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
v1.03

Abstract

We present Handel, a Byzantine fault tolerant aggregation protocol that allows for the quick aggregation of cryptographic signatures over a WAN. Handel has logarithmic time and polylogarithmic 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 BN256 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, the blockchain protocols Algorand [19], Dfinity [25] and Tendermint [7] 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 [9], 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 enabling the whole set of nodes enhances the security properties of a 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 [4, 5].

Aggregating at scale is a well known problem [24, 30, 33, 42]. Cappos et al. proposed San Fermín [10], 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 [10]. 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 peer ranking plus 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 [15] to hide signers' IP addresses. The last scenario uses ring signatures [39] 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 [32]. It allows 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 [1, 13]. 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 Handel protocol, (section 3), a detailed analysis of this protocol in our threat model (section 4.3), 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, consider 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. We ask for **unforgeability of valid contributions**, eg. 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.

For concreteness, let us describe these components in the case of BLS signature aggregation [5]. Suppose there are N participants signing off on a message m. We define C to be $E \times (\{0,1\}^N \setminus \{[0,\ldots,0]\})$ where E is the set of points of the elliptic curve used for signatures. Thus, a contribution is a pair $(\sigma, [\epsilon_1, \ldots, \epsilon_N])$ where $\sigma \in E$ is a point on the curve (an aggregate signature), and $[\epsilon_1, \ldots, \epsilon_N]$ 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 $Pub = [pk_1, \ldots, pk_N]$ of N public keys. The verification function is defined on $c = (\sigma, [\epsilon_1, \ldots, \epsilon_N])$ 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 $[\epsilon_1, \ldots, \epsilon_N]$.

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

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

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 $c = (\sigma, [\epsilon_1, \ldots, \epsilon_N])$ to be $w(c) = \epsilon_1 + \cdots + \epsilon_N \in [1, N-1]$.

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 [16] or [14][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 [18].
- **Authentication:** We assume point-to-point authenticated channels between each pair of participating nodes.
- **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.

2.4 Goals

Handel aims to be a large scale aggregation protocol that outputs an aggregate contribution including *at least* a predefined threshold number $T \leq N$ of contributions, T being no larger than the number of honest nodes. 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.

[1] [14] uses the term *semi-synchronous*

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 where nodes occupy 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 nodes optimistically send their current best aggregate contributions, even if their previous level is incomplete. Nodes will often receive multiple contributions for the same level, and update their 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 on all *active* levels. Nodes activate new levels every 50ms.

In a Byzantine context, nodes may receive invalid contributions. Contributions should be verified before they are aggregated. However, verification can be costly (a few milliseconds [31] 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 verification of contributions must therefore 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 worst 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.

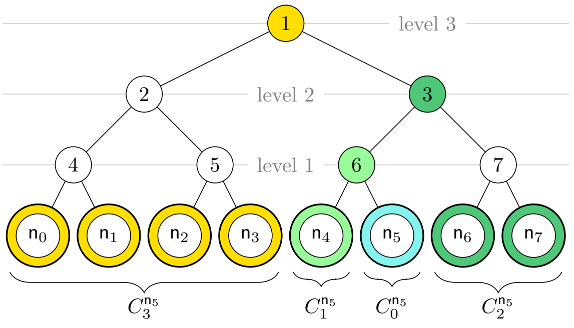

Figure 1: n_5's view of the network.

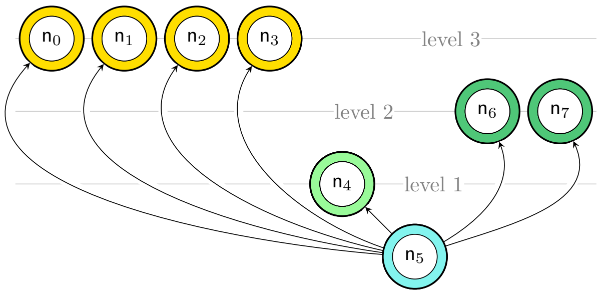

Figure 2: n_5 communication organization.

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 C_l^i node i's **level** l **peer set**. See figure 1 showing how node 5 partitions the set of nodes.

3.3 Peer ranks and contribution selection

As we saw, Handel is designed under the assumption that (1) nodes have limited resources and shouldn't have to verify all contributions (2) nodes may receive overlapping contributions (3) Byzantine nodes can send erroneous contributions or valid but nearly useless contributions, as a way to waste the node's resources and slow the aggregation. This is solved by selecting the contributions to verify through peer ranking and windowing.

