributions received by i are prioritized according to s' (its sender) $\text{VP}_i(s)$. Of course, node i is similarly ranked by its level l peers. To maximize the likelihood of having its level l contribution considered, it computes its level l Contribution Valorization Vector (CVV) $[\text{VP}_{s_1}(i), \ldots, \text{VP}_{s_{2^l}}(i)]$ where the s_j are their level l peers, and contacts the level l peers that rank it highest first.

The VP functions must be computable from public data and act as pseudo random permutation

$\text{VP}_i(j) = rand(N, seed; i, j) \in [0, N[$ of its last argument j, where *seed* is a seed value.

3.4 Node state

For every level l, node i creates and maintains In_l^i, the best contribution node i could assemble from incoming level l contributions, and Out_l^i, the best contribution it can send to its level l peers. In_0^i is the node's individual contribution. For all l, $Out_l^i = Aggr(In_0^i, In_1^i, ... In_{l-1}^i)$. Out_L^i is the final aggregation, the best aggregation built so far. We call Out_l^i (resp. In_l^i), and level l by extension, *complete* if it includes all the contributions from levels $\leq l$ (resp. $= l$).

On top of the aggregate contribution vectors $(In_l^i)_l$ and $(Out_l^i)_l$, node i internally maintains the following information for all levels l: (1) its level l CVV (2) the list of as-of-yet unverified received level l contributions Inc_l^i, (3) the start time of the level $StartTime_l$.

3.5 Messages in Handel

There is only one type of message in Handel. Messages contain the following data: the *level* l, the *sender's ID* i, an *aggregate contribution* Out_l^i and the sender's *individual contribution* In_0^i

There are two circumstances that trigger messages to be sent: **periodic dissemination** and **fast path**. Periodic dissemination is triggered after every elapsed **dissemination period** (DP) (eg. 20 milliseconds). A node will send a message to one node from every one of its *active levels*. A level l is *active* if Out_l is complete or if $StartTime_l$ has elapsed. The completion of a level triggers the fast path mode in which a node sends Out_l to small number of peers (eg. 10) at this level l.

3.6 Receiving Contributions

When node i receives a message from node j at level l, it includes the aggregate contribution Out_l^j and individual contribution In_0^j in its list of as-of-yet unverified level l contributions. If there is already a contribution from this node, it keeps the one with greater *weight*. The contribution vector is thus bounded in size.

Node i continuously evaluates this list to pick the next contribution to verify (and prune redundant contributions). This selection process involves the *windowing* and *scoring* functions described below. If the chosen level l contribution passes verification, Handel updates its local best contribution In_l^i and recomputes all of $Out_{l+1}^i, \ldots, Out_{l_{\max}}^i$. The rank of the contribution's sender is increased by the peers size.

3.7 Pruning

Handel **prunes** (1) all contributions from nodes identified as Byzantine, eg. which previously sent an invalid contribution, (2) all contributions from levels where In_l is complete, (3) all aggregate contributions that cannot be aggregate with, or have a lower weigth than In_l^i.

3.8 Windowing and scoring

Handel determines the order in which incoming contributions are verified in terms of their VP, their score and the windowing.

The **score** of a level l contribution c is, if (c, In_l^i) is aggregatable, the weight of the aggregate, otherwise, the weight of c aggregated with all previously verified and aggregatable individual (level l) contributions.

Windows allow to select contributions by determining the smallest VP r in the incoming contributions and selecting the ones with a rank inferior to $r + windowSize$. These contributions are scored, and the highest scoring one is verified first. The window's size is dynamic and changes exponentially depending on the result of the verification: on success the size is multiplied by 2. On failure the size is divided by 4.

3.9 Parameters value

We explained in 3.1 the rationale behind Handel's key mechanisms. This section explains the parameter choices in light of associated tradeoffs. We designed Handel to perform optimally in a large-scale WAN network and to require no further configuration regardless of node capacity and connectivity. To determine the optimal values of these parameters, we ran multiple simulations with various network configurations, described in the appendix A.

We chose a **dissemination period** (DP) of 20ms. This is compatible with verification times in the milliseconds (it leaves enough time to verify contribu-

tions between DPs) and typical latencies in a WAN (≈100ms). Experimental results show this is a good tradeoff between network bandwith and time to completion.

The number of messages sent in **fast path** is set to 10. The tradeoff is between completion time bandwidth consumption. Fast path is especially important when most nodes are available, as only completed levels can trigger it. Fast path incurs no extra cost if it is never triggered.

Level l start time is set to $(l-1) \times 50$ms. Starting levels incrementally saves an important number of messages when the threshold is reached quickly. For a timeout of 50ms, in A.1, we observed 20% less messages for a comparable time to completion.

Windowing preserves the benefits of scoring (verify only important contributions) while ranking senders (which limits the effects of DoS attacks). It is important that window size decrease quickly when under attack, so that Handel can quickly return to the best strategy of using only ranking. Similarly, window size must grow quickly under normal conditions, so that nodes exploit incoming contributions to the fullest. Window size can expand and contract exponentially. The **expansion factor** is 2, while the **contraction factor** is 4. These value perform well under multiple attack scenarios.

4 Analysis

4.1 Complexity

Consider the case when there are no Byzantine nodes. In this case:

1. Time complexity is $O(log(N))$, as there are $log(N)$ levels ([12]).
2. Message complexity is $O(polylog(N))$ per node: *periodic dissemination* accounts for less than $log(N)$ messages sent per period. With $O(log(N))$ DPs, $O(log(N)^2)$ messages are sent this way. *Fast path* can be triggered once per level, and involves sending a fixed number of messages (10 in our implementation), and thus adds $O(log(N))$ messages. Message complexity is thus $O(log^2(N))$.
3. CPU consumption is less than message complexity: not all messages received get verified. In the worst case, all received messages are verified and CPU complexity is $O(polylog(N))$.

