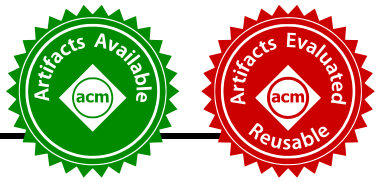


AGGREGATHOR: Byzantine Machine Learning via Robust Gradient Aggregation

Georgios Damaskinos [1] **El Mahdi El Mhamdi** [1] **Rachid Guerraoui** [1] **Arsany Guirguis** [1] **Sébastien Rouault** [1]

ABSTRACT

We present AGGREGATHOR, a framework that implements state-of-the-art robust (*Byzantine-resilient*) distributed stochastic gradient descent. Following the standard parameter server model, we assume that a minority of *worker* machines can be controlled by an adversary and behave arbitrarily. Such a setting has been theoretically studied with several of the existing approaches using a *robust aggregation* of the workers' gradient estimations. Yet, the question is whether a Byzantine-resilient aggregation can leverage more workers to speed-up learning. We answer this theoretical question, and implement these state-of-the-art *theoretical* approaches on AGGREGATHOR, to assess their *practical* costs. We built AGGREGATHOR around TensorFlow and introduce modifications for vanilla TensorFlow towards making it usable in an actual Byzantine setting. AGGREGATHOR also permits the use of unreliable gradient transfer over UDP to provide further speed-up (without losing the accuracy) over the native communication protocols (TCP-based) of TensorFlow in saturated networks. We quantify the overhead of Byzantine resilience of AGGREGATHOR to 19% and 43% (to ensure *weak* and *strong* Byzantine resilience respectively) compared to vanilla TensorFlow.

1 INTRODUCTION

Billions of Internet users share new data (e.g., photos, videos, posts) every day. The amount of data generated keeps increasing at a continuous rate and has been a wonderful opportunity for *machine learning (ML)* algorithms. Yet, precisely because of the huge amount of data available, these algorithms require immense demands of computing resources in order to *train* the ML model and provide accurate predictions. As a result, most ML algorithms today are distributed (Jaggi et al., 2014; Recht et al., 2011; Dean et al., 2012; Abadi et al., 2016a; Chilimbi et al., 2014; Li et al., 2014; Meng et al., 2016). Typically, a (parameter) server coordinates the distribution of the training tasks among a large set of worker nodes. The parameter server *aggregates* the responses of the workers (e.g., average the gradients) and updates a global view of the model.

Besides scaling, another motivation for distributed ML schemes is privacy. The workers could be user machines keeping their data locally and collaborating through a parameter server to achieve some global machine learning task (Abadi et al., 2016b; Shokri & Shmatikov, 2015).

The multiplicity of workers increases however the possibility of failures. Workers can be subject to software bugs and hardware faults[1] (Gunawi et al., 2018). They can access corrupt datasets or even be hijacked by an *adversary*. Such failures can be fatal to most modern ML schemes, even if only a single worker is faulty. Ideally, distributed ML applications should tolerate *Byzantine* failures, encapsulating all possible malfunctions. These failures include poisoning attacks (Biggio & Laskov, 2012), in the parlance of *adversarial machine learning* (Papernot et al., 2017; Biggio & Roli, 2017; Biggio & Laskov, 2012).

Traditionally, Byzantine resilience of a distributed service is achieved by using *Byzantine-resilient state machine replication* techniques (Schneider, 1990; Castro et al., 1999; Sen et al., 2010; Cowling et al., 2006; Chen et al., 2015). Whereas this approach looks feasible for the single and deterministic server, applying it to workers appears less realistic. Indeed, the workers *parallelize*, i.e., share the computational-heavy part of *Stochastic Gradient Descent* (SGD)[2]: the gradient estimation. Having them compute different gradients (inherently non-deterministic) to agree on only one of their estimations, would imply losing a significant amount of the computations made, i.e., losing the very purpose of the distribution of the estimation. This would merely lead to additional synchronization and communication costs compared to non-distributing SGD.

Some theoretical approaches have been recently proposed

*Equal contribution [1]EPFL. Correspondence to: FirstName LastName(without spaces) <firstname.lastname@epfl.ch>.

Proceedings of the 1st SysML Conference, Palo Alto, CA, USA, 2019.

[1]The parameter server is trusted in this paper.

[2]We focus on SGD as the workhorse optimization algorithm.

to address Byzantine-resilience without replicating the workers (Blanchard et al., 2017; Diakonikolas et al., 2017). In short, the idea is to use more sophisticated forms of aggregation[3] (e.g. *median*) than simple averaging. Despite their provable guarantees, most of these algorithms only ensure a *weak* form of resilience against Byzantine failures. These algorithms indeed ensure convergence to some state, but this final state could be heavily influenced by the Byzantine workers (El Mhamdi et al., 2018). For most critical distributed ML applications, a *stronger* form of Byzantine resilience is desirable, where SGD would converge to a state that could have been attained in a non-Byzantine environment. Draco (Chen et al., 2018) and BULYAN (El Mhamdi et al., 2018) are the only proposals that guarantee strong Byzantine resilience. On the one hand, Draco requires (a) computing several gradients per worker and step (instead of one), and (b) strong assumptions as we discuss in §5. On the other hand, BULYAN internally *iterates over* a weak Byzantine GAR (e.g. Krum (Blanchard et al., 2017)), which is experimentally shown by its authors to be sub-optimal. Apart from Draco, none of the Byzantine-resilient approaches has been implemented and tested in a realistic distributed ML environment to assess the scalability.

We present AGGREGATHOR, a light and fast framework that brings Byzantine resilience to distributed machine learning. Due to its popularity and wide adoption, we build AGGREGATHOR around TensorFlow. Our framework can thus be used to distribute, in a secure way, the training of any ML model developed for TensorFlow. AGGREGATHOR simplifies the experimentation on large and possibly heterogeneous server farms by providing automatic, policy-based device selection and cluster-wide allocation in TensorFlow. Following the TensorFlow design, any worker (including Byzantine ones) can alter the graph and execute code on any other node. We provide a code patch for TensorFlow that prohibits such a behavior to ensure Byzantine resilience. AGGREGATHOR allows for both levels of robustness: *weak* and *strong* resilience through MULTI-KRUM[4] and BULYAN respectively.

MULTI-KRUM assigns a *score* (based on a sum of distances with the closest neighbors) to each gradient a worker submits to the server, and then returns the average of the smallest scoring gradients set (§2.3). We provide a fast, memory scarce implementation of MULTI-KRUM by fully parallelizing each of the computational-heavy steps. We formally prove the Byzantine resilience, convergence and slowdown of MULTI-KRUM.

[3] We refer to the various forms of gradient aggregation as *Gradient Aggregation Rules (GAR)*.

[4] MULTI-KRUM was first discussed (Blanchard et al., 2017) without any theoretical guarantees for Byzantine resilience.

BULYAN robustly aggregates n vectors by iterating several times over a second (underlying) Byzantine-resilient GAR. In each loop, BULYAN extracts the gradient(s) selected by the underlying GAR, computes the closest values to the coordinate-wise median of the extracted gradient(s) and finally returns the coordinate-wise average of these values. We optimize our implementation of BULYAN given MULTI-KRUM as the underlying GAR. We accelerate the execution by removing all the redundant computations: MULTI-KRUM performs the distance computations only on the first iteration of BULYAN; the next iterations only update the scores. We also parallelize the loops over the gradient coordinates (e.g. median coordinate-wise). We reduce the memory cost by allocating space only for one iteration of MULTI-KRUM along with the intermediate selected gradients. Both of our implementations support non-finite (i.e., $\pm$*Infinity* and *NaN*) coordinates, which is a crucial feature when facing actual malicious workers.

We evaluate and compare the performance of AGGREGATHOR against vanilla TensorFlow when no attacks occur. We first deploy both systems on top of the default, reliable communication protocol and quantify the respective overhead of MULTI-KRUM and BULYAN, i.e., the cost of weak and strong Byzantine resilience, to 19% and 43% respectively. We then consider an unreliable UDP-based communication channel leading to packet losses, to which TensorFlow is intolerant. We provide the necessary TensorFlow modifications to accommodate this lossy scheme. We show that AGGREGATHOR can also tolerate unreliable communication with a speedup gain of six times against vanilla TensorFlow in saturated networks. We also compare AGGREGATHOR with Draco and show a performance gain in terms of throughput and convergence rate.

The rest of the paper is structured as follows. We recall some preliminary ML concepts and introduce weak and strong Byzantine resilience for distributed gradient descent in §2. We describe the design of AGGREGATHOR in §3 and empirically evaluate its effectiveness in §4. We review the related work in §5 and conclude our paper in §6.

The latest version of AGGREGATHOR is publicly available (MIT license) on GitHub: `LPD-EPFL/AggregaThor`.

2 BACKGROUND AND PRELIMINARIES

2.1 Stochastic Gradient Descent

For illustration purposes but without loss of generality, we consider supervised deep learning for image classification. The learning task consists in making accurate predictions for the labels of each data instance ξ_i by using a convolutional neural network (CNN); we denote the d parameters (model) of the CNN by $\boldsymbol{x}$. Each data instance has a set of

features (image pixels), and a set of labels (e.g., {cat, person}). The CNN is trained with the popular backpropagation algorithm based on SGD. Specifically, SGD addresses the following optimization problem.

$$\min_{\boldsymbol{x}\in\mathbb{R}^d} Q(\boldsymbol{x}) \triangleq \mathbb{E}_\xi F(\boldsymbol{x};\xi) \tag{1}$$

where ξ is a random variable representing a total of B data instances and $F(\boldsymbol{x};\xi)$ is the loss function. The function $Q(\boldsymbol{x})$ is smooth but not convex.

SGD computes the gradient ($\boldsymbol{G}(\boldsymbol{x},\xi) \triangleq \nabla F(\boldsymbol{x};\xi)$) and then updates the model parameters ($\boldsymbol{x}$) in a direction opposite to that of the gradient (descent). The vanilla SGD update rule given a sequence of learning rates $\{\gamma_k\}$ at any given step[5] is the following:

$$\boldsymbol{x}^{(k+1)} = \boldsymbol{x}^{(k)} - \gamma_k \cdot \boldsymbol{G}(\boldsymbol{x}^{(k)},\xi)) \tag{2}$$

The popularity of SGD stems from its ability to employ noisy approximations of the actual gradient. In a distributed setup, SGD employs a *mini-batch* of $b < B$ training instances for the gradient computation:

$$\boldsymbol{G}(\boldsymbol{x},\xi) = \sum_{i=1}^{b} \boldsymbol{G}(\boldsymbol{x},\xi_i) \tag{3}$$

The size of the mini-batch (b) affects the amount of parallelism (Equation 3) that modern computing clusters (multi-GPU etc.) largely benefit from. Scaling the mini-batch size to exploit additional parallelism requires however a non-trivial selection of the sequence of learning rates (Goyal et al., 2017). A very important assumption for the convergence properties of SGD is that each gradient is an unbiased estimation of the actual gradient, which is typically ensured through uniform random sampling, i.e., gradients that are on expectation equal to the actual gradient.

2.2 Byzantine Resilience

SGD has been both theoretically and empirically proven to not be resilient against Byzantine worker behavior (Blanchard et al., 2017). A Byzantine worker can propose a gradient that can completely ruin the training procedure.