We call **Verification Priority** (VP) the ranks publicly given by a node to its peers. The VP function is computable from public data –the node list and a seed shared by all nodes–, and act as pseudo random permutation $\mathrm{VP}_i(j) = \mathit{rand}(N, \mathit{seed}; i, j) \in [0, N[$ of its last argument j, where *seed* is a seed value.

Nodes sort the received contributions by their sender's VP, and select the ones in the scoring **window**: if v denotes the highest VP_i among i's incoming contributions, we only **score** contributions with sender VP between v and $v + \mathit{windowSize}$.

The **score** of a level l contribution c is, if (c, In_l^i) is aggregatable, the weight of the aggregate. Otherwise, it is the weight of the contribution obtained by aggregating c with all previously verified and aggregatable individual level l contributions. The highest scoring contribution gets verified first.

The result of the verification changes the window size. The size is increased by a factor two on verification success, decreased by a factor 4 on verification failure.

Nodes take into account the VP when they send their own contributions: inside a level, they send their contributions to the nodes that give them the highest VP first. In other words, every node computes its level l Contribution Prioritization Vector (CPV) $[\mathrm{VP}_{s_1}(i), \ldots, \mathrm{VP}_{s_{2^l}}(i)]$ where the s_j are its level l peers, and contacts the level l peers that rank it highest first.

3.4 Node state

The internal state of a node i consists of, for every level l

1. the current best aggregate contribution In_l^i,
2. the current best aggregate contribution Out_l^i to send out,
3. the list of as yet unverified contributions $\mathrm{Inc}_{l,j}^i$ received from level l peers j
4. the start time $StartTime_l$
5. the current window size *window*,
6. the VP_i and CPV vectors.

In_0^i is i's individual contribution. For all l, $Out_l^i = Aggr(In_0^i, In_1^i, \ldots, In_{l-1}^i)$ is the aggregate of the levelwise best received contributions up to level $l-1$. We call Out_l^i (resp. In_l^i), and level l by extension, *complete* if it includes all contributions from all levels $\leq l$ (resp. $= l$).

3.5 Messages in Handel

There is only one type of message in Handel which contains 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.

3.5.1 Sending contributions

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 each 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.

3.5.2 Receiving contributions

When node i receives a message from node j at level l, it includes j's aggregate and individual contributions (Out_l^j and In_0^j respectively) in its list of as-of-yet unverified level l contributions. If this list already contains a contribution from j, it keeps the one with greater *weight*. The contribution vector is thus bounded in size. The node purges this list of any useless contributions, and continuously selects the best one to verify, as described in section 3.3.

3.5.3 Pruning contributions

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 aggregated with, and have a lower score than In_l^i.

3.6 Parameter values

We explained in section 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 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 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 contributions 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. Activating levels incrementally saves an important number of messages when the threshold is reached quickly. Compared to a deployment without this technique we observed similar times to completion but 20% fewer messages, see appendix A.1.

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. We limit the maximum value at 128 to cap the time to return to a window size of 1 if attacked.

4 Analysis

4.1 Complexity

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

Time complexity is $O(log(N))$, as there are $log(N)$ levels, as in [10].

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, $O(log^2(N))$ messages are sent this way. *Fast path* can be triggered once per level, and involves sending a small fixed number of messages, and thus adds $O(log(N))$ messages. Message complexity is thus $O(log^2(N))$.

CPU consumption is less than message complexity: not all messages received get verified. In the worst case, all received messages are verified and CPU consumption is $O(log^2(N))$.

With a large fraction of Byzantine nodes, time complexity degenerates to $O(N)$: 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 $O(log(N))$, the worst they can do is not participate. Indeed, Byzantine nodes can either (1) refrain from participating, (2) send high-scoring invalid contributions (this wastes the target's CPU cycles and prevents honest contributions from ever being examined and aggregated), (3) send valid but very small contributions (this prevents larger contributions from honest nodes from being integrated) or perform (4) a combination of the above. The raison d'être of the score function is to counteract attack (3): nodes verify 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 valid contributions from getting verified, hence the protocol will progress. (4) reduces to (3): with window size reduced to 1, peer ranking applies with maximum force, as nodes will only verify the contributions from their top ranking peer. all of them will be verified anyway, so (4) is equivalent to (3).

4.2 Versatility of Handel

4.2.1 Heterogeneous latencies

Handel can deal with networks with very heterogeneous latencies. To illustrate this, we consider the following 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, the protocol's time complexity remains $O(log(N))$. This is because fast nodes are the main drivers of aggregation. Their progress is initially independent of that of the slow nodes, which are initially indistinguishable from fail-silent nodes. However, once a slow node has sent its contribution to a fast one, its contribution will become widely available among the fast nodes.