With a large fraction of byzantine nodes, time complexity becomes $O(N)$: after N periods, every honest node will have contacted all other honest nodes, as Byzantine nodes cannot prevent honest nodes from sending their individual contributions.

With a small fraction of Byzantine nodes, time complexity is $O(log(N))$. Byzantine nodes can (1) refrain from participating, (2) send invalid contributions with a high score (with the idea of wasting the target's CPU cycles to prevent honest contributions from ever being examined and aggregated), (3) send very small but valid contributions, again with the idea of preventing larger contributions from honest nodes to be integrated, (4) A combination of the above. The purpose of the score function is to render the attack (3) obsolete: nodes will verify the large contributions first. (2) causes the window size to be reduced to 1, eg. a node will evaluate all the contributions received. Because of the ranking, Byzantine nodes cannot prevent the valid contributions to then be verified, hence the protocol will progress. (4) Once the window's size has been reduced to 1, there is no difference between a valid and an invalid contribution: all of them will be verified anyway, so (4) is equivalent to (3).

4.2 Versatility of Handel

4.2.1 Heterogeneous latencies

Consider the following simple scenario : (1) 80% of the nodes, call them *fast nodes*, have latency 5 times the dissemination period (DP) (2) 20% of the nodes, call them *slow nodes*, have latency 20 times the DP (3) all nodes have fast CPUs and can verify all incoming signatures (4) the gap between two consecutive $StartTime$ is twice the DP (5) there are 12 levels, i.e. 4096 nodes. For such a configuration phenomena occur: firstly, the fast nodes progress independently of the slow nodes. Slow nodes can be seen as not participating and are similar to fail-silent nodes from the fast node's point of view. The first consequence, the complexity between fast nodes is $O(log(N))$ as with

a bounded limit of Byzantine nodes. The second consequence, if a slow node can send its contribution to a fast one, its contribution will become widely available among the fast nodes.

Secondly, for a slow node the DP is negligible relative to its latency: the expected time for its contribution to reach a fast node is, with ρ the ratio of fast nodes, and λ the latency: $\sum_{k=0}^{\infty}(\lambda + k \cdot \text{DP})\rho(1-\rho)^k$. In our example with 20% of slow nodes, ($\lambda = 20\text{DP}$), a slow node will reach a fast one on average in $20.25\text{DP} \approx \lambda$.

Thirdly, periodical dissemination ensures that a signature is well distributed. As at each period all nodes send their aggregate contributions to one peer per level, hence the number of nodes receiving the slow node's individual contribution grows polynomially with elapsed period. In other words, if we consider the period time as negligible against the latency, the time to get the final aggregation is $2 \cdot \textit{slow latency} + \textit{fast latency} \cdot log(N)$ for slow nodes, and $\textit{slow latency} + \textit{fast latency} \cdot log(N)$ for fast ones.

4.2.2 Heterogeneous CPU capacities

If all nodes have similar latencies, but very different CPU power, the aggregate contributions from the weak nodes have little value, score poorly with other nodes and are mostly ignored. Indeed, the weak nodes check fewer contributions, but prioritize interesting contributions because of the score. As for latencies, weak nodes' individual contributions are included very quickly by the fast nodes.

4.2.3 Heterogeneous clocks

There are two synchronization points in Handel: (1) the dissemination period (DP) (2) the time when the protocol starts. DPs may vary from node to node, a situation similar to that of heterogeneous network latencies. Some nodes can start the protocol later than others. This is similar to having a large latency, but only at the beginning of the protocol. In other words, heteroeneous clocks will have less impacts than heteregeneous latencies, hence late nodes will quickly catch up on account of them immediately receiving weighty contributions from the nodes that started early.

4.3 Global shuffling of the IDs

In this section, we argue for a global shuffle of the IDs using an unpredictable seed prior to each Handel round. Indeed, two attacks exploiting the VP are foiled this way : (1) an attacker may try to prevent an honest node's contributions from reaching any other honest node during the first d DPs or (2) an attacker may try to prevent an honest node from verifying honest contributions in the first d DPs.

The second attack requires that the attacker fill up the first d VP slots of its target. If there is no global shuffle, and since the attacker may freely choose its IDs, the attacker can fill up the first $d \leq f$ VP slots of a target node, where f is the number nodes controlled by the attacker.

However, if the list of IDs is randomly shuffled using an unpredictable seed before every round of the protocol, that probability becomes f/N. The probability that an honest node's first d VP slots contain an honest node is $1 - \frac{f}{N} \cdot \frac{f-1}{N-1} \cdots \frac{f-d+1}{N-d+1} > 1 - (\frac{f}{N})^d$. Thus, with $d \geq 7$ and $f/N < ½$, an honest node has a 0.99 probability of having at least one honest node in its top $d = 7$ VP slots. Similarly, a node's level l CVV is computed deterministically from the VP functions of its level l peers and therefore the probability of having d malicious nodes occupy the first d slots is $\frac{f(f-1)\cdots(f-d+1)}{N^d} < \frac{f^d}{N^d}$. There are multiple techniques in the litterature to generate unpredictable random coins, in particular using blockchain data [8, 3, 1].

4.4 Security Analysis

This section provides a security analysis of Handel. We prove that Handel is secure against Byzantine actors by showing that all honest nodes *eventually* output an aggregate contribution containing *at least* a threshold of honest nodes' contributions, and *eventually* all honest nodes' contributions.

To prove Handel is secure, we want to prove properties similar to the ones of a consistent broadcast [10], closely related to Handel. This analysis is tailored to Handel's usage in the context of multi-signatures that are secure against existential forgery under known-message attacks [24]. We argue that this is a realistic assumption as Handel assumes a publicly known mes-

sage before the aggregation protocol starts. The security analysis assumes the network and threat model described in section 2. We make the assumption that the individual contributions received by a node are stored until the end of the protocol. We say that an honest node *diffuses* its individual contribution when it is sent to all other nodes. We say that a node *includes* a contribution when it is included in its final aggregate contribution. In order to stay consistent with the notation in this paper, we use the *honest* node notation instead of the *correct* node notation.