Weak Byzantine resilience. A very recent line of theoretical research has addressed the problem of Byzantine-resilient SGD (Blanchard et al., 2017; El Mhamdi et al., 2018; Su, 2017; Xie et al., 2018; Yin et al., 2018). These SGD variants all aggregate the gradients obtained from the workers before deriving the final gradient. Essentially, they compute statistics (e.g. median, quantiles, principal component analysis) over the set of aggregated gradients to derive the final gradient. Basically, the update rule (Equation 2) for n workers becomes:

$$\boldsymbol{x}^{(k+1)} = \boldsymbol{x}^{(k)} - \gamma_k F\Big(\boldsymbol{G}_1(\boldsymbol{x}^{(k)},\xi_1)\ldots\boldsymbol{G}_n(\boldsymbol{x}^{(k)},\xi_n)\Big) \tag{4}$$

[5] A step denotes an update in the model parameters.

where F denotes the *gradient aggregation rule* (GAR), and $\boldsymbol{G}_i(\boldsymbol{x}^{(k)},\xi_i)$ denotes the gradient estimate of worker i, using its own randomly drawn mini-batch ξ_i and the global model $\boldsymbol{x}$ at epoch k.

In the context of non-convex optimization, it is generally hopeless to try to minimize $Q(\boldsymbol{x})$. Instead, what can be proven is that the sequence of parameter vectors converges to a region around some $\boldsymbol{x}^*$ where $\boldsymbol{\nabla} Q(\boldsymbol{x}^*) = 0$, i.e, a flat region of the loss function (Bottou, 1998). Any gradient aggregation rule, F, which satisfies this convergence property despite the presence of f Byzantine workers, among the total of n workers, is called *weakly Byzantine-resilient*.

Strong Byzantine resilience. In high dimensional spaces (i.e., $d \gg 1$), and with a highly non-convex loss function (which is the case in modern machine learning (Haykin, 2009; Mallat, 2016)), weak Byzantine resilience may lead to models with poor performance in terms of prediction accuracy, as a Byzantine worker can fool a provably converging SGD rule by leveraging a *dimensional leeway* (El Mhamdi et al., 2018). More precisely, this worker can make the system converge, as guaranteed by its designers, but to a state with poor (as compared to the maximum possible one in a non-Byzantine environment) prediction accuracy.[6]

We define *strong Byzantine resilience* as the ability for a GAR, in addition to being weakly Byzantine-resilient, to select gradients that are (in each coordinate) in a distance of at most $\frac{1}{\sqrt{d}}$ from some correct gradient, despite the presence of f Byzantine workers among the total n workers. A more detailed analysis of strong Byzantine resilience is available (El-Mhamdi & Guerraoui, 2019).

Attacking a non-Byzantine resilient GARs such as averaging is easy. Attacking a GAR that ensures weak Byzantine resilience requires a powerful adversary, i.e., at least able to carry out the attack presented in (El Mhamdi et al., 2018). Our threat model (§3.1), both states sufficient requirements for such an adversary and allows its existence.

2.3 Algorithms

AGGREGATHOR relies on two algorithmic components: MULTI-KRUM (Blanchard et al., 2017) and BULYAN (El Mhamdi et al., 2018). The former rule requires that $n \geq 2f + 3$ and the second requires that $n \geq 4f + 3$.

[6] The intuition behind this issue relates to the so-called *curse of dimensionality*, a fundamental problem in learning: a square of unit 1 on each side has a diagonal of length $\sqrt{2}$. In dimension 3, the cube of unit 1 has a diagonal of length $\sqrt{3}$. In dimension d, the diagonal is of length $\sqrt{d}$. Given $d \gg 1$, points that differ by a distance of at most 1 in each direction end up being in a huge distance from each other.

Intuitively, the goal of MULTI-KRUM is to select the gradients that deviate less from the "majority" based on their relative distances. Given gradients $\boldsymbol{G}_1 \dots \boldsymbol{G}_n$ proposed by workers 1 to n respectively, MULTI-KRUM selects the m gradients with the smallest sum of scores (i.e., L2 norm from the other gradients) as follows:

$$(m) \underset{i \in \{1,\dots,n\}}{\arg\min} \sum_{i \to j} \|\boldsymbol{G}_i - \boldsymbol{G}_j\|^2 \tag{5}$$

where given a function $X(i)$, $(m)\arg\min(X(i))$ denotes the indexes i with the m smallest $X(i)$ values, and $i \to j$ means that $\boldsymbol{G}_j$ is among the $n - f - 2$ closest gradients to $\boldsymbol{G}_i$. BULYAN in turn takes the aforementioned m vectors, computes their coordinate-wise median and produces a gradient which coordinates are the average of the $m - 2f$ closest values to the median.

In (Blanchard et al., 2017), it was proven that Krum (i.e., MULTI-KRUM for $m = 1$) is weakly Byzantine-resilient. Yet, choosing $m = 1$ hampers the speed of convergence (Chen et al., 2018; Alistarh et al., 2018). MULTI-KRUM becomes practically interesting when we can chose the highest possible value for m to leverage all the workers (in the limit of no Byzantine workers, this value should be n).

In our Appendix, we answer the open question of Byzantine resilience for $m > 1$ and we prove that for $n \geq 2f + 3$ and any integer m s.t. $m \leq n - f - 2$: (1) MULTI-KRUM ensures weak Byzantine resilience against f failures, and (2) BULYAN ensures strong Byzantine resilience against f failures. As a consequence, AGGREGATHOR can safely be used with any value between $1 \leq m \leq n - f - 2$, and not only $m = 1$.

3 DESIGN OF AGGREGATHOR

Our goal is two-fold: First we target faster development and testing of robust, distributed training in TensorFlow; that essentially boils down to providing ease-of-use and modularity. Second we want to enable the deployment of Byzantine-resilient learning algorithms outside the academic environment.

3.1 Threat model and parameter server

We assume the standard synchronous parameter server model (Li et al., 2014), with the only dissimilarity being that $f < n$ of the n *workers* are controlled by an *adversary*. We refer to these workers as *Byzantine*. The goal of the adversary is to impair the learning, by making it converge to a state different from the one that would have been obtained if no adversary had stymied the learning process.

We assume the adversary, in the instance of the f cooperating Byzantine workers, has unbounded computational power, and arbitrarily fast communication channels between its f workers and with the parameter server. We assume an asynchronous network (see §3.3) and we assume that even Byzantine workers send gradients at each step[7]. We assume the adversary has access to the full training dataset B, and the gradients computed by each correct worker.

Finally, we assume, as in (El Mhamdi et al., 2018; Blanchard et al., 2017; Xie et al., 2018; Yin et al., 2018), that the parameter server is correct. This server could be reliable and implemented on a trustworthy machine, unlike workers that could be remote user machines in the wild. This server could also be implemented by using standard Byzantine-resilient state machine replication (Schneider, 1990; Chen et al., 2015; Castro et al., 1999).

3.2 Architecture and Byzantine resilience

AGGREGATHOR is a light framework (as shown in Figure 1) that handles the distribution of the training of a TensorFlow neural network *graph* over a cluster of machines. One of our main contributions is that this distribution is robust to Byzantine cluster nodes, in a proportion that depends on the GAR used.

In TensorFlow, Byzantine resilience cannot be achieved solely through the use of a Byzantine-resilient GAR. Indeed, TensorFlow allows any node in the cluster to execute arbitrary operations anywhere in the cluster. A single Byzantine worker could then continually overwrite the shared parameters[8] with arbitrary values. We overcome this issue in two steps: (a) by patching TensorFlow to make *tf.train.Server* instances belonging to the job named "ps" to discard remote graph definitions and executions, and (b) by using *in-graph replication*, where only the parameter server ("ps") builds the graph (Figure 2).

Pursuing our goal of ease-of-use and modularity, we provide the user of our framework with two tools; a *deploy* tool to deploy a cluster through SSH, and a *run* tool to launch a training session on the deployed cluster. Adding a new GAR or a new experiment boils down to (1) adding a python script to a directory, and (2) testing this new component; this consists in changing one or two command line parameters when calling the *run* tool. Cluster-wide device allocation, specifying which operations should run on which devices, is managed by our framework.

Byzantine resilience (using a complex GAR in our case) comes with cost which we quantify for some settings in §4.

[7] The default behavior of TensorFlow is to wait indefinitely for non-responding remote nodes, which is incompatible with asynchrony and Byzantine workers (not responding on purpose).

[8] This is actually how the distributed example of TensorFlow given on `https://www.tensorflow.org/deploy/distributed` works: each worker node keeps overwriting the shared parameters.

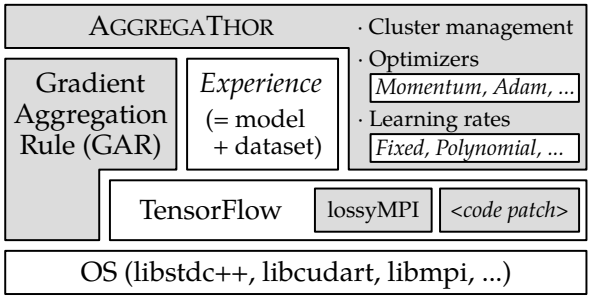


Figure 1. The components of AGGREGATHOR, and their layered relations with existing components. New components have a gray background. AGGREGATHOR acts as a light framework, that manages the deployment and execution of a *model training session* over a cluster of machines.

3.3 Communication layer

A Byzantine-resilient GAR at a high-level layer enables the usage of a fast but unreliable communication protocol at the low-level one. Using the vanilla TensorFlow averaging does not work while employing unreliable communication, because lost or shuffled coordinates/packets (of even one gradient) can lead to learning divergence. The most straightforward solution to guarantee convergence in this case, is to drop the whole gradient if at least one coordinate was lost (i.e., the packet containing the coordinate was lost). We expect that such a solution will delay convergence especially in networks with high loss ratios. To avoid dropping the whole gradient in such a case, one can implement a variant of averaging which we call *selective averaging*. In this GAR, the lower layer replaces the lost coordinates with a special value (e.g. *NaN*) while the GAR layer ignores these coordinates while averaging. We expect this method to be faster than the first one. A third solution would be simply to use AGGREGATHOR on top, and put random values at the lost coordinates. Not caring about what happens at the low-level layer would not be harmful, as the Byzantine-resilient GAR on top guarantees convergence (as long as the unreliable communication is deployed only at (up to) f links). Comparing the last two proposed solutions, it is worth noting that using the *selective averaging* model requires a special care for out-of-order packets. A sequence number should define the correct position of each packet so that the received packets are correctly put in their positions (in the gradient). Otherwise, learning convergence is not guaranteed. However, using AGGREGATHOR does not require sending the sequence number because this GAR does not have any assumptions on what is delivered at the lower-level layers.

TensorFlow does not support UDP and hence, we modify its underlying networking layer to support a fast but unreliable communication protocol, which we call *lossyMPI*, alongside those already supported, e.g., gRPC, RDMA, MPI. LossyMPI is devised by modifying the MPI communication endpoints to employ UDP sockets. At the receiving endpoint, we implement the aforementioned GARs to recoup the lost coordinates and guarantee convergence. To use UDP, we also implement a reliability scheme for metadata (accompanying gradients) and packets ordering.

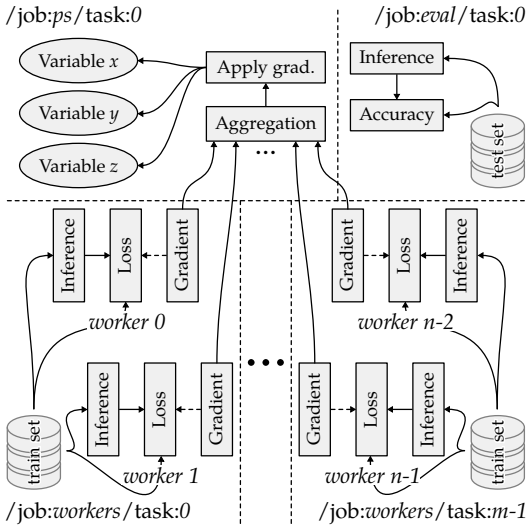


Figure 2. High-level components and execution graph. Each gray rectangle represents a group of `tf.Operation`, and each plain arrow represents a `tf.Tensor`. The sub-graph between the gradients to the variables corresponds to Equation 4. For readability purpose, the tensors from the variables to each "Inference" and "Gradient" groups of operations have not been represented.