For slow nodes, the expected time τ to reach a fast node is dictated mainly by latency. In our example with 20% slow nodes and $\lambda = 20\text{DP}$, a slow node will reach a fast one on average[2] in $20.25\text{DP} \approx \lambda$. In other words, the time for a fast node to get the final aggregation is $\textit{slow latency} + \textit{fast latency} \cdot O(log(N))$.

As latency between slow nodes is much larger than latency between fast nodes, slow nodes will rarely contribute interesting new aggregates and mostly just receive those calculated by fast nodes. Hence, the time for a slow node to receive the final aggregation is $2 \cdot \textit{slow latency} + \textit{fast latency} \cdot O(log(N))$.

[2]with ρ the ratio of fast nodes, and λ the latency we have $\tau = \sum_{k=0}^{\infty}(\lambda + k \cdot \text{DP})\rho(1-\rho)^k$.

4.2.2 Heterogeneous CPU capacities

If all nodes have similar latencies, but very different CPU power, the time complexity does not change: because of the scoring function, the weak nodes check fewer contributions, but prioritize interesting (high-scoring) contributions. As was the case for heterogeneous 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), and (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 may start the protocol late. This is similar to having a one-time latency spike at the start. However, late nodes quickly catch up on account of them immediately receiving weighty contributions from the nodes that started early. Heterogeneous clocks thus have smaller impact than heterogeneous latencies.

4.3 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 [8], 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 [21]. We argue that this is a realistic assumption as Handel assumes a publicly known message before the aggregation protocol starts. The security analysis uses 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 section 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 i's 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 section 2.1. □

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. Its strengths and weaknesses, however, aren't uniform across all deployment scenarios. We focus on the privacy and DoS protection provided by various Handel deployments in the context of signature aggregation.

Handel forces signers to reveal their IP addresses so they can receive Handel packets. In some cases, this public mapping between public keys and IP addresses is problematic as it leaves signers vulnerable to DoS attacks and privacy breaches. For instance, in proof-of-stake protocols, a signer may want to hide its IP address if it is linked with the public key controlling its staked crypto-currency funds. The first scenario we consider is the vanilla deployment of Handel where IP addresses are public. The second one explores using an anonymity network (such as Tor [15]) and its synergies with Handel. Thirdly, we describe a deployment scenario where ring signatures provide signers anonymity within the set of participants.

5.1 Vanilla deployment

In this deployment, every node is reachable from the Internet. There is a public global directory containing the mapping between signers' public keys and their public addresses. This deployment scenario is standard in distributed systems and, as such, well understood. On the other hand, having signers' IP addresses be public exposes them to DoS attacks or even full compromise. One can still put in place defenses against this class of attacks. In order to defend against a full compromise, a signer can put its signing key in an external HSM module [11]. 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 a node's individual contribution be included in a final aggregate contribution.

5.2 Anonymity network

Participants wishing to hide their public IP address can use Tor [15] (alternatively I2P [26] or Freenet [12]), to deploy hidden services and use the service address instead of a public IP address. Importantly, participants can be on different networks: some can be on Tor while others can use their real IP address. In such a deployment, latencies are very heterogeneous: UDP messages typically reach their destination within 100-200ms [38], while Tor messages take upwards of 500ms [36]. We show in section 4.2.1 that with a minority (eg. 20%) of slow nodes the increase on time to completion is limited to a single Tor roundtrip time.

In this deployment, *Anonymity* guarantees are 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 [27] and the most effective DoS attack remains that against the whole Tor network. Of course, the security guarantees offered by such a deployment are limited by those of the underlying anonymous network. For example, a passive global adversary can target Tor users with traffic correlation attacks [29].

5.3 Anonymous signers

Anonymity can be achieved if participants are allowed to sign messages using fresh secret/public key pairs. By means of *linkable ring signatures* [34], 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 $\textit{IP-address}_i \| AK_i$ 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 (through *linkability*), the associated AK_i is discarded from AK. Using *traceable ring signatures* [17], 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 opt in, which is fortunate considering Handel's ability to scale. Importantly, the setup phase only has to be done *once* and can be used for multiple Handel rounds.

6 Implementation

We released a reference implementation of Handel as an open source Golang library [32] 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 [4, 5] on the BN256 [37] curve (optimized implementation by Cloudflare [13]), with UDP as the transport layer. Our implementation includes several technical optimizations not mentioned in Section 3. Specifically, 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. As an alternative, nodes using raw 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 strives to minimize 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 scores and 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), an aggregate contribution (as defined in section 2.1) and the signature of the sender (i.e. its individual contribution). 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 about 400 bytes.

Wire protocol. We used the gob encoding scheme [20] which adds a few extra bytes to 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 the effect of different parameters on Handel's performance.