Lemma 1 (Validity)**.** *If an honest node diffuses its individual contribution, then all other honest nodes eventually include it.*

Proof. Because of our **network assumptions** (partial synchrony and bounded intentional delays, see 2.3), all honest nodes eventually receive (and verify) any individual contribution that was diffused. Since channels are authenticated, honest individual contributions are valid and all other nodes eventually include them. □

Lemma 2 (Integrity)**.** *Aggregate contributions assembled by honest nodes include individual contributions at most once. Moreover, if some node's i individual contribution is accounted for in a valid contribution, then i must have produced a valid contribution at some point.*

Proof. Both points readily follow from our **unforgeability of valid contributions** assumption, see 2.3. □

These two properties together ensure that an honest node's contribution eventually gets included in all other honest node's final contribution exactly once, thus proving fairness of Handel. Termination is immediate considering network assumptions and the inclusion of individual contributions in all messages. Finally, as long as the threshold parameter is lower than the number of honest nodes, the number of contributions in the final multi-signature eventually reaches the threshold parameter.

5 Deployment Scenarios

We expect Handel to be widely applicable in many different contexts. Its strengths and weaknesses, however, aren't uniform across all deployment scenarios. We focus on the privacy and DoS protection various Handel deployments provide in the context of signature aggregation.

Handel forces signers to reveal their IP address so they can receive Handel packets. In some cases, this public mapping between public key and IP addresses is problematic since it leaves signers vulnerable to DoS attacks and privacy breaches. For example, in proof-of-stake protocols signers might want to hide their IP addresses if they are linked with the public key controlling its staked crypto-currency funds. The first scenario we consider is the vanilla deployment of Handel, where the IP addresses are public. The second one explores using an anonymity network (such as Tor [18]) and its synergies with Handel. Thirdly, we describe a deployment scenario using ring signatures to provide anonymity within the set of participants. Finally, we end with a brief overview of two network protocols offering interesting trade-offs for Handel.

5.1 Vanilla Deployment

This deployment scenario is the one described in section 3. Every node is reachable from the Internet and each signer participates in the protocol. There is a public global directory containing the mapping between signers' public keys and their public addresses. This deployment is simple to understand and to administrate: drawbacks are clear, and are shared by many existing systems, making them widely understood, which is always an advantage when discussing actual security deployments. On the other hand, having signer's IP addresses be public exposes them to DoS attacks or even full compromise. However, one can put in place defenses against this class of attacks. In order to accommodate for a full compromise, a signer can install its signing key in an external HSM module [13]. Regarding DoS attacks, a single offline signer has little effect on the outcome of Handel. To make a node appear offline, a DoS attack must block *all* outgoing packets. But one packet reaching one

honest node is enough to ensure that node's individual contribution be included in a final aggregate contribution.

5.2 Anonymity Network

This section explores how synergies between anonymous network communication overlays and Handel drastically increase signers' anonymity, privacy and DoS-resistance guarantees. Consider how Handel may work on top of the anonymous communication network Tor [18] (alternatively I2P [28] or Freenet [14]). Signers could run hidden services on Tor reachable via a 56-character long address. A public global directory could store the mapping from public keys to Tor addresses. Participants would send their aggregate signatures to the other nodes' Tor addresses. Running Handel on an anonymous communication network exhibits the usual trilemma between *strong anonymity*, *bandwidth overhead* and *low latency* [16]. *Anonymity* guarantees are clearly far greater than in the vanilla deployment. In order to DoS a signer, an attacker would have to either attack the Tor network itself, or de-anonymize the signer. While the latter is possible for state-level adversaries, it is a nontrivial attack. Regarding the former, Tor has implemented multiple DoS protections [29] and the most effective DoS attack remains that against the whole Tor network. However, such anonymity network only offers the same guarantees as the underlying anonymous network offers. For example, against a passive global adversaries, traffic correlation attacks are possible against Tor users [31]. Compared to a UDP-based solution, *Latency* is one order of magnitude greater: on Tor, which is based on TCP, circuit building takes more than 500ms [37], whereas UDP messages reach their destination typically within 100-200ms [39]). Also, *bandwidth consumption* increases by an order of magnitude since participants first need to construct their circuit on the network.

One interesting point of this deployment model is its compatibility with other deployment models: some nodes could run behind an anonymity network while others run Handel's vanilla mode.

5.3 Anonymous Signers

Anonymity can be achieved if participants are allowed to sign messages using fresh secret/public key pairs. Using *linkable ring signatures* [35] individual public keys PK_i from a set of public keys $PK = \{PK_i\}_i$ can create new "anonymous" public keys AK_i in such a way that any attempt to create multiple AK_i from a single public key PK_i is apparent.

Using anonymous public keys in Handel requires a setup phase where every signer i produces a new anonymous private / public key pair ak_i / AK_i.

Setup Phase. Every signer locally creates a new private key ak_i and public key AK_i. It then broadcasts the message $IP\text{-}address_i \| AK_i$ with σ_i, a linkable ring signature for the set of signers PK of this message. At the end of the setup phase, the signers know the list of IP addresses of the other nodes and their new anonymous public keys AK_i. If an honest signer detects two signatures from the same signer (by *linkability*) the associated AK_i is discarded from AK. Using *traceable ring signatures* [20], the owner of PK_i can even be identified and removed from the public list of signers.

Handel then runs its course using this new set of public keys AK, without the mapping between signing keys and IP addresses.

The anonymity protection offered by this protocol becomes stronger as more signers adopt it, which is remarkable considering Handel's ability to scale. Importantly, the setup phase only has to be done once and can be reused for multiple Handel rounds.