4 EVALUATION OF AGGREGATHOR

We evaluate the performance of AGGREGATHOR following a rather standard methodology in the distributed ML literature. In particular, we consider the image classification task due to its wide adoption as a benchmark for distributed ML literature (Chilimbi et al., 2014; Abadi et al., 2016a; Zhang et al., 2017).

4.1 Evaluation Setup

We present the details of the configuration, benchmarks, and methods we employ for our evaluation. For clarity and for the rest of this section, we will refer with MULTI-KRUM to the deployment of AGGREGATHOR with the GAR being MULTI-KRUM and with BULYAN to the deployment of AGGREGATHOR with the GAR being BULYAN.

Platform. Our experimental platform is Grid5000 (g5k). Unless stated otherwise, we employ 20 nodes from the same cluster, each having 2 CPU (Intel Xeon E5-2630) with 8 cores, 128 GiB RAM and 10 Gbps Ethernet.

Dataset. We use the CIFAR-10 dataset (cif), a widely used dataset in image classification (Srivastava et al., 2014; Zhang et al., 2017), which consists of 60,000 colour images in 10 classes. We perform min-max scaling as a preprocessing step for the input features of the dataset. We employ a convolutional neural network with a total of 1.75M parameters as shown in Table 1. We have implemented the same model with PyTorch to be compatible with Draco.

Table 1. CNN Model parameters.

	Input	Conv1	Pool1	Conv2	Pool2	FC1	FC2	FC3
Kernel size	32×32×3	5×5×64	3×3	5×5×64	3×3	384	192	10
Strides		1×1	2×2	1×1	2×2			

Evaluation metrics. We evaluate the performance of AGGREGATHOR using the following standard metrics.

Throughput. This metric measures the total number of gradients that the aggregator receives per second. The factors that affect the throughput is the time to compute a gradient, the communication delays (worker receives the model and sends the gradient) and the idle time of each worker. The idle time is determined by the overhead of the aggregation at the server. While the server performs the aggregation and the descent, the workers wait (synchronous training).

Accuracy. This metric measures the top-1 cross-accuracy: the fraction of correct predictions among all the predictions, using the *testing* dataset (see below). We measure accuracy both with respect to time and model updates.

Evaluation scheme. To cross-validate the performance, we split the dataset into *training* and *test sets*. The dataset includes 50,000 training examples and 10,000 test examples. Note that, if not stated otherwise, we employ an RMSprop optimizer (Tieleman & Hinton, 2012) with a fixed initial learning rate of 10^{-3} and a mini-batch size of 100.

We split our 20 nodes into $n = 19$ workers and 1 parameter server. If not stated otherwise, we set $f = 4$ given that BULYAN requires $n \geq 4f + 3$.

We employ the best (in terms of convergence rate) combination of other hyper-parameters for the deployment of Draco. For example, we use the repetition method because it gives better results than the cyclic one. Also, we use the reversed gradient adversary model with the same parameters recommended by the authors and a momentum of 0.9.

4.2 Non-Byzantine Environment

In this section, we report on the performance of our framework in a non-Byzantine distributed setup. Our baseline is *vanilla* TensorFlow (TF) deployed with the built-in averaging GAR: *tf.train.SyncReplicasOptimizer*. We compare TF against AGGREGATHOR using (a) MULTI-KRUM, (b) BULYAN, (c) an alternative Byzantine-resilient median-based algorithm (Xie et al., 2018) (*Median*) implemented as a new GAR in our framework, and (d) the basic gradient averaging GAR (*Average*). We also report on the performance of (e) Draco.

Overhead in terms of convergence time. In Figure 3(a), TensorFlow reaches 50% of its final accuracy in 3 minutes and 9 seconds, whereas MULTI-KRUM and BULYAN are respectively 19% and 43% slower for reaching the same accuracy. Our framework with *Average* leads to a 7% slowdown compared to the baseline. The *Median* GAR, with a mini-batch size of $b = 250$, converges as fast as the baseline (model update-wise), while with $b = 20$, *Median* prevents convergence to a model achieving baseline accuracy.

We identify two separate causes for the overhead of AGGREGATHOR. The first is the *computational overhead* of carrying out the Byzantine-resilient aggregation rules. The second cause is the inherent *variance increase* that Byzantine-resilient rules introduce compared to *Average* and the baseline. This is attributed to the fact that MULTI-KRUM, BULYAN and *Median* only use a fraction of the computed gradients; in particular *Median* uses only one gradient. Increasing the variance of the gradient estimation is a cause of convergence slowdown (Bottou, 1998). Since even *Median* converges as fast as the baseline with $b = 250$, the respective slowdowns of 19% and 43% for MULTI-KRUM and BULYAN correspond *only* to the computational overhead. The practitioner using AGGREGATHOR does not need to increase the mini-batch size to achieve baseline final accuracy (Figure 3(d)).

Although Draco reaches the same final accuracy, the time to reach the model's maximal accuracy is slower than with our TensorFlow-based system. We attribute this mainly to the fact that Draco requires $2f + 1$ times more gradients to be computed than our system before performing a step.

We decompose the average latency per epoch to assess the effect of the aggregation time on the overhead of AGGREGATHOR against TensorFlow. We employ the same setup as in Figure 3(a).

Figure 4 shows that the aggregation time accounts for 35%, 27% and 52% of run times of Median, MULTI-KRUM, and BULYAN respectively. These ratios do not depend on the variance of the aggregated gradients, but solely on the gradient computation time: a larger/more complex model would naturally make these ratios decrease (i.e., the *relative cost* of Byzantine resilience would decrease). See Figure 5.

Impact of f on scalability. We measure the scalability of AGGREGATHOR with respect to TensorFlow for two models: the CNN that we use throughout the evaluation and a significantly larger one, ResNet50. Figure 5(a) shows that the throughput of all TensorFlow-based systems with up to

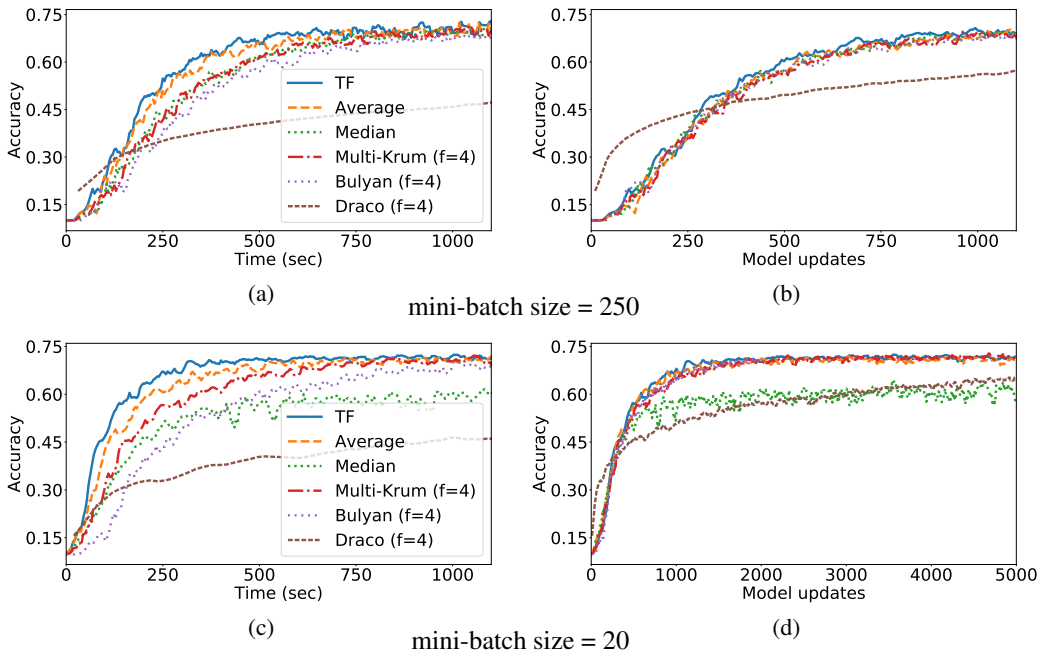


Figure 3. Overhead of AGGREGATHOR in a non-Byzantine environment.

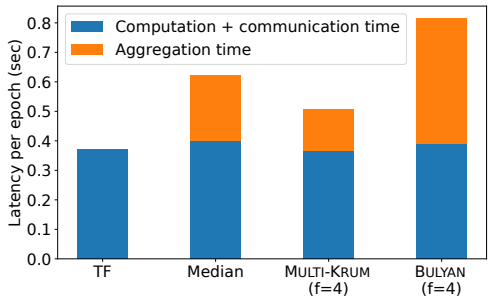


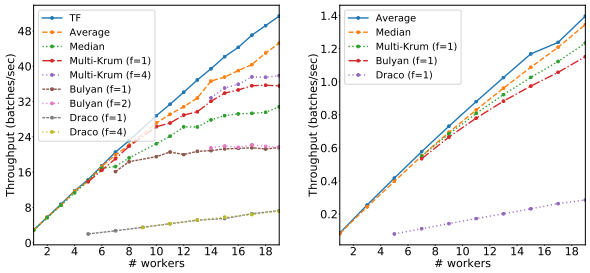
Figure 4. Latency breakdown.

6 workers is the same. From this point on, the larger the number of workers, the larger the deviation between the Byzantine-resilient algorithms and TensorFlow. The reason behind this behavior is the fact that an increase in the number of workers introduces a larger overhead to the aggregation of a Byzantine-resilient algorithm (logic to ensure Byzantine resilience) than simple averaging. The more expensive the logic, the bigger this difference. For example, BULYAN scales poorly for this setup. This is confirmed in Figure 5(b) where the gradient computation is significantly more costly than gradient aggregation. This allows MULTI-KRUM and BULYAN to have better scalability.

Figure 5(a) confirms that the higher the declared f the higher the throughput. This may appear counter-intuitive (resilience against more failures provides a performance benefit) but is the direct outcome of the design of the algorithmic components of AGGREGATHOR. Since $m = n - f - 2$, the higher f the fewer *iterations* for BULYAN (El Mhamdi et al., 2018) and the fewer the neighbors for MULTI-KRUM (Blanchard et al., 2017). Moreover, for a larger value of f, these algorithms become more selective for the gradients that will be averaged. It is however very important to highlight that non-convex optimization is a notably complex problem that might take advantage of more variance to converge faster (as we discuss in our Appendix). Therefore, anticipating faster convergence for a larger value of f does not always hold, i.e., larger throughput does not always lead to faster convergence (Bottou, 1998). In conclusion, there exists a trade-off between the update throughput and the quality of each update that is partially controlled by the choice of f.

Draco is always at least one order of magnitude slower than the TensorFlow-based systems. This low throughput limits its scalability. An interesting observation here is that changing the number of Byzantine workers does not have a remarkable effect on the throughput. This is attributed to the method Draco uses to tolerate Byzantine behavior which is linear in n (Chen et al., 2018), thus the performance is not affected by changing f.

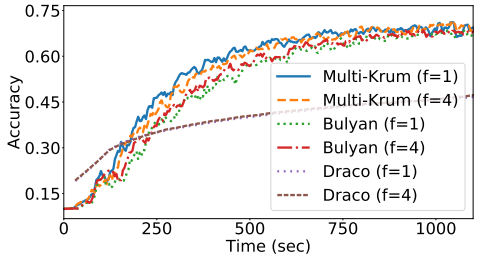
Impact of f on convergence. We show the effect of the choice of f in a non-Byzantine environment. Figure 6(a) shows that the larger value of f triggers a slightly slower convergence for MULTI-KRUM and slightly faster convergence for BULYAN. This is the direct consequence of the aforementioned trade-off. The throughput of MULTI-KRUM is boosted more than the throughput of BULYAN for the same increase on f (from 1 to 4). Therefore, in the case of BULYAN, the faster model updates



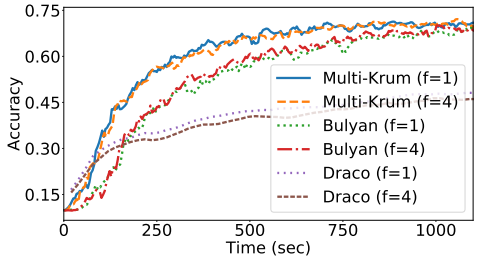
(a) CNN (b) ResNet50

Figure 5. Throughput comparison.

compensate for the additional noise whereas in the case of MULTI-KRUM the throughput boost is not enough. For a smaller mini-batch size (Figure 6(b)) the behaviour is similar but the impact of f is smaller. This is because the mini-batch size is the second (the first is f) important parameter that affects the trade-off between update throughput and quality of each update (§2.1).