7.1 Experimental setup

Measurements. All Handel nodes log measurements such as the total number of messages received/sent, verifications performed, 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 a 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. Refer to appendix B for the exact latency distribution between regions.

7.2 Comparative baseline

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

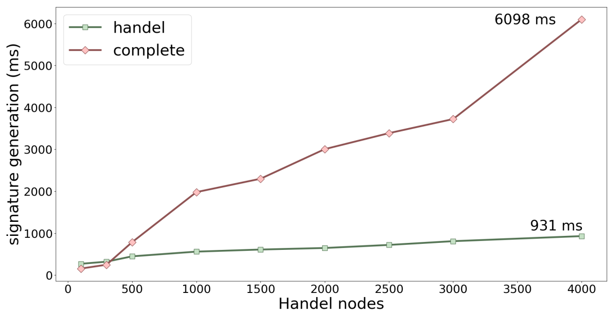

Figure 3: **Time** comparison up to 4000 nodes with a 99% threshold

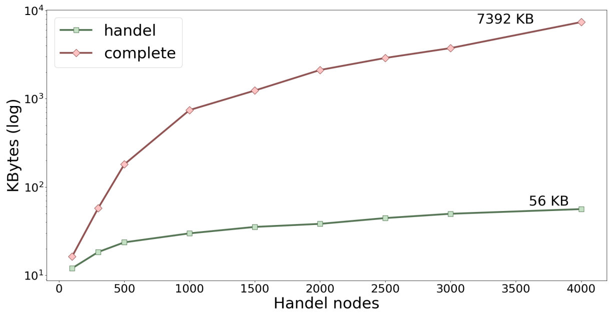

Figure 4: **Outgoing Network** comparison up to 4000 nodes with a 99% threshold

nodes. Since Handel eventually leads to this scenario (see section 4.3) we wanted 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 tested Handel's resistance to node failures. Figure 5 illustrates Handel's ability to aggregate 51% of signatures from 4000 nodes at various rates of node failure.

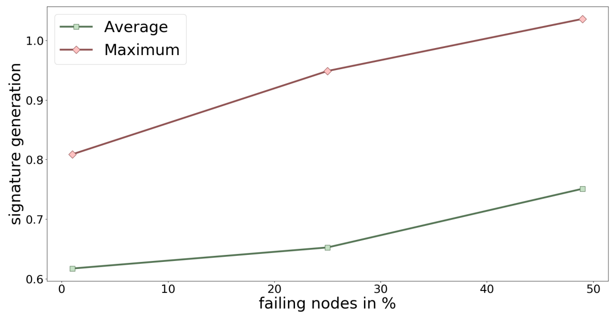

Figure 5: Various percentages of failing nodes over a total of 4000 nodes for a 51% threshold

7.4 CPU consumption

This experiment was 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 close to the least possible number of aggregate signatures for 4000 nodes: around $30 \approx 24 = 2 \cdot log_2(4000)$ (recall that every message contains *two* signatures). The graph also shows that some nodes verified *far more* signatures than that. High latencies contribute to this discrepancy: some nodes may have to wait *longer* to receive interesting contributions. In the meantime, they will verify all other incoming signatures, eventually even those that score poorly.

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 *permissioned networks* such as grid networks or monitoring networks.

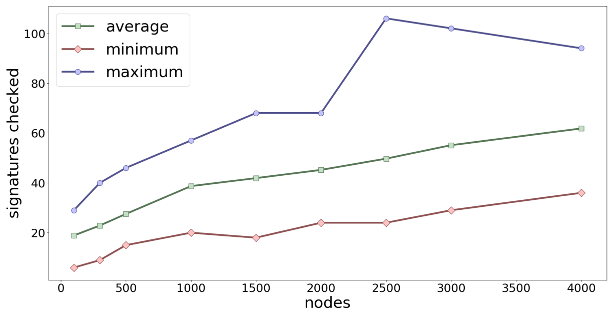

Figure 6: Number of signatures checked on a node up to 4000 nodes with a 99% threshold

8.1 San Fermín

San Fermín [10] introduced the *binomial swap forest* technique enabling $O(log(N))$ 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; long timeouts on the other hand negatively impact time to completion.

Byzantine actors. San Fermín is vulnerable to Byzantine attacks: an adversary can prevent a contribution from being included by contacting high level nodes early on. A byzantine node can thus harness many honest messages while also reducing the chances of its low level peers of ever being contacted by higher level ones.

8.2 Gossip-based solutions

The systems from [24, 28, 30] are gossip-based solutions that converge exponentially fast for simple aggregation functions such as sum and average. However, these systems do not protect against Byzantine behaviors.