6 Implementation

We released a reference implementation of Handel as an open source Golang library [5] under the Apache license. The library implements the protocol described in 3 in ∼3000 lines of code. It is able to accommodate different *aggregation schemes* (BLS with BN256, BN12-381, etc), *network topologies* (direct communication, different branch factor, etc), and *network layers* (UDP, QUIC, etc). We implemented BLS aggregation [7, 6] on the BN256 [38] curve (optimized implementation by Cloudflare [15]), with UDP as transport layer. Our implementation includes sev-

eral technical optimisations not mentioned in 3. In QUIC it is possible to know if a sent message was received. Therefore, when using QUIC, we do not send the same message twice to a peer. Nodes using UDP may include a flag in their messages to signal they already received a message, so nodes do not keep sending messages in the lower levels.

Isolated Verification. Handel thrives to reduce the number of cores dedicated to scoring and contribution verification. Since Handel is to be used alongside an application, Handel's CPU consumption must neither impact nor hinder the application's normal functioning. Our implementation thus verifies incoming contributions on a single CPU core.

Message Format. Messages in the reference implementation contain: the sender's ID (a 32 bit integer), the level (a byte), the aggregate contribution (as defined in 2.1) and the signature of the sender. A BN256 signature is 64 bytes. The bitset's size depends on the level ($size = 2^{level-1}$ bits). For 4000 nodes, the bitset size can go up to 250 bytes. The maximum total message size is thus 417 bytes.

Wire Protocol We used the gob encoding scheme [23] which adds a few linear extra bytes to the messages.

7 Evaluation

This section contains a discussion of our test results for multi-signature aggregation using Handel. Our focus is on Handel's performance and comparing it to other aggregation protocols. We observe logarithmic completion times and resource consumption, and observe how different parameters impact Handel's performance.

7.1 Experimental Setup

Measurements. All Handel nodes log measurements such as the number of received messages, the number of verifications, the number of messages sent and other internal statistics. For each experiment, we computed minima, maxima and averages of test data.
Network Topology. We ran large scale experiments of Handel on AWS EC2 instances. Our biggest tests involved 2000 t2.nano instances, each with on 3.3Ghz core. In most cases, we ran two Handel nodes per EC2 instance, sometimes incurring an over-subscription effect. Therefore, we expect real-world deployments on commodity hardware to exhibit even better performance than the ones measured in our tests. In order to simulate a real world deployment, and latency variability in particular, we used AWS instances from 10 AWS regions located all over the world. Compare figure B for the exact latency distribution between each region.

7.2 Comparative Baseline

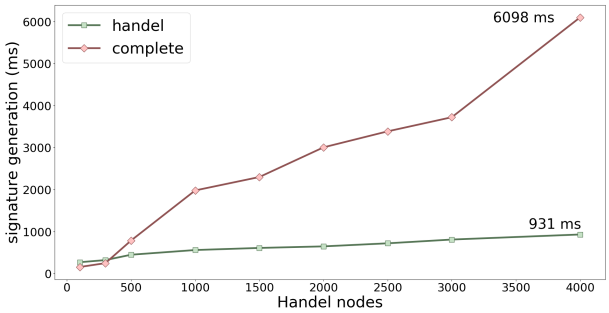

Figure 3: **Time** comparison up to 4000 nodes with a 99% threshold

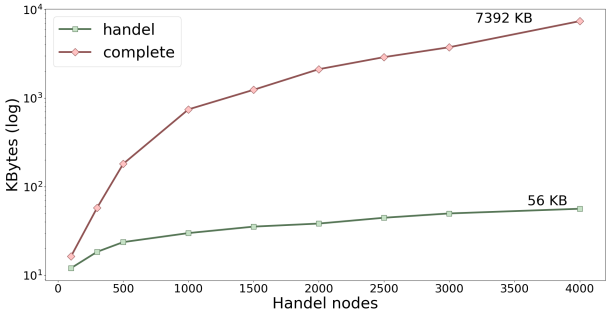

Figure 4: **Outgoing Network** comparison up to 4000 nodes with a 99% threshold

We compare Handel against a "complete graph" aggregation scenario. In the complete graph scenario, nodes send their individual signatures to all other

nodes. Since Handel eventually leads to this scenario (see 4.4) we want to compare both approaches. In both scenarios, completion means aggregating 99% of all individual signatures. See Figure 3 for a comparison of completion times and Figure 4 for a comparison of the outgoing network consumption.

We observe that, on average, Handel is able to aggregate 4000 signatures in under a second. Futhermore, Handel is both orders of magnitude faster and more resource efficient than the complete graph approach. Finally, Handel's logarithmic time complexity is apparent from Figure 3.

7.3 Robustness of Handel

In these experiments, we test Handel's resistance to node failure. Figure 5 illustrates Handel's ability to aggregate 51% of signatures from 4000 nodes at different rates of node failure.

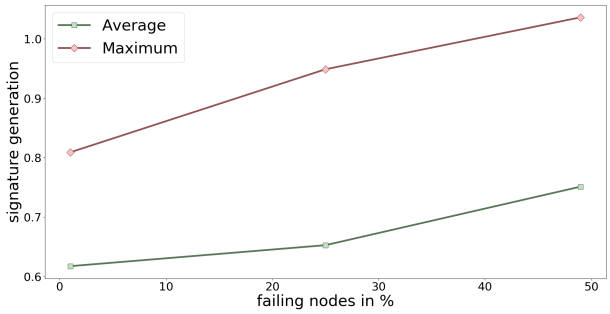

Figure 5: Various percentages of failing nodes over a total of 4000 nodes for a 51% threshold

7.4 CPU consumption

This experiment is designed to test Handel's CPU consumption under heavy load. In Handel, signature verification is the main CPU bottleneck. Figure 6 shows the *{minimum,maximum,average}* number of signatures checked for different node counts. All curves exhibit the expected logarithmic behaviour. In particular, the minimum curve shows that some nodes verified the smallest possible number of signature for 4000 nodes: around $30 \approx 24 = 2 \cdot log_2(4000)$ (every message contains two signatures). The graph shows that some nodes needed to verify *many more* signatures in order to create the final multi-signature. This is due to the high latency that some pairs of nodes (nodes in different regions). In this case, these nodes have to wait *longer* to compute their final multi-signature. In the meantime, these nodes verify all other incoming signatures, even though these contributions may have a low score, thereby unduly increasing the number of signatures checked.