(a) mini-batch size = 250



(b) mini-batch size = 20

Figure 6. Impact of f on convergence.

Cost analysis. Our empirical results are consistent with the complexity of the algorithmic components of AGGREGATHOR. The model update time complexity[9] of both MULTI-KRUM and BULYAN is $\mathcal{O}(n^2 d)$. This is essentially the same as a baseline GAR-based SGD algorithm, i.e., averaging[10] when $d \gg n$ (valid assumption for modern ML). BULYAN induces an additional overhead of $\mathcal{O}(nd)$ (coordinate-wise median) on top of $n - 2f$ executions of MULTI-KRUM, leading to a total model update complexity of $\mathcal{O}(nd + f \cdot nd) = \mathcal{O}(n^2 d)$. We note that $\mathcal{O}(n^2 d)$ is a common bound on the complexity per round for weakly Byzantine-resilient SGD algorithms (Blanchard et al., 2017; El Mhamdi et al., 2018; Su, 2017). AGGREGATHOR is strongly Byzantine-resilient with the same complexity.

Baseline averaging SGD requires $\mathcal{O}(\frac{1}{\sqrt{nb}})$ steps to converge. In other words SGD goes as fast as permitted by the square root of the total number of samples used per step. The more samples, the lower the variance and the better the gradient estimation.

Everything else being equal (mini-batch sizes, smoothness of the cost function etc), and in the absence of Byzantine workers, the number of steps required for AGGREGATHOR to converge is $\mathcal{O}(\frac{1}{\sqrt{m}})$. The higher the value of m the fewer steps required for convergence. The temptation is then to increase m to the highest value possible, so that there are fewer steps required to converge. We derive the maximum possible value for m that ensures weak Byzantine resilience, $\tilde{m} = n - 2f - 2$[11], in our Appendix. We also prove that this will induce a slowdown of $\Omega(\sqrt{\frac{\tilde{m}}{n}})$ (always computed as the ratio between AGGREGATHOR and averaging), in the absence of Byzantine workers. In other words, $\tilde{m}$ enables the fastest convergence while still ensuring the safety requirement of Byzantine resilience.

4.3 Byzantine Environment

We now report on our evaluation of AGGREGATHOR in a distributed setting with Byzantine workers. We first report on two forms of *weak* Byzantine behavior, namely corrupted data and dropped packets. Then we discuss the effect of a *stronger* form of Byzantine behavior, drawing the line between MULTI-KRUM and BULYAN.

Corrupted data. Figure 7 shows that for a mini-batch size of 250, the convergence behavior of AGGREGATHOR is similar to the ideal one (TensorFlow in a non-Byzantine environment). We thus highlight the importance of Byzantine resilience even for this "mild" form of Byzantine behavior (only one worker sends corrupted data) to which TensorFlow is intolerant (TensorFlow diverges).

Dropped packets. We evaluate the impact of using unreliable communication between the parameter server and f (Byzantine) workers. We also evaluate the performance of alternatives we proposed in §3.3 to tolerate the lost, malformed, and out-of-order packets. For this experiment we

[9] We refer to the worst-case time complexity.

[10] Averaging is not Byzantine-resilient.

[11] A higher value $(n - f - 2)$ for m can be selected if only weak Byzantine resilience is required and only MULTI-KRUM is used.

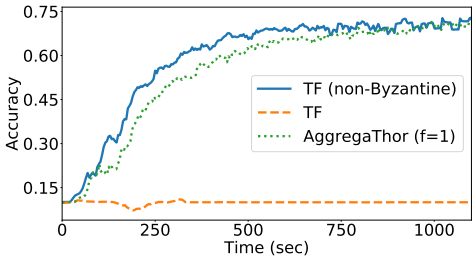


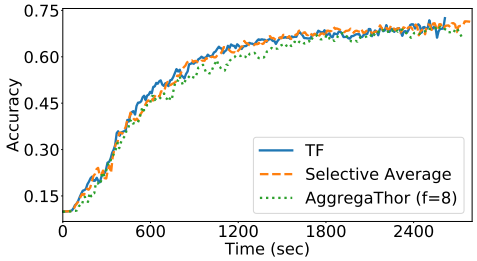
Figure 7. Impact of malformed input on convergence.

set f to the maximum possible value given our 19 workers, i.e., we set f to 8 (assuming MULTI-KRUM). For simplicity, we employ unreliable communication only for the gradient transfer[12].

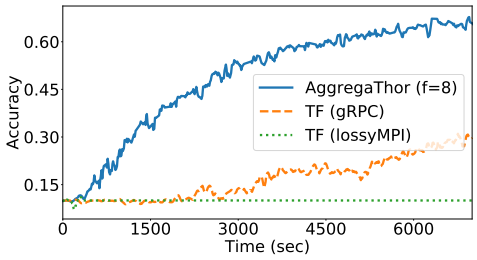
We assess the effect of unreliable communication in a lossy environment by introducing additional (to the existing ones by the network) network packet drops via the Linux tc tool. We evaluate the performance of AGGREGATHOR in the absence of additional packet drops (0% loss) and in the presence of a drop rate of 10%. This order of magnitude for the drop ratio, although high for a data-center environment, can be realistic in a WAN environment (Kumar et al., 2015) for distributed machine learning (Hsieh et al., 2017).

Figure 8(a)[13] shows the performance of the three solutions proposed to tolerate unreliability of the communication layer (§3.3). The three solutions achieve almost the same performance. This highlights the advantage of using UDP as it mitigates the performance lost by Byzantine resilience. In this environment where no packet loss exists, dropping the whole gradient (while using vanilla TensorFlow) does not have remarkable effect on the learning convergence. We expect to see a delayed convergence for such an algorithm in an environment with a higher loss ratio.

Figure 8(b) shows the advantage of using UDP in a lossy environment. It depicts that AGGREGATHOR over lossyMPI converges to 30% accuracy more than 6 times faster than TensorFlow over gRPC, under an artificial 10% drop rate. The main reason behind this big difference in the convergence speed is that the links that employ lossyMPI (between the server and the Byzantine workers) use a rapid mechanism to address the packet drops, i.e., they deliver corrupted messages to which AGGREGATHOR is tolerant. The convergence time for both systems is one order of magnitude larger compared to the environment with no artificial drops. We believe this performance drop is induced by TCP reducing (halving) its transmission rate following packet losses. Finally, Figure 8(b) confirms the divergence of TensorFlow, which is non Byzantine-resilient, over lossyMPI.



(a) 0% drop rate



(b) 10% drop rate

Figure 8. Impact of dropped packets on convergence.

Byzantine gradients. The cost of attacking a non-Byzantine resilient GARs (such as averaging) is the cost for the computation of an estimate of the gradient, i.e., can be done in $O(nd)$ operations per round by a Byzantine worker. This cost is the same as the aggregation cost of the server per epoch.

To attack weakly Byzantine-resilient GARs however, such as MULTI-KRUM, one needs to find a legitimate but harmful vector. A harmful vector is a vector that will (i) be selected by a weakly Byzantine-resilient GAR, and (ii) triggers a poor convergence, i.e., a convergence to an optimum that makes the performance of the model (e.g., in terms of accuracy) low in comparison with the one achieved when learning with no Byzantine workers. An attacker must thus first collect the vector of every correct worker (before they reach the server), and solve an optimization problem (with linear regression) to approximate this harmful but legitimate (selected by weakly Byzantine-resilient GAR) vector. If the desired quality of the approximation is ϵ, the Byzantine worker would need at least $\Omega(\frac{nd}{\epsilon})$ operation to reach it with regression. This is a tight lower bound for a regression problem in d dimensions with n vectors (Haykin, 2009). In practice, if the required precision is in the order of 10^{-9}, a total of 100 workers and a model of dimension 10^9 would require a prohibitive cost for the attack ($\approx 10^{20}$ operations to be done in each step by the attacker).

To summarize, weak Byzantine resilience can be enough as a practical solution against attackers whose resources

[12]Our setup can be easily extended to support an unreliable communication for the model transfer without any impact on the conclusions of our evaluation.

[13]TF here drops corrupted gradients as described in §3.3.

are comparable to the ones of the server. If that is the expected setting, we could switch-off BULYAN and use only MULTI-KRUM. However, strong Byzantine resilience, i.e., AGGREGATHOR combining both MULTI-KRUM and BULYAN, remains the provable solution against attackers with significant resources (El Mhamdi et al., 2018).

5 RELATED WORK

Several weakly Byzantine-resilient algorithms have been proposed as an improvement of the workhorse SGD component in the synchronous non-convex setting. Krum (Blanchard et al., 2017) employed a median-like aggregation rule. (Yin et al., 2018) proposed a median-based and a mean-based aggregation rules (Equation 4). (Xie et al., 2018) evaluated three other median-based aggregation rules under different practical attack scenarios. (Su, 2017) presented a combination of a median-based over a mean-based aggregation rule. A quorum-based aggregation approach was recently proposed in (Alistarh et al., 2018), achieving optimal convergence rates but suitable only for convex machine learning.

Following a different direction, Draco (Chen et al., 2018) has been the first framework to address the scalability of SGD through redundant gradient computations combined with a specific encoding scheme. In fact Draco is also strongly Byzantine-resilient following our definition, and has the advantage of requiring only $2f+1$ workers (instead of $4f+3$ for BULYAN). It has however a serious practical concern: it can only be used in settings where workers need (at least) an agreement on the ordering of the dataset so that the coding scheme can be agreed on. This violates critical privacy concerns in distributed learning and does not allow learning on private data. For instance, their algorithmic redundancy scheme requires a comparison between gradients sum provided by different workers, for this comparison to be meaningful, the server needs the incoming gradients to be computed on similar datapoints, therefore hampering any possible use in private and local datasets. AGGREGATHOR in turn only requires the workers to be drawing data independently and identically distributed (but not the same datapoints). Additionally, as pointed out by the authors (Chen et al., 2018), the encoding and decoding time of Draco can be several times larger than the computation time of ordinary SGD. AGGREGATHOR avoids this overhead along with the redundant computations.

The first algorithmic component of AGGREGATHOR, namely MULTI-KRUM, is a generalization (adaptive version) of Krum (Blanchard et al., 2017) in the sense that we employ a multi-aggregation rule with a dynamic size based on the value of f. In short, we enable the server to leverage $m > 1$ workers in each step. The idea was mentioned in (Blanchard et al., 2017) but the proof of (weak) Byzantine resilience was left open. We answer this open question in the Appendix and prove weak Byzantine resilience for any integer m such that $m \leq n-f-2$ as well as the resulting strong Byzantine resilience of AGGREGATHOR when $m \leq n-2f-2$.

6 CONCLUDING REMARKS

We built AGGREGATHOR, a Byzantine-resilient framework, on top of TensorFlow without adding any constraints to the application development, i.e., AGGREGATHOR exposes the same APIs as TensorFlow. We also corrected an inherent vulnerability of TensorFlow in the Byzantine setting. The overhead of AGGREGATHOR over TensorFlow is moderate when there are no Byzantine failures. In fact, we have also shown that AGGREGATHOR could be viewed as a performance booster for TensorFlow, as it enables the use of an unreliable (and faster) underlying communication protocol, namely UDP instead of TCP, when running through a saturated network.