8.3 Tree-based solutions

Tree based constructions can help address scalability. Astrolab [40] is the first system using hierarchy zones mimicking the design of DNS zones. Systems such as SDIMS [42], CONE [3], Willow [41] combine hierarchy-based constructions with the design principle underlying *distributed hash table* (DHT) systems. Li's protocol [33] is natively compatible with any DHT abstraction. However, individual nodes are solely responsible for aggregating and forwarding contributions from their respective subtrees; a single node failure can thus severely impact the protocol. Finally, Grumbach et al. [23] use a mechanism similar to binomial swap forests with the native underlying trees of the Kademlia DHT [35]. 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 Conclusion

Multi-signature schemes have garnered a lot of attention lately, yet data aggregation frameworks at scale have been examined only under the fail-silent model, and no multi-signature aggregation frameworks have been developed to withstand Byzantine scenarios. We argue that this is because Byzantine nodes are able to follow the protocol while providing 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 [22], zk-STARKS [2]), we expect fast Byzantine fault tolerant aggregation schemes to become increasingly important, as these schemes can guarantee the validity of the data to be aggregated. Lastly, Handel improves on the simple fail-silent case by not relying on timeouts and by supporting heterogeneous network and CPU capacities of the participating nodes.

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] Barreto, P. S., and Naehrig, M. Pairing-friendly elliptic curves of prime order. In *International Workshop on Selected Areas in Cryptography* (2005), Springer, pp. 319–331.

[2] Ben-Sasson, E., Bentov, I., Horesh, Y., and Riabzev, M. Scalable, transparent, and post-quantum secure computational integrity. *IACR Cryptology ePrint Archive 2018* (2018), 46.

[3] Bhagwan, R., Varghese, G., and Voelker, G. M. Cone: augmenting DHTs to qupport distributed resource discovery. Tech. rep., University of California, San Diego, July 2003.

[4] Boneh, D., Drijvers, M., and Neven, G. BLS multi-signatures with public-key aggregation, 2018.

[5] Boneh, D., Lynn, B., and Shacham, H. Short signatures from the Weil pairing. In *International Conference on the Theory and Application of Cryptology and Information Security* (2001), Springer, pp. 514–532.

[6] Bridge, V., Gailly, N., Kolad, B., and Liochon, N. Wittgenstein: protocol simulator. https://github.com/ConsenSys/wittgenstein, 2018.

[7] Buchman, E. Tendermint: Byzantine fault Tolerance in the age of blockchains, 2016.

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

[9] Canetti, R., Damgaard, I., Dziembowski, S., Ishai, Y., and Malkin, T. On adaptive vs. non-adaptive security of multiparty protocols. In

International Conference on the Theory and Applications of Cryptographic Techniques (2001), Springer, pp. 262–279.

[10] Cappos, J., and Hartman, J. H. San fermín: Aggregating large data sets using a binomial swap forest. In *NSDI* (2008).

[11] Cifuentes, F., Hevia, A., Montoto, F., Barros, T., Ramiro, V., and Bustos-Jiménez, J. Poor man's hardware security module (pmhsm): A threshold cryptographic backend for dnssec. In *Proceedings of the 9th Latin America Networking Conference* (2016), ACM, pp. 59–64.

[12] Clarke, I., Sandberg, O., Wiley, B., and Hong, T. W. Freenet: A distributed anonymous information storage and retrieval system. In *International Workshop on Designing Privacy Enhancing Technologies: Design Issues in Anonymity and Unobservability* (Berlin, Heidelberg, 2001), Springer-Verlag, pp. 46–66.

[13] Cloudflare. BN256 Cloudflare's implementation. https://github.com/cloudflare/bn256, 2018.

[14] David, B., Gaži, P., Kiayias, A., and Russell, A. Ouroboros Praos: An adaptively-secure, semi-synchronous proof-of-stake blockchain. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques* (2018), Springer, pp. 66–98.

[15] Dingledine, R., Mathewson, N., and Syverson, P. F. Tor: The second-generation onion router. In *USENIX Security Symposium* (2004).

[16] Dwork, C., Lynch, N., and Stockmeyer, L. Consensus in the presence of partial synchrony. *Journal of the ACM (JACM) 35*, 2 (1988), 288–323.

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

[18] Garay, J., Kiayias, A., and Leonardos, N. The Bitcoin backbone protocol: analysis and applications. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques* (2015), Springer, pp. 281–310.

[19] Gilad, Y., Hemo, R., Micali, S., Vlachos, G., and Zeldovich, N. Algorand: scaling Byzantine agreements for cryptocurrencies. In *Proceedings of the 26th Symposium on Operating Systems Principles* (2017), ACM, pp. 51–68.