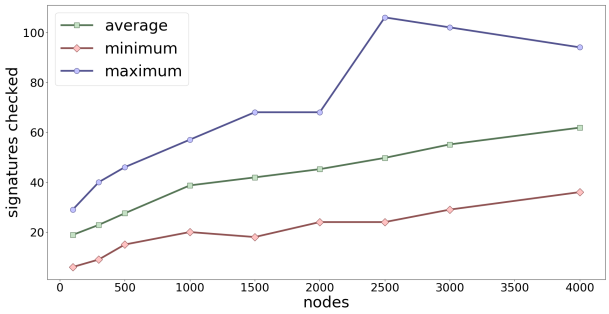

Figure 6: Number of signatures checked on a node up to 4000 nodes with a 99% threshold

8 Related Work

The growing need for scalable distributed data aggregation systems has spawned a number of interesting designs in the literature. These systems are primarily designed for *permissionned networks* such as grid networks or monitoring networks.

8.1 San Fermín

San Fermín [12] introduced the *binomial swap forest* technique enabling a $O(log(N))$ time complexity for its aggregation protocol. Handel is based on this technique. San Fermín has two important limitations that Handel overcomes.

Timeout sensitivity. San Fermín is defined in the *fail-silent* model where a node can fail and stop responding. In San Fermín, failure to send one's contribution within a limited amount of time is interpreted as node failure. Hence, timeouts are a key parameter of the system: if the timeout is too short, nodes might be evicted too early; if the timeout is

too long, the protocol's completion time increases linearly.

Byzantine actors. San Fermín is vulnerable to Byzantine attacks. An adversary can do an *eclipse attacks*, where it prevents a contribution from being included in the final aggregate. To achieve this, it is enough to contact the other nodes very early on, making them move to the next levels without having completed previous levels.

8.2 Gossip-based solutions

Systems such as [26, 32, 30] are gossip-based solutions that converge exponentially for simple aggregation functions such as sum and average. However, these systems do not protect against Byzantine behaviors and are designed for synchronous networks models which we argue is impractical for large scale deployments over the Internet. Moreover these systems exhibit an inherent trade-off between scalability and precision rendering them harder to control in a permissionless setting.

8.3 Tree-based solutions

Tree based constructions can help address scalability. Astrolab [42] is the first system using hierarchy zones mimicking the design of DNS zones. Systems such as SDIMS [44], CONE [4], Willow [43] combine hierarchy-based constructions with the design principle underlying *distributed hash table* (DHT) systems. Li's system [34] has the advantage of being compatible with any DHT abstraction instead of requiring changes to the underlying protocols. However, it uses one root node per aggregation function hence offers poor resistance against failures. Finally, Grumbach et al. [25] use a mechanism similar to binomial swap forests with the native underlying trees of the Kademlia DHT [36]. In particular, it is the first system tackling aggregation with *some* Byzantine nodes. However, their scheme requires interactions between multiple nodes in a subtree to validate an aggregate contribution.

9 Conclusions

We presented Handel, an scalable byzantine resistant aggregation protocol with logarithmic time and resource complexity. Large scale test of our Golang Handel library show it can aggregate 4000 signatures in under a second.

Multi-signature schemes have garnered a lot of attention lately, yet data aggregation frameworks at scale have only been examined under the fail-silent model, and no multi-signature aggregation frameworks have been developed to withstand Byzantine scenarios. We argue it's because Byzantine nodes could follow the protocol but provide invalid contributions. When applied to cryptographic schemes such as BLS signatures, a Byzantine fault tolerant protocol becomes useful. With the rise of proof of computation schemes (zk-snarks, zk-starks), we expect fast byzantine fault tolerant aggregation schemes to become increasingly important. Moreover Handel's versatility makes it useful even in non Byzantine scenarios.

Acknowledgements

We thank Justin Cappos for his helpful insights, Vanessa Bridge, Ben Edgington and Shahan Katchadourian for their comments and suggestions and Nadim Haveric for sharing his expertise on AWS.

References

[1] Baruch Awerbuch and Christian Scheideler. Robust random number generation for peer-to-peer systems. In *International Conference On Principles Of Distributed Systems*, pages 275–289. Springer, 2006.

[2] Paulo SLM Barreto and Michael Naehrig. Pairing-friendly elliptic curves of prime order. In *International Workshop on Selected Areas in Cryptography*, pages 319–331. Springer, 2005.

[3] Iddo Bentov, Ariel Gabizon, and David Zuckerman. Bitcoin beacon. *CoRR*, abs/1605.04559, 2016.

[4] Ranjita Bhagwan, , and Geoffrey M. Voelker. Cone: Augmenting dhts to support distributed resource discovery. July 2003. POPL '01 Proceedings of the 28th ACM SIGPLAN-SIGACT symposium on Principles of programming languages.

[5] Nicolas Gailly Blazej Kolad, Nicolas Liochon. Handel implementation in go. https://github.com/ConsenSys/handel, 2019.

[6] Dan Boneh, Manu Drijvers, and Gregory Neven. Bls multi-signatures with public-key aggregation, 2018.

[7] Dan Boneh, Ben Lynn, and Hovav Shacham. Short signatures from the weil pairing. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 514–532. Springer, 2001.

[8] Joseph Bonneau, Jeremy Clark, and Steven Goldfeder. On bitcoin as a public randomness source.

[9] Ethan Buchman. *Tendermint: Byzantine fault tolerance in the age of blockchains*. PhD thesis, 2016.

[10] Christian Cachin. Byzantine broadcasts and randomized consensus. https://lpd.epfl.ch/site/_media/education/sdc_byzconsensus.pdf, 2009.