While designing AGGREGATHOR, we followed the parameter server model (Li et al., 2014) and assumed the server is reliable while workers are not. This model has been considered for most theoretical analysis of Byzantine-resilient SGD (Blanchard et al., 2017; Damaskinos et al., 2018; El Mhamdi et al., 2018; Chen et al., 2018; Alistarh et al., 2018; Xie et al., 2018). An orthogonal problem that should be investigated is the setting where the owner of the system does not trust their servers. In this case, a server could be made Byzantine-resilient using some state machine replication scheme (Schneider, 1990; Castro et al., 1999; Sen et al., 2010; Cowling et al., 2006; Chen et al., 2015). Essentially, each worker could communicate with the replicas of the server and use the model that has been sent by 2/3 of the replicas. Since the computation in the server (GAR and model update) is deterministic, the correct servers will propose identical models to the workers. Although at first glance simple, we believe that the interplay between the specificity of gradient descent and the state machine replication approach might end up challenging to achieve efficiently.

ACKNOWLEDGMENT

This work has been supported in part by the Swiss National Science Foundation (FNS grant 200021_182542/1 and FNS grant 200021_ 169588/TARBDA).

Experiments presented in this paper were carried out using the Grid'5000 testbed, supported by a scientific interest group hosted by Inria and including CNRS, RENATER and several Universities as well as other organizations (see https://www.grid5000.fr).

REFERENCES

Cifar dataset. https://www.cs.toronto.edu/~kriz/cifar.html.

Grid5000. https://www.grid5000.fr/.

Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., Devin, M., Ghemawat, S., Irving, G., Isard, M., et al. Tensorflow: A system for large-scale machine learning. In *OSDI*, 2016a.

Abadi, M., Chu, A., Goodfellow, I., McMahan, H. B., Mironov, I., Talwar, K., and Zhang, L. Deep learning with differential privacy. In *SIGSAC*, pp. 308–318, 2016b.

Alistarh, D., Allen-Zhu, Z., and Li, J. Byzantine stochastic gradient descent. In *Neural Information Processing Systems, to appear*, 2018.

Biggio, B. and Laskov, P. Poisoning attacks against support vector machines. In *ICML*, 2012.

Biggio, B. and Roli, F. Wild patterns: Ten years after the rise of adversarial machine learning. *arXiv preprint arXiv:1712.03141*, 2017.

Blanchard, P., El Mhamdi, E. M., Guerraoui, R., and Stainer, J. Machine learning with adversaries: Byzantine tolerant gradient descent. In *Neural Information Processing Systems*, pp. 118–128, 2017.

Bottou, L. Online learning and stochastic approximations. *Online learning in neural networks*, 17(9):142, 1998.

Castro, M., Liskov, B., et al. Practical Byzantine fault tolerance. In *OSDI*, volume 99, pp. 173–186, 1999.

Chen, A., Xiao, H., Haeberlen, A., and Phan, L. T. X. Fault tolerance and the five-second rule. In *HotOS*, 2015.

Chen, L., Wang, H., Charles, Z., and Papailiopoulos, D. Draco: Byzantine-resilient distributed training via redundant gradients. In *International Conference on Machine Learning*, pp. 902–911, 2018.

Chilimbi, T. M., Suzue, Y., Apacible, J., and Kalyanaraman, K. Project adam: Building an efficient and scalable deep learning training system. In *OSDI*, volume 14, pp. 571–582, 2014.

Cowling, J., Myers, D., Liskov, B., Rodrigues, R., and Shrira, L. Hq replication: A hybrid quorum protocol for Byzantine fault tolerance. In *OSDI*, pp. 177–190. USENIX Association, 2006.

Damaskinos, G., El Mhamdi, E. M., Guerraoui, R., Patra, R., Taziki, M., et al. Asynchronous byzantine machine learning (the case of sgd). In *ICML*, pp. 1153–1162, 2018.

Dean, J., Corrado, G., Monga, R., Chen, K., Devin, M., Mao, M., Senior, A., Tucker, P., Yang, K., Le, Q. V., et al. Large scale distributed deep networks. In *Neural Information Processing Systems*, pp. 1223–1231, 2012.

Diakonikolas, I., Kamath, G., Kane, D. M., Li, J., Ankur, M., and Alistair, S. Robustly learning a gaussian: Getting optimal error, efficiently. *arXiv preprint arXiv:1704.03866*, 2017.

El-Mhamdi, E.-M. and Guerraoui, R. Fast and secure distributed learning in high dimension. *arXiv preprint arXiv:*, 2019.

El Mhamdi, E. M., Guerraoui, R., and Rouault, S. The hidden vulnerability of distributed learning in Byzantium. In Dy, J. and Krause, A. (eds.), *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pp. 3521–3530, Stockholmsmässan, Stockholm Sweden, 10–15 Jul 2018. PMLR. URL http://proceedings.mlr.press/v80/mhamdi18a.html.

Goyal, P., Dollár, P., Girshick, R., Noordhuis, P., Wesolowski, L., Kyrola, A., Tulloch, A., Jia, Y., and He, K. Accurate, large minibatch sgd: training imagenet in 1 hour. *arXiv preprint arXiv:1706.02677*, 2017.

Gunawi, H. S., Suminto, R. O., Sears, R., Golliher, C., Sundararaman, S., Lin, X., Emami, T., Sheng, W., Bidokhti, N., McCaffrey, C., Grider, G., Fields, P. M., Harms, K., Ross, R. B., Jacobson, A., Ricci, R., Webb, K., Alvaro, P., Runesha, H. B., Hao, M., and Li, H. Fail-slow at scale: Evidence of hardware performance faults in large production systems. In *FAST*, pp. 1–14, 2018.

Haykin, S. S. *Neural networks and learning machines*, volume 3. Pearson Upper Saddle River, NJ, USA:, 2009.

Hsieh, K., Harlap, A., Vijaykumar, N., Konomis, D., Ganger, G. R., Gibbons, P. B., and Mutlu, O. Gaia: Geo-distributed machine learning approaching lan speeds. In *NSDI*, pp. 629–647, 2017.

Jaggi, M., Smith, V., Takác, M., Terhorst, J., Krishnan, S., Hofmann, T., and Jordan, M. I. Communication-efficient distributed dual coordinate ascent. In *Neural Information Processing Systems*, pp. 3068–3076, 2014.

Kumar, A., Jain, S., Naik, U., Raghuraman, A., Kasinadhuni, N., Zermeno, E. C., Gunn, C. S., Ai, J., Carlin, B., Amarandei-Stavila, M., et al. Bwe: Flexible, hierarchical bandwidth allocation for wan distributed computing. In *SIGCOMM*, volume 45, pp. 1–14. ACM, 2015.

Li, M., Andersen, D. G., Park, J. W., Smola, A. J., Ahmed, A., Josifovski, V., Long, J., Shekita, E. J., and Su, B.-Y. Scaling distributed machine learning with the parameter server. In *OSDI*, volume 1, pp. 3, 2014.

Mallat, S. Understanding deep convolutional networks. *Phil. Trans. R. Soc. A*, 374(2065):20150203, 2016.

Meng, X., Bradley, J., Yavuz, B., Sparks, E., Venkataraman, S., Liu, D., Freeman, J., Tsai, D., Amde, M., Owen, S., et al. Mllib: Machine learning in apache spark. *JMLR*, 17(1):1235–1241, 2016.

Papernot, N., McDaniel, P., Goodfellow, I., Jha, S., Celik, Z. B., and Swami, A. Practical black-box attacks against machine learning. In *Asia Conference on Computer and Communications Security*, pp. 506–519, 2017.

Recht, B., Re, C., Wright, S., and Niu, F. Hogwild: A lock-free approach to parallelizing stochastic gradient descent. In *Neural Information Processing Systems*, pp. 693–701, 2011.

Schneider, F. B. Implementing fault-tolerant services using the state machine approach: A tutorial. *CSUR*, 22(4): 299–319, 1990.

Sen, S., Lloyd, W., and Freedman, M. J. Prophecy: Using history for high-throughput fault tolerance. In *NSDI*, pp. 345–360, 2010.

Shokri, R. and Shmatikov, V. Privacy-preserving deep learning. In *SIGSAC*, pp. 1310–1321, 2015.

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. Dropout: A simple way to prevent neural networks from overfitting. *JMLR*, 15(1):1929–1958, 2014.

Su, L. *Defending distributed systems against adversarial attacks: consensus, consensus-based learning, and statistical learning*. PhD thesis, University of Illinois at Urbana-Champaign, 2017.

Tieleman, T. and Hinton, G. Lecture 6.5–rmsprop: Divide the gradient by a running average of its recent magnitude. COURSERA: Neural Networks for Machine Learning, 2012.

Xie, C., Koyejo, O., and Gupta, I. Generalized Byzantine-tolerant sgd. *arXiv preprint arXiv:1802.10116*, 2018.

Yin, D., Chen, Y., Ramchandran, K., and Bartlett, P. Byzantine-robust distributed learning: Towards optimal statistical rates. *arXiv preprint arXiv:1803.01498*, 2018.

Zhang, H., Zheng, Z., Xu, S., Dai, W., Ho, Q., Liang, X., Hu, Z., Wei, J., Xie, P., and Xing, E. P. Poseidon: An efficient communication architecture for distributed deep learning on GPU clusters. In *USENIX ATC*, pp. 181–193, 2017.

A. Artifact Appendix

A.1 Abstract

A.2 Artifact check-list (meta-information)

- **Program: TensorFlow (1.10.1), Python (3.5+)**
- **Compilation: C++11**
- **Data set: MNIST, CIFAR–10, ImageNet (optional), ...**
- **Run-time environment: Debian GNU/Linux (or equivalent)**
- **Hardware: G5k (or equivalent), see Section A.3.2**
- **Metrics: (time to reach some) top–1 cross–accuracy**
- **How much disk space required (approximately)?: <1 MB (code only)**
- **How much time is needed to prepare workflow (approximately)?: a few minutes to a few hours (several optional components, some demanding a rebuild of TensorFlow)**
- **How much time is needed to complete experiments (approximately)?: two minutes (see Section A.6) to a few days (for all the experiments of the main paper)**
- **Publicly available?: Yes**
- **Code licenses (if publicly available)?: MIT**

A.3 Description

A.3.1 How delivered

Two open–sourced versions:

- `AggregaThor.zip`, reviewed and obtained the ACM badges. `https://doi.org/10.5281/zenodo.2548779`.
- `LPD-EPFL/AggregaThor`, public git repository, latest version. `https://github.com/LPD-EPFL/AggregaThor`

A.3.2 Hardware dependencies

No hardware dependencies for CPU nodes. GPU nodes must be CUDA–compatible. To reproduce the experiments in the same conditions, use Grid5000 (`https://www.grid5000.fr`). Nevertheless, any recent consumer–grade computer enables basic checks.

A.3.3 Software dependencies

- TensorFlow (1.10.1)
- Python (3.5+)

A.3.4 Datasets

We employ `MNIST` and `CIFAR-10` in our experiments. `MNIST` is automatically downloaded by running our code. `CIFAR-10` has to be installed manually. The setup procedure should simply consist in **1.** cloning `tensorflow/models` from GitHub, **2.** download the dataset (1 command) and **3.** updating a *symlink*. See Section A.4.1.

A.4 Installation

Our implementation of AggregaThor consists of Python scripts and C++11 sources. Compilation of the native source code is carried out automatically. We refer to the AggregaThor/README.md file inside our zip file (A.3.1) for installation and usage instructions.

Our experiments have been run using TensorFlow 1.10.1. The installation steps are available at `https://www.tensorflow.org/install/`. See A.4.2 for installing the optional patches.