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

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

[22] Groth, J. On the size of pairing-based non-interactive arguments. *IACR Cryptology ePrint Archive 2016* (2016), 260.

[23] Grumbach, S., and Riemann, R. Secure and trustable distributed aggregation based on Kademlia. *CoRR abs/1709.03265* (2017).

[24] Gupta, I., Van Renesse, R., and Birman, K. P. Scalable fault-tolerant aggregation in large process groups. In *2001 International Conference on Dependable Systems and Networks* (2001), IEEE, pp. 433–442.

[25] Hanke, T., Movahedi, M., and Williams, D. Dfinity technology overview series, consensus system. *arXiv preprint arXiv:1805.04548* (2018).

[26] I2P. Invisible Internet Project. https://geti2p.net/en/, 2000.

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

[28] Jelasity, M., Montresor, A., and Babaoglu, O. Gossip-based aggregation

in large dynamic networks. *ACM Transactions on Computer Systems (TOCS) 23*, 3 (2005), 219–252.

[29] Johnson, A., Wacek, C., Jansen, R., Sherr, M., and Syverson, P. Users get routed: Traffic correlation on Tor by realistic adversaries. In *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security* (2013), ACM, pp. 337–348.

[30] Kempe, D., Dobra, A., and Gehrke, J. Gossip-based computation of aggregate information. In *44th Annual IEEE Symposium on Foundations of Computer Science, 2003. Proceedings.* (2003), IEEE, pp. 482–491.

[31] Kolad, B. Benchmarking of Milagro BLS implementation. https://github.com/ConsenSys/mikuli, 2018.

[32] Kolad, B., Gailly, N., and Liochon, N. Handel implementation in go. https://github.com/ConsenSys/handel, 2019.

[33] Li, J., Sollins, K., and Lim, D.-Y. Implementing aggregation and broadcast over distributed hash tables. *ACM SIGCOMM Computer Communication Review 35*, 1 (2005), 81–92.

[34] Liu, J. K., and Wong, D. S. Linkable ring signatures: Security models and new schemes. In *International Conference on Computational Science and Its Applications* (2005), Springer, pp. 614–623.

[35] Maymounkov, P., and Mazieres, D. Kademlia: A peer-to-peer information system based on the xor metric. In *International Workshop on Peer-to-Peer Systems* (2002), Springer, pp. 53–65.

[36] Metrics, T. https://metrics.torproject.org/torperf.html, 2018.

[37] Naehrig, M., Niederhagen, R., and Schwabe, P. New software speed records for cryptographic pairings. In *International Conference on Cryptology and Information Security in Latin America* (2010), Springer, pp. 109–123.

[38] Network, W. Global ping statistics. https://wondernetwork.com/pings, Dec 2018.

[39] Rivest, R. L., Shamir, A., and Tauman, Y. How to leak a secret. In *International Conference on the Theory and Application of Cryptology and Information Security* (2001), Springer, pp. 552–565.

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

[41] Van Renesse, R., and Bozdog, A. Willow: DHT, aggregation, and publish/subscribe in one protocol. In *International Workshop on Peer-to-Peer Systems* (2004), Springer, pp. 173–183.

[42] Yalagandula, P., and Dahlin, M. A scalable distributed information management system. In *SIGCOMM* (2004).

Appendices

A Simulations

We present here results from simulations we ran to determine good parameters ahead of running the large-scale tests from section 7. We used the Wittgenstein simulator [6] to run these simulations, with the following configurations: (1) nodes are deployed either on AWS datacenters, using the configuration we presented in 7.1, or in 242 different cities for which the latencies are known [38]. (2) Verification time is 4ms on a standard node, 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 nodes are down. (4) Nodes can be on a Tor network (eg. have an extra latency of 1000ms round-trip). We measure the average time for the honest nodes to reach a threshold of 99% of the honest nodes. The numbers presented are averages of 5 runs.

A.1 Deployment in 242 cities

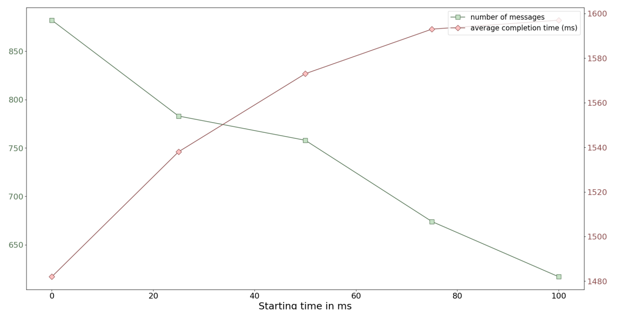

Figure 7: **Start Time.** Comparison of different starting times for levels. A start time of x means that level i starts at time $i \times x$. The left y-axis shows the *average* number of messages received per node. The right y-axis shows the average completion time of Handel.