[11] Ran Canetti, Ivan Damgaard, Stefan Dziembowski, Yuval Ishai, and Tal Malkin. On adaptive vs. non-adaptive security of multiparty protocols. In *International Conference on the Theory and Applications of Cryptographic Techniques*, pages 262–279. Springer, 2001.

[12] Justin Cappos and John H Hartman. San fermın: aggregating large data sets using a binomial swap forest.

[13] Francisco Cifuentes, Alejandro Hevia, Francisco Montoto, Tomás Barros, Victor Ramiro, and Javier Bustos-Jiménez. Poor man's hardware security module (pmhsm): A threshold cryptographic backend for dnssec. In *Proceedings of the 9th Latin America Networking Conference*, pages 59–64. ACM, 2016.

[14] Ian Clarke et al. *A distributed decentralised information storage and retrieval system*. PhD thesis.

[15] Cloudflare. Bn256 cloudflare implementation. https://github.com/cloudflare/bn256, 2018.

[16] Debajyoti Das, Sebastian Meiser, Esfandiar Mohammadi, and Aniket Kate. Anonymity trilemma: Strong anonymity, low bandwidth overhead, low latency-choose two. In *2018 IEEE Symposium on Security and Privacy (SP)*, pages 108–126. IEEE, 2018.

[17] Bernardo David, Peter Gaži, Aggelos Kiayias, and Alexander Russell. Ouroboros praos: An adaptively-secure, semi-synchronous proof-of-stake blockchain. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 66–98. Springer, 2018.

[18] Roger Dingledine, Nick Mathewson, and Paul Syverson. Tor: The second-generation onion router.

[19] Cynthia Dwork, Nancy Lynch, and Larry Stockmeyer. Consensus in the presence of partial synchrony. *Journal of the ACM (JACM)*, 35(2):288–323, 1988.

[20] Eiichiro Fujisaki and Koutarou Suzuki. Traceable ring signature. *IEICE transactions on fundamentals of electronics, communications and computer sciences*, 91(1):83–93, 2008.

[21] Juan Garay, Aggelos Kiayias, and Nikos Leonardos. The bitcoin backbone protocol: Analysis and applications. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 281–310. Springer, 2015.

[22] Yossi Gilad, Rotem Hemo, Silvio Micali, Georgios Vlachos, and Nickolai Zeldovich. Algorand: Scaling byzantine agreements for cryptocurrencies. In *Proceedings of the 26th Symposium on Operating Systems Principles*, pages 51–68. ACM, 2017.

[23] Golang. Gob encoding. https://golang.org/pkg/encoding/gob/, 2014.

[24] Shafi Goldwasser, Silvio Micali, and Ronald L Rivest. A digital signature scheme secure against adaptive chosen-message attacks. *SIAM Journal on Computing*, 17(2):281–308, 1988.

[25] Stéphane Grumbach and Robert Riemann. Secure and trustable distributed aggregation based on kademlia. *CoRR*, abs/1709.03265, 2017.

[26] Indranil Gupta, Robbert Van Renesse, and Kenneth P Birman. Scalable fault-tolerant aggregation in large process groups. In *2001 International Conference on Dependable Systems and Networks*, pages 433–442. IEEE, 2001.

[27] Timo Hanke, Mahnush Movahedi, and Dominic Williams. Dfinity technology overview series, consensus system. *arXiv preprint arXiv:1805.04548*, 2018.

[28] I2P. Invisible internet project. 2000.

[29] Rob Jansen. New tor denial of service attacks and defenses. https://blog.torproject.org/new-tor-denial-service-attacks-and-defenses, 2014.

[30] Márk Jelasity, Alberto Montresor, and Ozalp Babaoglu. Gossip-based aggregation in large dynamic networks. *ACM Transactions on Computer Systems (TOCS)*, 23(3):219–252, 2005.

[31] Aaron Johnson, Chris Wacek, Rob Jansen, Micah Sherr, and Paul Syverson. Users get routed: Traffic correlation on tor by realistic adversaries. In *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*, pages 337–348. ACM, 2013.

[32] David Kempe, Alin Dobra, and Johannes Gehrke. Gossip-based computation of aggregate information. In *44th Annual IEEE Symposium on Foundations of Computer Science, 2003. Proceedings.*, pages 482–491. IEEE, 2003.

[33] Blazej Kolad. Benchmarking of milagro bls implementation. 2018.

[34] Ji Li, Karen Sollins, and Dah-Yoh Lim. Implementing aggregation and broadcast over distributed hash tables. *ACM SIGCOMM Computer Communication Review*, 35(1):81–92, 2005.

[35] Joseph K Liu and Duncan S Wong. Linkable ring signatures: Security models and new schemes. In *International Conference on Computational Science and Its Applications*, pages 614–623. Springer, 2005.

[36] Petar Maymounkov and David Mazieres. Kademlia: A peer-to-peer information system based on the xor metric. In *International Workshop on Peer-to-Peer Systems*, pages 53–65. Springer, 2002.

[37] Tor Metrics. https://metrics.torproject.org/torperf.html, 2018.

[38] Michael Naehrig, Ruben Niederhagen, and Peter Schwabe. New software speed records for cryptographic pairings. In *International Conference on Cryptology and Information Security in Latin America*, pages 109–123. Springer, 2010.

[39] Wonder Network. Global ping statistics. https://wondernetwork.com/pings, 2019.

[40] Vanessa Bridge Nicolas Liochon Nicolas Gailly, Blazej Kolad. Wittgenstein: protocol simulator. https://github.com/ConsenSys/wittgenstein, 2018.

[41] Ronald L Rivest, Adi Shamir, and Yael Tauman. How to leak a secret. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 552–565. Springer, 2001.

[42] Robbert Van Renesse, Kenneth P Birman, and Werner Vogels. Astrolabe: A robust and scalable technology for distributed system monitoring, management, and data mining. *ACM transactions on computer systems (TOCS)*, 21(2):164–206, 2003.