A.4.1 TensorFlow slim

AggregaThor has one recommended (but not necessary) dependency: "slim" (`https://github.com/tensorflow/models/tree/master/research/slim`). You would need to clone (or download) this dependency, and modify the following *symlinks*:

- (recommended) AggregaThor/experiments/slim_package → *path/to*/research/slim
- (recommended) AggregaThor/experiments/slim_datasets/cifar10 → *path/to*/cifar10/tfrecord
- (optional) AggregaThor/experiments/slim_datasets/imagenet → *path/to*/imagenet/tfrecord

Please note that the dataset must be in the *tfrecord* format. Please see `https://github.com/tensorflow/models/tree/master/research/slim#Data` for setting up these datasets.

As a side note, the directory AggregaThor/external/slim contains *optional* modifications for "slim". The sub–directory structure matches the one of "slim". The files found there are a modified version of the files found at in the respective sub–directory , which sole purpose is to add variants of the *ResNet* model family.

A.4.2 TensorFlow patches

Installation is necessary only for reproducing the udp-related experiments. Otherwise this step can be skipped. We refer to the AggregaThor/tf_patches/README.md file inside our zip file for detailed instruction.

A.5 Experiment workflow

We recommend first following AggregaThor/README.md, and the sections "Local deployment" and "Distributed deployment".

In particular, right after having *inflated* the provided ZIP file and installed TensorFlow (Section A.4), the command from "Local deployment" should work (basically, no error message *in red*). This provides to the reviewer a quick way to check the proper setup of TensorFlow and Python, if needed. For convenience, we copy the aforementioned command below:

```
$ python3 runner.py
    --server '{"local": ["127.0.0.1:7000"]}'
    --ps-job-name local
    --wk-job-name local
    --ev-job-name local
    --experiment mnist
    --learning-rate-args
          initial-rate:0.05
    --aggregator average
    --nb-workers 4
    --reuse-gpu
    --max-step 10000
    --evaluation-period -1
    --checkpoint-period -1
    --summary-period -1
    --evaluation-delta 100
    --checkpoint-delta -1
    --summary-delta -1
    --no-wait
```

Each experiment is entirely controlled by the command line of AggregaThor/runner.py. The parameters of this script are documented in AggregaThor/README.md and in Section A.7.

The textual output of AggregaThor/runner.py uses intuitive *color codes*. Main execution steps are reported *in green*, while most other informational lines are written *in blue*. Warnings (i.e. non fatal errors) are reported *in yellow*. These often provide useful information to the user, e.g., why an experiment cannot be used (one would see some of these messages when using command from "Local deployment" before having installed "slim"). Fatal errors are reported *in red*. Our code raises several kinds of exceptions to signal fatal conditions. An explanatory string is always provided.

There is no script that reproduces the full set of experiments in the paper. We provide a template script (experiments.sh) that merely executes the two commands (`deploy.py` then `runner.py`) needed to carry out any distributed experiment. See Section A.7.

A.6 Evaluation and expected result

For the command line given in sections "Local deployment" and "Distributed deployment" of AggregaThor/README.md, both should terminate on a recent consumer–grade computer in less than 2 minutes. One one *Intel(R) Xeon(R) CPU E5-2680 v2 @ 2.80GHz*, the command given in Section A.5 terminates in less than a minute, with a top–1 cross–accuracy around 97%.

You would need to run on Grid5000 to obtain the same performance results of the main paper. Nevertheless, you should be able to obtain very similar results using any equivalent platform. Please see the main paper for the specifications of the machines we used.

A.7 Experiment customization

As written above, each experiment is entirely controlled by the command line of AggregaThor/runner.py. We list below useful options, and the *hyperparameters* each of these options controls:

`--experiment`
: Experiment (i.e. model + dataset) to train/evaluate. The list may be limited if "slim" has not been (properly) installed. Leave this field empty (i.e. put `""`) to get the list of available experiments.

`--experiment-args`
: A experiment–dependant space–separated set of strings. The usual format for each of these string is: *property*`:`*value*, e.g. `batch-size:32`. Since these arguments are experiment–dependant, you can find a per–experiment complete list in the respective sources files, in AggregaThor/experiments/...

... mnist.py:108
... cnnet.py:117
... slim.py:100

`--aggregator`
: Gradient Aggregation Rule (GAR) to use. Leave this field empty (i.e. put `""`) to get the list of available GARs.

`--aggregator-args`
: Same as for `--experiment-args`, although none of the provided GAR support these additional arguments (i.e. if you provide some arguments here, they will be ignored by our GARs).

`--optimizer`
: Update rule to use. One of:
  - *adadelta* (`tf.train.AdadeltaOptimizer`)
  - *adagrad* (`tf.train.AdagradOptimizer`)
  - *adam* (`tf.train.AdamOptimizer`)
  - *rmsprop* (`tf.train.RMSPropOptimizer`)
  - *sgd* (`tf.train.GradientDescentOptimizer`)

`--optimizer-args`
: The list of dependant parameters is available at AggregaThor/graph.py:58

`--learning-rate`
: Learning rate evolution. One of:
  - *fixed* (`tf.constant`)
  - *polynomial* (`tf.train.polynomial_decay`)
  - *exponential* (`tf.train.exponential_decay`)

`--optimizer-args`
: The list of dependant parameters is available at: AggregaThor/graph.py:51

`--l1-regularize`
: L1 regularization parameter to use, none by default.

`--l2-regularize`
: L2 regularization parameter to use, none by default.

Please note that none of the options `--attack` or `--attack-args` is yet supported. The attack shown in the paper can be carried out by the *mnistAttack* experiment (experiments/mnistAttack.py).

Our implementation of AggregaThor is modular. Adding a new experiment only boils down to adding a Python module to AggregaThor/experiments/. The same applies for a new aggregation rule, in AggregaThor/aggregators/. We don't provide explicit documentation for these steps, but we believe experiments/mnist.py and aggregators/average.py are simple enough to act as templates.

A.8 Methodology

Submission, reviewing and badging methodology:

- `http://cTuning.org/ae/submission-20190109.html`
- `http://cTuning.org/ae/reviewing-20190109.html`
- `https://www.acm.org/publications/policies/artifact-review-badging`

Supplementary proofs: AGGREGATHOR Byzantine-resilience and Convergence Speed*

Abstract

In [1], Krum, the first provably Byzantine resilient algorithm for SGD, was introduced. Krum only uses one worker per step, which hampers its speed of convergence, especially in best case conditions when none of the workers is actually Byzantine. The idea behind MULTI-KRUM, of using $m > 1$ different workers per step was mentioned in [1], without however any proof neither on its Byzantine resilience nor on its slowdown. The present technical report closes this open problem and provides proofs of (weak) Byzantine resilience, convergence, and $\sqrt{\frac{m}{n}}$ slowdown of MULTI-KRUM compared to the optimal averaging in the absence of Byzantine workers. Based on that, and on the theoretical work of [4], we prove the similar $\sqrt{\frac{m}{n}}$ slowdown of AGGREGATHOR and its (strong) Byzantine resilience. We deduce that AGGREGATHOR ensures strong Byzantine resilience and the very fact that it is $\sqrt{\frac{m}{n}}$ times as fast as the optimal algorithm (averaging) in the absence of Byzantine workers.

AGGREGATHOR is the composition of MULTI-KRUM and BULYAN, which can be viewed as generalization (also using m different workers per step to leverage the fact that f, possibly less than a minority can be faulty) of *Bulyan*, the defense mechanism of [4]. Before presenting in Section 2, our proofs of convergence and slow down of MULTI-KRUM and in Section 3 our proofs of convergence and slow down of BULYAN and hence AGGREGATHOR, we introduce in Section 1 a toolbox of formal definitions: weak, strong, and (α, f)–Byzantine resilience. We also present a necessary context on non-convex optimization, as well as its interplay with the high dimensionality of machine learning together with the $\sqrt{d}$ leeway it provides to strong attackers.

1 Theoretical Context

Intuitively, weak Byzantine resilience requires a GAR to guarantee convergence despite the presence of f Byzantine workers. It can be formally stated as follows.

Definition 1 (Weak Byzantine resilience)**.** *We say that a GAR ensures weak f-Byzantine resilience if the sequence $\boldsymbol{x}^{(k)}$ (Equation 2 in the main paper) converges almost surely to some $\boldsymbol{x}^*$ where $\boldsymbol{\nabla} Q(\boldsymbol{x}^*) = 0$, despite the presence of f Byzantine workers.*

*This theoretical note is part of a more detailed analysis available in [3]

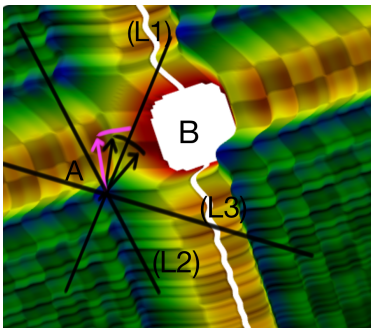


Figure 1: In a non-convex situation, two correct vectors (black arrows) are pointing towards the deep optimum located in area B, both vectors belong to the plane formed by lines L1 and L2. A Byzantine worker (magenta) is taking benefit from the third dimension, and the non-convex landscape, to place a vector that is heading towards one of the bad local optimums of area A. This Byzantine vector is located in the plane (L1,L3). Due to the variance of the correct workers on the plane (L1,L2), the Byzantine one has a budget of about $\sqrt{3}$ times the disagreement of the correct workers, to put as a deviation towards A, on the line (L3), while still being selected by a weak Byzantine resilient GAR, since its projection on the plane (L1,L2) lies exactly on the line (L1), unlike that of the correct workers. In very high dimensions, the situation is amplified by $\sqrt{d}$.

On the other hand, strong Byzantine resilience requires that this convergence does not lead to "bad" optimums, and is related to more intricate problem of non-convex optimization, which, in the presence of Byzantine workers, is highly aggravated by the dimension of the problem as explained in what follows.

Specificity of non-convex optimization. Non-convex optimization is one of the earliest established NP-hard problems [5]. In fact, many, if not all of the interesting but hard questions in machine learning boil down to one answer: "because the cost function is not convex".

In distributed machine learning, the non-convexity of the cost function creates two non-intuitive behaviours that are important to highlight.

(1) A "mild" Byzantine worker can make the system converge faster. For instance, it has been reported several times in the literature that noise accelerates learning [2, 5]. This can be understood from the "S" (stochasticity) of SGD: as (correct) workers cannot have a full picture of the surrounding landscape of the loss, they can only draw a sample at random and estimate the best direction based on that sample, which can be - and is probably - different the true gradient. But on expectation (over samples) this gradient estimate is equal to the true gradient. Moreover, due to non-convexity, even the true gradient might be leading to the local minima where the parameter server is. By providing a wrong direction (i.e., not the true gradient, or a correct stochastic estimation), a Byzantine worker might end up providing a direction to get out of that

local minima ! Unless of course when the computational resources of that Byzantine worker can face the high-dimensional landscape of the loss and find a truly misleading update vector.

(2) Combined with high dimensional issues, non-convexity explains the need for strong Byzantine resilience. Unlike the "mild" Byzantine worker, a strong adversary with more resources than the workers and the server, can see a larger picture and provide an attack that requires a stronger requirement. Namely, a requirement that would cut the $\sqrt{d}$ leeway offered to an attacker in each dimension. Figure 1 provides an illustration.

This motivates the following formalization of strong Byzantine resilience.

Definition 2 (Strong Byzantine resilience)**.** *We say that a GAR ensures strong f-Byzantine resilient if for every $i \in [1, d]$, there exists a correct gradient $\boldsymbol{G}$ (i.e., computed by a non-Byzantine worker) s.t. $\mathbb{E}|\boldsymbol{GAR}_i - \boldsymbol{G}_i| = O(\frac{1}{\sqrt{d}})$. The the expectation is taken over the random samples (ξ in Equation 4 of the main paper)and $\boldsymbol{v}_i$ denotes the i^{th} coordinate of a vector $\boldsymbol{v}$.*

For the sake of our theoretical analysis, we also introduce the definition of (α, f)–Byzantine resilience (Definition 3). This definition is a sufficient condition (as proved in [1] based on [2]) for weak Byzantine resilience that we introduce and require from GARs in our main paper (Section 2, Definition 1). Eventhough the property of (α, f)–Byzantine resilience is a sufficient, but not a necessary condition for (weak) Byzantine resilience, it has been so far used as the defacto standard [1, 7] to guarantee (weak) Byzantine resilience for SGD. We will therefore follow this standard and require (α, f)–Byzantine resilience from any GAR that is plugged into AGGREGATHOR, in particular, we will require it from MULTI-KRUM. The theoretical analysis done in [4] guarantees that BULYAN inherits it.