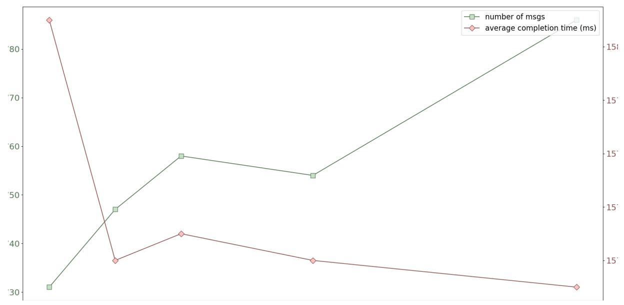

Figure 8: **Fast Path.** Comparison of different values for the fast path mechanism. A fast path of x means a node simultaneously contacts x nodes when a level is completed. The left y-axis shows the *average* number of messages received per node. The right y-axis shows the average completion time of Handel.

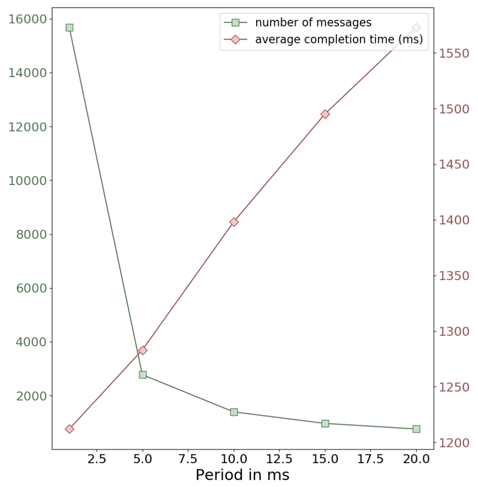

Figure 9: **Dissemination period.** Comparison of different Dissemination periods. The left y-axis shows the *average* number of messages received per node. The right y-axis shows the average completion time of Handel.

A.2 AWS deployment

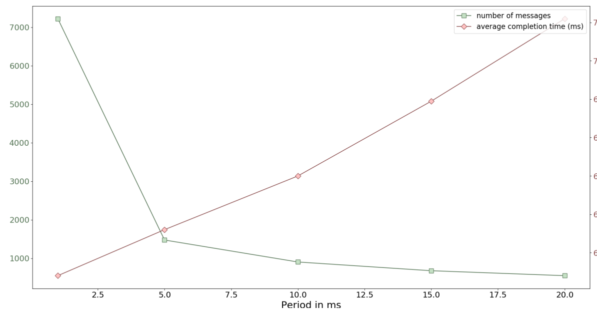

Figure 10: **Period AWS.** Comparison of different period times. The left y-axis shows the *average* number of messages received per node. The right y-axis shows the average completion time of Handel.

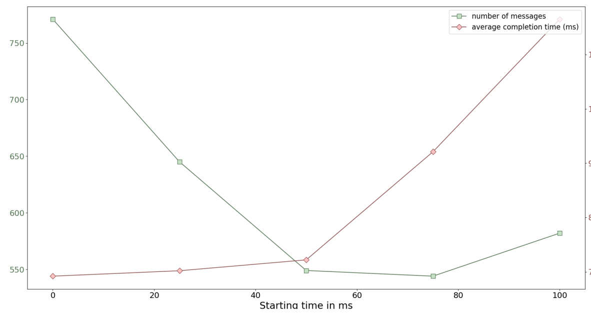

Figure 11: **Start Time AWS.** Comparison of different starting times for the levels. A start time of x means that level i starts at time $x * i$. The left y-axis shows the *average* number of messages received per node. The right y-axis shows the average completion time of Handel.

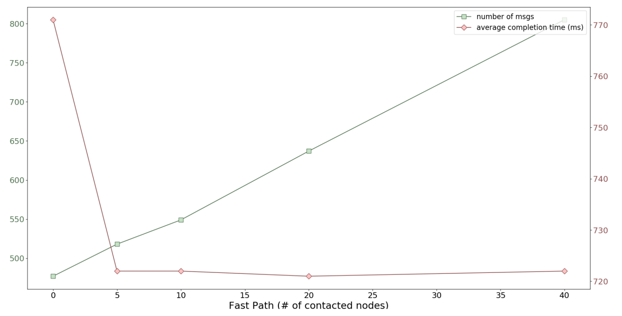

Figure 12: **Fast Path AWS.** Comparison of different values for the fast path mechanism. A fast path of x means a node simultaneously contacts x node when a level is completed. The left y-axis shows the *average* number of messages received per node. The right y-axis shows the average completion time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