[43] Robbert Van Renesse and Adrian Bozdog. Willow: Dht, aggregation, and publish/subscribe in one protocol. In *International Workshop on Peer-to-Peer Systems*, pages 173–183. Springer, 2004.

[44] Praveen Yalagandula. Sdims: A scalable distributed information management system. 2004.

Appendices

A Simulation run

We present here the results of the simulations we ran to determine Handel's parameter before running our large-scale evaluation in section 7. We used the Wittgenstein simulator [40] to run the simulations. Simulated environment is (1) nodes are deployed either on AWS datacenters, reusing the number we presented in 7.1, or in 242 different cities for which the latency is known [5] (2) Verification time is 4ms on a standard nodes, but nodes can be up to 3 times slower or up to 3 times faster. (3) nodes can be down: we looked at two different scenarios: 1% of the nodes are down or 20% of them are down (4) nodes can be on a Tor (eg. have an extra latency of 1000ms round-trip) network. We measure the average time for the honest nodes to reach a threshold of 99% of the honest nodes. The numbers presented are the average of 5 runs.

A.1 Cities Measurements

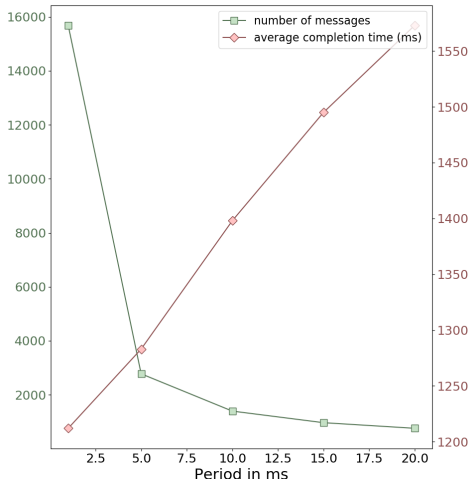

Figure 7: **Period.** Comparison of different period time. The left y-axis shows the *average* number of messages received per node. The right y-axis shows the average completion time of Handel.

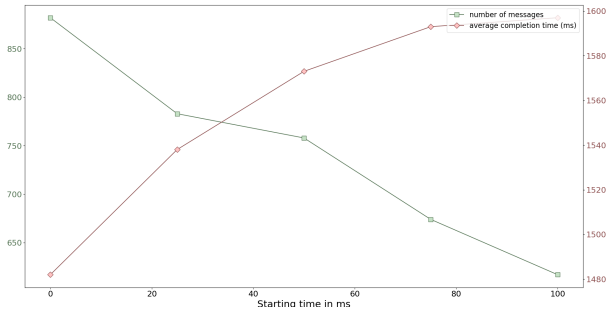

Figure 8: **Start Time.** Comparison of different starting time for the levels. A start time of x means the level i starts at time $x * i$. The left y-axis shows the *average* number of messages received per node. The right y-axis shows the average completion time of Handel.

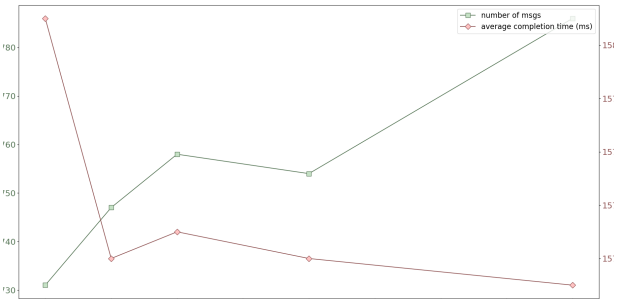

Figure 9: **Fast Path.** Comparison of different values for the fast path mechanism. A fast path of x means a node contacts simultaneously x node when a level is completed. The left y-axis shows the *average* number of messages received per node. The right y-axis shows the average completion time of Handel.

A.2 AWS Measurements

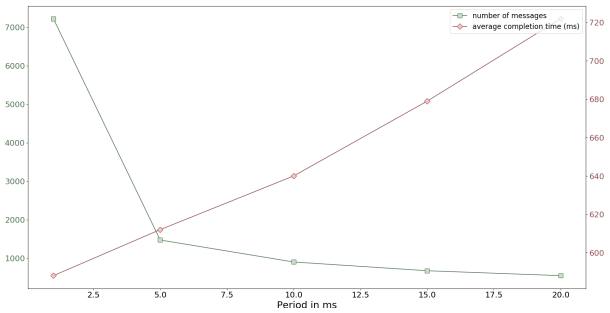

Figure 10: **Period AWS.** Comparison of different period time. The left y-axis shows the *average* number of messages received per node. The right y-axis shows the average completion time of Handel.

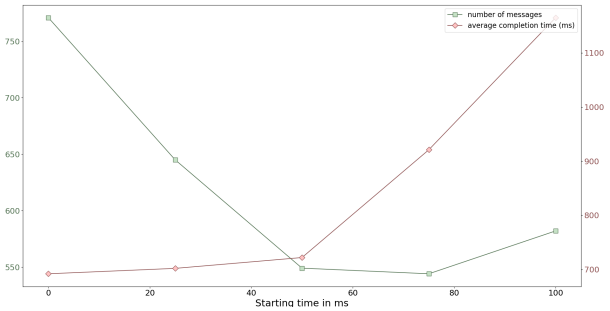

Figure 11: **Start Time AWS.** Comparison of different starting time for the levels. A start time of x means the level i starts at time $x * i$. The left y-axis shows the *average* number of messages received per node. The right y-axis shows the average completion time of Handel.