Intuitively, Definition 3 states that the gradient aggregation rule GAR produces an output vector that lives, on average (over random samples used by SGD), in the cone of angle α around the true gradient. We simply call this the "correct cone".

Definition 3 ((α, f)–Byzantine resilience)**.** *Let $0 \leq \alpha < \pi/2$ be any angular value, and any integer $0 \leq f \leq n$. Let $V_1, \ldots, V_n$ be any independent identically distributed random vectors in $\mathbb{R}^d$, $V_i \sim G$, with $\mathbb{E}G = g$. Let $B_1, \ldots, B_f$ be any random vectors in $\mathbb{R}^d$, possibly dependent on the V_i's. An aggregation rule GAR is said to be (α, f)-Byzantine resilient if, for any $1 \leq j_1 < \cdots < j_f \leq n$, vector*

$$GAR = GAR(V_1, \ldots, \underbrace{B_1}_{j_1}, \ldots, \underbrace{B_f}_{j_f}, \ldots, V_n)$$

satisfies (i) $\langle \mathbb{E}GAR, g \rangle \geq (1 - \sin\alpha) \cdot \|g\|^2 > 0$ [1] *and* (ii) *for $r = 2, 3, 4$, $\mathbb{E}\|GAR\|^r$ is bounded above by a linear combination of terms $\mathbb{E}\|G\|^{r_1} \ldots \mathbb{E}\|G\|^{r_{n-1}}$ with $r_1 + \cdots + r_{n-1} = r$.*

[1] Having a scalar product that is lower bounded by this value guarantees that the GAR of MULTI-KRUM lives in the aformentioned cone. For a visualisation of this requirement, see the ball and inner triangle of Figure 2

We first prove the (α, f)–Byzantine resilience of MULTI-KRUM (Lemma 1), then prove its almost sure convergence (Lemma 2) based on that, which proves the weak Byzantine resilience of MULTI-KRUM (Theorem 1).

In all what follows, expectations are taken over random samples used by correct workers to estimate the gradient, i.e the "S" (stochasticity) that is inherent to SGD. It is worth noting that this analysis in expectation is not an average case analysis from the point of view of Byzantine fault tolerance. For instance, the Byzantine worker is always assumed to follow arbitrarily bad policies and the analysis is a worst-case one.

The Byzantine resilience proof (Lemma 1) relies on the following observation: given $m \leq n - f - 2$, and in particular $m = n - f - 2$ [2], m-Krum averages m gradients that are all in the "correct cone", and a cone is a convex set, thus stable by averaging. The resulting vectors therefore also live in that cone. The angle of the cone will depend on a variable $\eta(n.f)$ as in [1], the value of $\eta(n.f)$ itself depends on m. This is what enables us to use multi-Krum as the basis of our MULTI-KRUM, unlike [1] where a restriction is made on $m = 1$.

The proof of Lemma 2 is the same as the one in [1] which itself draws on the rather classic analysis of SGD made by L.Bottou [2]. The key concepts are (1) a global confinement of the sequence of parameter vectors and (2) a bound on the statistical moments of the random sequence of estimators built by the GAR of MULTI-KRUM. As in [1,2], reasonable assumptions are made on the cost function Q, those assumption are not restrictive and are common in practical machine learning.

2 MULTI-KRUM: Weak Byzantine Resilience and Slowdown

Let n be any integer greater than 2, f any integer s.t $f \leq \frac{n-2}{2}$ and m an integer s.t $m \leq n - f - 2$. Let $\tilde{m} = n - f - 2$.

Theorem 1 (Byzantine resilience and slowdown of MULTI-KRUM)**.** *Let m be any integer s.t. $m \leq n - f - 2$.* ***(i)*** MULTI-KRUM *has weak Byzantine resilience against f failures.* ***(ii)*** *In the absence of Byzantine workers,* MULTI-KRUM *has a slowdown (expressed in ratio with averaging) of $\Omega(\sqrt{\frac{\tilde{m}}{n}})$.*

Proof. **Proof of (i).** To prove **(i)**, we will require Lemma 1 and Lemma 2, then conclude by construction of MULTI-KRUM as a multi-Krum algorithm with $m = n-f-2$.

Lemma 1. *Let $V_1, \ldots, V_n$ be any independent and identically distributed random d-dimensional vectors s.t $V_i \sim G$, with $\mathbb{E}G = g$ and $\mathbb{E}\left\|G - g\right\|^2 = d\sigma^2$. Let $B_1, \ldots, B_f$ be any f random vectors, possibly dependent on the V_i's. If $2f+2 < n$ and $\eta(n, f)\sqrt{d} \cdot \sigma < \|g\|$, where*

$$\eta(n, f) \underset{def}{=} \sqrt{2\left(n - f + \frac{f \cdot m + f^2 \cdot (m+1)}{m}\right)},$$

[2] The slowdown question is an incentive to take the highest value of m among those that satisfy Byzantine resilience, in this case $\tilde{m}$.

then the GAR function of MULTI-KRUM *is (α, f)-Byzantine resilient where $0 \leq \alpha < \pi/2$ is defined by*

$$\sin \alpha = \frac{\eta(n,f) \cdot \sqrt{d} \cdot \sigma}{\|g\|}.$$

Proof. Without loss of generality, we assume that the Byzantine vectors $B_1, \ldots, B_f$ occupy the last f positions in the list of arguments of MULTI-KRUM, i.e., MULTI-KRUM $=$ MULTI-KRUM$(V_1, \ldots, V_{n-f}, B_1, \ldots, B_f)$. An index is *correct* if it refers to a vector among $V_1, \ldots, V_{n-f}$. An index is *Byzantine* if it refers to a vector among $B_1, \ldots, B_f$. For each index (correct or Byzantine) i, we denote by $\delta_c(i)$ (resp. $\delta_b(i)$) the number of correct (resp. Byzantine) indices j such that $i \to j$ (the notation we introduced in Section 3 when defining MULTI-KRUM), i.e the number of workers, among the m neighbors of i that are correct (resp. Byzantine). We have

$$\begin{aligned} \delta_c(i) + \delta_b(i) &= m \\ n - 2f - 2 \leq \delta_c(i) &\leq m \\ \delta_b(i) &\leq f. \end{aligned}$$

We focus first on the condition *(i)* of (α, f)-Byzantine resilience. We determine an upper bound on the squared distance $\|\mathbb{E}\text{MULTI-KRUM} - g\|^2$. Note that, for any correct j, $\mathbb{E}V_j = g$. We denote by i_* the index of the worst scoring among the m vectors chosen by the MULTI-KRUM function, i.e one that ranks with the m^{th} smallest score in Equation 5 of the main paper (Section 3).

$$\begin{aligned} \|\mathbb{E}\text{MULTI-KRUM} - g\|^2 &\leq \left\| \mathbb{E}\left(\text{MULTI-KRUM} - \frac{1}{\delta_c(i_*)} \sum_{i_* \to \text{ correct } j} V_j \right) \right\|^2 \\ &\leq \mathbb{E} \left\| \text{MULTI-KRUM} - \frac{1}{\delta_c(i_*)} \sum_{i_* \to \text{ correct } j} V_j \right\|^2 \quad \text{(Jensen inequality)} \\ &\leq \sum_{\text{correct } i} \mathbb{E} \left\| V_i - \frac{1}{\delta_c(i)} \sum_{i \to \text{ correct } j} V_j \right\|^2 \mathbb{I}(i_* = i) \\ &\quad + \sum_{\text{byz } k} \mathbb{E} \left\| B_k - \frac{1}{\delta_c(k)} \sum_{k \to \text{ correct } j} V_j \right\|^2 \mathbb{I}(i_* = k) \end{aligned}$$

where $\mathbb{I}$ denotes the indicator function[3]. We examine the case $i_* = i$ for some correct index i.

$$\left\| V_i - \frac{1}{\delta_c(i)} \sum_{i \to \text{ correct } j} V_j \right\|^2 = \left\| \frac{1}{\delta_c(i)} \sum_{i \to \text{ correct } j} V_i - V_j \right\|^2$$

[3] $\mathbb{I}(P)$ equals 1 if the predicate P is true, and 0 otherwise.

$$\leq \frac{1}{\delta_c(i)} \sum_{i\to \text{ correct } j} \|V_i - V_j\|^2 \quad \text{(Jensen inequality)}$$

$$\mathbb{E}\left\|V_i - \frac{1}{\delta_c(i)} \sum_{i\to \text{ correct } j} V_j\right\|^2 \leq \frac{1}{\delta_c(i)} \sum_{i\to \text{ correct } j} \mathbb{E}\,\|V_i - V_j\|^2$$
$$\leq 2d\sigma^2.$$

We now examine the case $i_* = k$ for some Byzantine index k. The fact that k minimizes the score implies that for all correct indices i

$$\sum_{k\to \text{ correct } j} \|B_k - V_j\|^2 + \sum_{k\to \text{ byz } l} \|B_k - B_l\|^2 \leq \sum_{i\to \text{ correct } j} \|V_i - V_j\|^2 + \sum_{i\to \text{ byz } l} \|V_i - B_l\|^2 .$$

Then, for all correct indices i

$$\left\|B_k - \frac{1}{\delta_c(k)} \sum_{k\to \text{ correct } j} V_j\right\|^2 \leq \frac{1}{\delta_c(k)} \sum_{k\to \text{ correct } j} \|B_k - V_j\|^2$$
$$\leq \frac{1}{\delta_c(k)} \sum_{i\to \text{ correct } j} \|V_i - V_j\|^2 + \frac{1}{\delta_c(k)} \underbrace{\sum_{i\to \text{ byz } l} \|V_i - B_l\|^2}_{D^2(i)} .$$

We focus on the term $D^2(i)$. Each correct process i has m neighbors, and $f+1$ non-neighbors. Thus there exists a correct worker $\zeta(i)$ which is farther from i than any of the neighbors of i. In particular, for each Byzantine index l such that $i \to l$, $\|V_i - B_l\|^2 \leq \left\|V_i - V_{\zeta(i)}\right\|^2$. Whence

$$\left\|B_k - \frac{1}{\delta_c(k)} \sum_{k\to \text{ correct } j} V_j\right\|^2 \leq \frac{1}{\delta_c(k)} \sum_{i\to \text{ correct } j} \|V_i - V_j\|^2 + \frac{\delta_b(i)}{\delta_c(k)} \left\|V_i - V_{\zeta(i)}\right\|^2$$

$$\mathbb{E}\left\|B_k - \frac{1}{\delta_c(k)} \sum_{k\to \text{ correct } j} V_j\right\|^2 \leq \frac{\delta_c(i)}{\delta_c(k)} \cdot 2d\sigma^2 + \frac{\delta_b(i)}{\delta_c(k)} \sum_{\text{correct } j\neq i} \mathbb{E}\,\|V_i - V_j\|^2 \mathbb{I}(\zeta(i) = j)$$
$$\leq \left(\frac{\delta_c(i)}{\delta_c(k)} \cdot + \frac{\delta_b(i)}{\delta_c(k)}(m+1)\right) 2d\sigma^2$$
$$\leq \left(\frac{m}{n-2f-2} + \frac{f}{n-2f-2} \cdot (m+1)\right) 2d\sigma^2.$$