A.3 Deployment with Tor

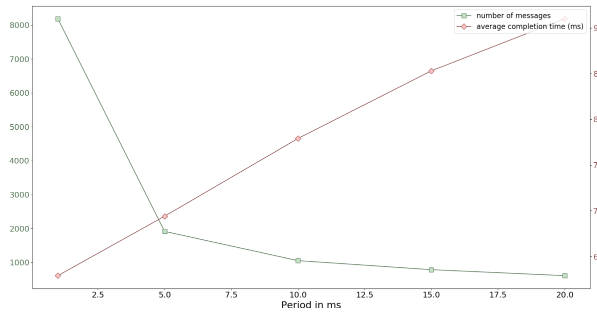

Figure 13: **Period AWS.** Comparison of different period times. The left y-axis shows the *average* number of messages received per node. The right y-axis shows the average completion time of Handel.

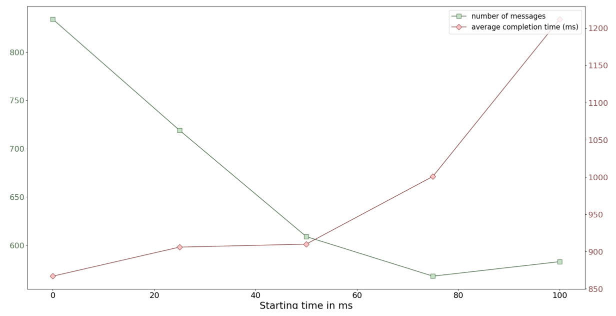

Figure 14: **Start Time AWS.** Comparison of different starting times for the levels. A start time of x means the level i starts at time $x * i$. The left y-axis shows the *average* number of messages received per node. The right y-axis shows the average completion time of Handel.

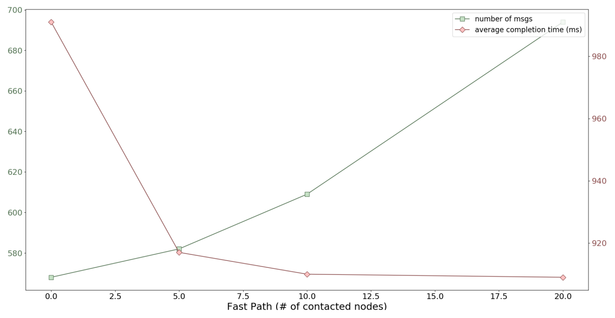

Figure 15: **Fast Path AWS.** Comparison of different values for the fast path mechanism. A fast path of x means a node simultaneously contacts x node when a level is completed. The left y-axis shows the *average* number of messages received per node. The right y-axis shows the average completion time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

A.4 Deployment with and 20% dead nodes

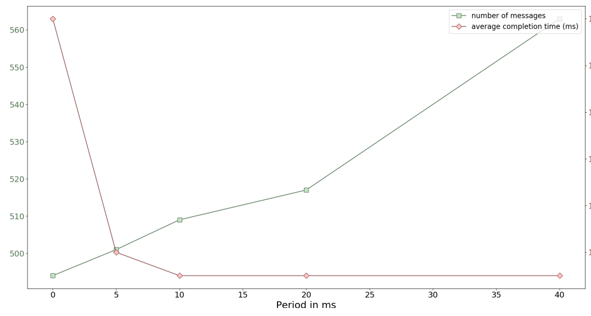

Figure 16: **Period AWS.** Comparison of different period times. The left y-axis shows the *average* number of messages received per node. The right y-axis shows the average completion time of Handel.

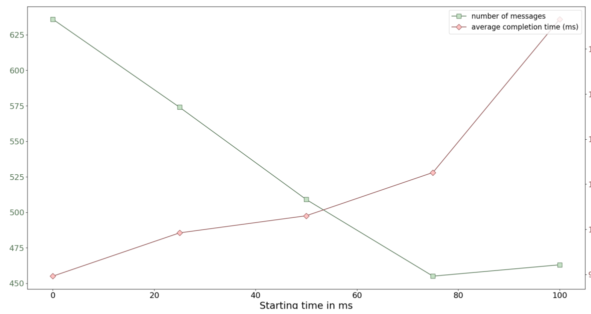

Figure 17: **Start Time AWS.** Comparison of different starting times for the levels. A start time of x means the level i starts at time $x * i$. The left y-axis shows the *average* number of messages received per node. The right y-axis shows the average completion time of Handel.

B AWS Latencies

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

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