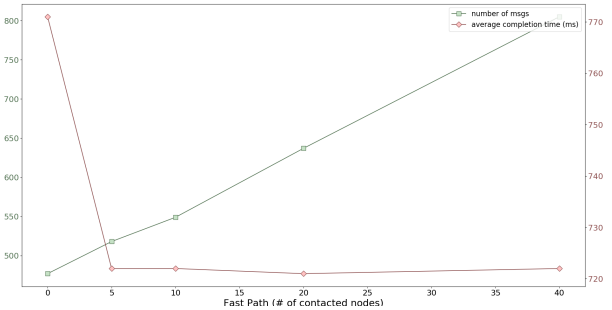

Figure 12: **Fast Path AWS.** Comparison of different values for the fast path mechanism. A fast path of x means a node contacts simultaneously x node when a level is completed. The left y-axis shows the *average* number of messages received per node. The right y-axis shows the average completion time of Handel.

A.3 Tor Measurements

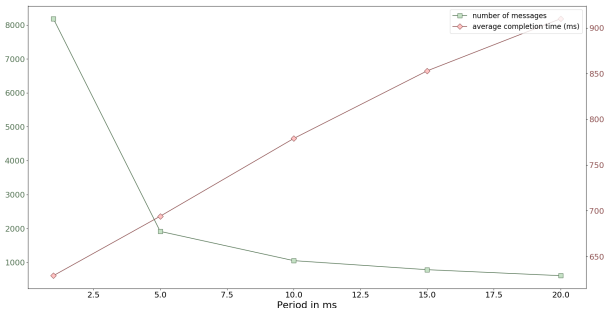

Figure 13: **Period AWS.** Comparison of different period time. The left y-axis shows the *average* number of messages received per node. The right y-axis shows the average completion time of Handel.

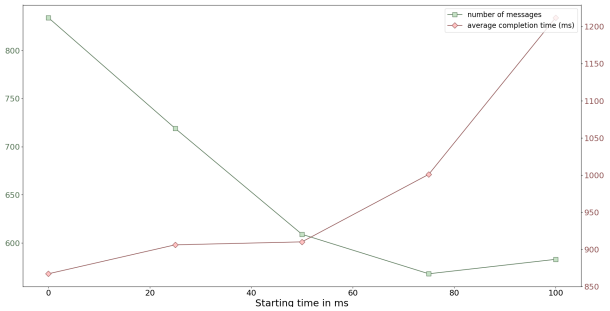

Figure 14: **Start Time AWS.** Comparison of different starting time for the levels. A start time of x means the level i starts at time $x * i$. The left y-axis shows the *average* number of messages received per node. The right y-axis shows the average completion time of Handel.

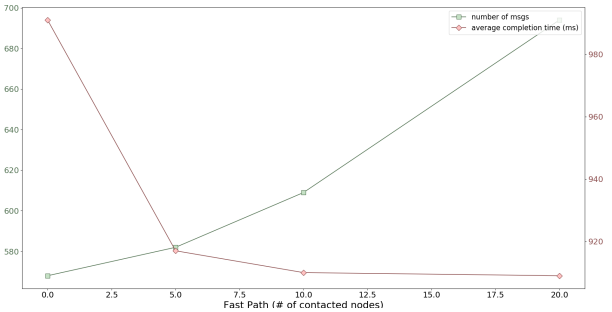

Figure 15: **Fast Path AWS.** Comparison of different values for the fast path mechanism. A fast path of x means a node contacts simultaneously x node when a level is completed. The left y-axis shows the *average* number of messages received per node. The right y-axis shows the average completion time of Handel.

A.4 TOR measurements with 20% dead nodes

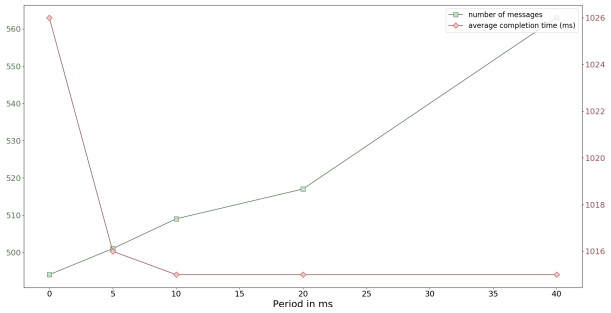

Figure 16: **Period AWS.** Comparison of different period time. The left y-axis shows the *average* number of messages received per node. The right y-axis shows the average completion time of Handel.

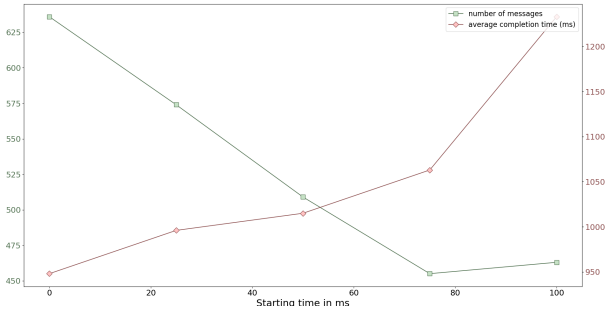

Figure 17: **Start Time AWS.** Comparison of different starting time for the levels. A start time of x means the level i starts at time $x * i$. The left y-axis shows the *average* number of messages received per node. The right y-axis shows the average completion time of Handel.

B AWS Latencies

The following table shows the different latencies from each of the 10 regions we used during our evaluation of Handel. Regions spans the entirety of the globe and therefore give rise to variations of more than 250ms.

Regions	Virginia	Mumbai	Seoul	Singapore	Sydney	Tokyo	Canada	Frankfurt	Ireland	London
Oregon	81	216	126	165	138	97	64	164	131	141
Virginia	-	182	181	232	195	167	13	88	80	75
Mumbai	-	-	152	62	223	123	194	111	122	113
Seoul	-	-	-	97	133	35	184	259	254	264
Singapore	-	-	-	-	169	69	218	162	174	171
Sydney	-	-	-	-	-	105	210	282	269	271
Tokyo	-	-	-	-	-	-	156	235	222	234
Canada	-	-	-	-	-	-	-	101	78	87
Frankfurt	-	-	-	-	-	-	-	-	24	13
Ireland	-	-	-	-	-	-	-	-	-	12