Putting everything back together, we obtain

$$\|\mathbb{E}\textsc{Multi-Krum} - g\|^2 \leq (n-f)2d\sigma^2 + f \cdot \left(\frac{m}{n-2f-2} + \frac{f}{n-2f-2} \cdot (m+1)\right) 2d\sigma^2$$
$$\leq \underbrace{2\left(n - f + \frac{f\cdot m + f^2 \cdot (m+1)}{n-2f-2}\right)}_{\eta^2(n,f)} d\sigma^2.$$

By assumption, $\eta(n,f)\sqrt{d}\sigma < \|g\|$, i.e., $\mathbb{E}$MULTI-KRUM belongs to a ball centered at g with radius $\eta(n,f)\cdot\sqrt{d}\cdot\sigma$. This implies

$$\langle \mathbb{E}\text{MULTI-KRUM}, g\rangle \geq \Big(\|g\| - \eta(n,f)\cdot\sqrt{d}\cdot\sigma\Big)\cdot\|g\| = (1-\sin\alpha)\cdot\|g\|^2.$$

To sum up, condition *(i)* of the (α, f)-Byzantine resilience property holds. We now focus on condition *(ii)*.

$$\begin{aligned}\mathbb{E}\|\text{MULTI-KRUM}\|^r &= \sum_{\text{correct } i}\mathbb{E}\,\|V_i\|^r\,\mathbb{I}(i_* = i) + \sum_{\text{byz } k}\mathbb{E}\,\|B_k\|^r\,\mathbb{I}(i_* = k)\\ &\leq (n-f)\mathbb{E}\,\|G\|^r + \sum_{\text{byz } k}\mathbb{E}\,\|B_k\|^r\,\mathbb{I}(i_* = k).\end{aligned}$$

Denoting by C a generic constant, when $i_* = k$, we have for all correct indices i

$$\begin{aligned}\left\|B_k - \frac{1}{\delta_c(k)}\sum_{k\to\text{correct } j} V_j\right\| &\leq \sqrt{\frac{1}{\delta_c(k)}\sum_{i\to\text{ correct } j}\|V_i - V_j\|^2 + \frac{\delta_b(i)}{\delta_c(k)}\left\|V_i - V_{\zeta(i)}\right\|^2}\\ &\leq C\cdot\left(\sqrt{\frac{1}{\delta_c(k)}}\cdot\sum_{i\to\text{correct } j}\|V_i - V_j\| + \sqrt{\frac{\delta_b(i)}{\delta_c(k)}}\cdot\left\|V_i - V_{\zeta(i)}\right\|\right)\\ &\leq C\cdot\sum_{\text{correct } j}\|V_j\| \quad \text{(triangular inequality)}.\end{aligned}$$

The second inequality comes from the equivalence of norms in finite dimension. Now

$$\begin{aligned}\|B_k\| &\leq \left\|B_k - \frac{1}{\delta_c(k)}\sum_{k\to\text{correct } j} V_j\right\| + \left\|\frac{1}{\delta_c(k)}\sum_{k\to\text{correct } j} V_j\right\|\\ &\leq C\cdot\sum_{\text{correct } j}\|V_j\|\\ \|B_k\|^r &\leq C\cdot\sum_{r_1+\cdots+r_{n-f}=r}\|V_1\|^{r_1}\cdots\|V_{n-f}\|^{r_{n-f}}.\end{aligned}$$

Since the V_i's are independent, we finally obtain that $\mathbb{E}\,\|\text{MULTI-KRUM}\|^r$ is bounded above by a linear combination of terms of the form $\mathbb{E}\,\|V_1\|^{r_1}\cdots\mathbb{E}\,\|V_{n-f}\|^{r_{n-f}} = \mathbb{E}\,\|G\|^{r_1}\cdots\mathbb{E}\,\|G\|^{r_{n-f}}$ with $r_1+\cdots+r_{n-f}=r$. This completes the proof of condition *(ii)*. □

Lemma 2. *Assume that* (i) *the cost function Q is three times differentiable with continuous derivatives, and is non-negative, $Q(x)\geq 0$;* (ii) *the learning rates satisfy $\sum_t\gamma_t = \infty$ and $\sum_t\gamma_t^2 < \infty$;* (iii) *the gradient estimator satisfies $\mathbb{E}G(x,\xi) = \nabla Q(x)$*

and $\forall r \in \{2, \dots, 4\},\ \mathbb{E}\|G(x,\xi)\|^r \leq A_r + B_r\|x\|^r$ *for some constants* A_r, B_r*;* (iv) *there exists a constant* $0 \leq \alpha < \pi/2$ *such that for all* x

$$\eta(n,f) \cdot \sqrt{d} \cdot \sigma(x) \leq \|\nabla Q(x)\| \cdot \sin\alpha;$$

(v) *finally, beyond a certain horizon,* $\|x\|^2 \geq D$*, there exist* $\epsilon > 0$ *and* $0 \leq \beta < \pi/2 - \alpha$ *such that*

$$\begin{aligned} \|\nabla Q(x)\| &\geq \epsilon > 0 \\ \frac{\langle x, \nabla Q(x)\rangle}{\|x\| \cdot \|\nabla Q(x)\|} &\geq \cos\beta. \end{aligned}$$

Then the sequence of gradients $\nabla Q(x_t)$ *converges almost surely to zero.*

Proof. For the sake of simplicity, we write $\textsc{Multi-Krum}_t = \textsc{Multi-Krum}(V_1^t, \dots, V_n^t)$. Before proving the main claim of the proposition, we first show that the sequence x_t is almost surely globally confined within the region $\|x\|^2 \leq D$.

(Global confinement). Let $u_t = \phi(\|x_t\|^2)$ where

$$\phi(a) = \begin{cases} 0 & \text{if } a < D \\ (a-D)^2 & \text{otherwise} \end{cases}$$

Note that

$$\phi(b) - \phi(a) \leq (b-a)\phi'(a) + (b-a)^2. \tag{1}$$

This becomes an equality when $a, b \geq D$. Applying this inequality to $u_{t+1} - u_t$ yields

$$\begin{aligned} u_{t+1} - u_t \leq\ & \left(-2\gamma_t\langle x_t, \textsc{Multi-Krum}_t\rangle + \gamma_t^2\|\textsc{Multi-Krum}_t\|^2\right) \cdot \phi'(\|x_t\|^2) \\ & + 4\gamma_t^2\langle x_t, \textsc{Multi-Krum}_t\rangle^2 - 4\gamma_t^3\langle x_t, \textsc{Multi-Krum}_t\rangle\|\textsc{Multi-Krum}_t\|^2 \\ & + \gamma_t^4\|\textsc{Multi-Krum}_t\|^4 \\ & \leq -2\gamma_t\langle x_t, \textsc{Multi-Krum}_t\rangle\phi'(\|x_t\|^2) + \gamma_t^2\|\textsc{Multi-Krum}_t\|^2\phi'(\|x_t\|^2) \\ & + 4\gamma_t^2\|x_t\|^2\|\textsc{Multi-Krum}_t\|^2 + 4\gamma_t^3\|x_t\|\|\textsc{Multi-Krum}_t\|^3 \\ & + \gamma_t^4\|\textsc{Multi-Krum}_t\|^4. \end{aligned}$$

Let $\mathcal{P}_t$ denote the σ-algebra encoding all the information up to round t. Taking the conditional expectation with respect to $\mathcal{P}_t$ yields

$$\begin{aligned} \mathbb{E}\left(u_{t+1} - u_t | \mathcal{P}_t\right) \leq\ & -2\gamma_t\langle x_t, \mathbb{E}\textsc{Multi-Krum}_t\rangle + \gamma_t^2\mathbb{E}\left(\|\textsc{Multi-Krum}_t\|^2\right)\phi'(\|x_t\|^2) \\ & + 4\gamma_t^2\|x_t\|^2\mathbb{E}\left(\|\textsc{Multi-Krum}_t\|^2\right) + 4\gamma_t^3\|x_t\|\mathbb{E}\left(\|\textsc{Multi-Krum}_t\|^3\right) \\ & + \gamma_t^4\mathbb{E}\left(\|\textsc{Multi-Krum}_t\|^4\right). \end{aligned}$$

Thanks to condition *(ii)* of (α, f)-Byzantine resilience, and the assumption on the first four moments of G, there exist positive constants A_0, B_0 such that

$$\mathbb{E}\left(u_{t+1} - u_t | \mathcal{P}_t\right) \leq -2\gamma_t\langle x_t, \mathbb{E}\textsc{Multi-Krum}_t\rangle\phi'(\|x_t\|^2) + \gamma_t^2\left(A_0 + B_0\|x_t\|^4\right).$$

Thus, there exist positive constant A, B such that

$$\mathbb{E}\left(u_{t+1} - u_t | \mathcal{P}_t\right) \leq -2\gamma_t \langle x_t, \mathbb{E}\text{MULTI-KRUM}_t \rangle \phi'(\|x_t\|^2) + \gamma_t^2 \left(A + B \cdot u_t\right).$$

When $\|x_t\|^2 < D$, the first term of the right hand side is null because $\phi'(\|x_t\|^2) = 0$. When $\|x_t\|^2 \geq D$, this first term is negative because (see Figure 2)

$$\langle x_t, \mathbb{E}\text{MULTI-KRUM}_t \rangle \geq \|x_t\| \cdot \|\mathbb{E}\text{MULTI-KRUM}_t\| \cdot \cos(\alpha + \beta) > 0.$$

Hence

$$\mathbb{E}\left(u_{t+1} - u_t | \mathcal{P}_t\right) \leq \gamma_t^2 \left(A + B \cdot u_t\right).$$

We define two auxiliary sequences

$$\mu_t = \prod_{i=1}^{t} \frac{1}{1 - \gamma_i^2 B} \xrightarrow[t \to \infty]{} \mu_\infty$$
$$u'_t = \mu_t u_t.$$

Note that the sequence μ_t converges because $\sum_t \gamma_t^2 < \infty$. Then

$$\mathbb{E}\left(u'_{t+1} - u'_t | \mathcal{P}_t\right) \leq \gamma_t^2 \mu_t A.$$

Consider the indicator of the positive variations of the left-hand side

$$\chi_t = \begin{cases} 1 & \text{if } \mathbb{E}\left(u'_{t+1} - u'_t | \mathcal{P}_t\right) > 0 \\ 0 & \text{otherwise} \end{cases}$$

Then

$$\mathbb{E}\left(\chi_t \cdot (u'_{t+1} - u'_t)\right) \leq \mathbb{E}\left(\chi_t \cdot \mathbb{E}\left(u'_{t+1} - u'_t | \mathcal{P}_t\right)\right) \leq \gamma_t^2 \mu_t A.$$

The right-hand side of the previous inequality is the summand of a convergent series. By the quasi-martingale convergence theorem [6], this shows that the sequence u'_t converges almost surely, which in turn shows that the sequence u_t converges almost surely, $u_t \to u_\infty \geq 0$.

Let us assume that $u_\infty > 0$. When t is large enough, this implies that $\|x_t\|^2$ and $\|x_{t+1}\|^2$ are greater than D. Inequality 1 becomes an equality, which implies that the following infinite sum converges almost surely

$$\sum_{t=1}^{\infty} \gamma_t \langle x_t, \mathbb{E}\text{MULTI-KRUM}_t \rangle \phi'(\|x_t\|^2) < \infty.$$

Note that the sequence $\phi'(\|x_t\|^2)$ converges to a positive value. In the region $\|x_t\|^2 > D$, we have

$$\begin{aligned}
\langle x_t, \mathbb{E}\text{MULTI-KRUM}_t \rangle &\geq \sqrt{D} \cdot \|\mathbb{E}\text{MULTI-KRUM}_t\| \cdot \cos(\alpha + \beta) \\
&\geq \sqrt{D} \cdot \left(\|\nabla Q(x_t)\| - \eta(n, f) \cdot \sqrt{d} \cdot \sigma(x_t)\right) \cdot \cos(\alpha + \beta) \\
&\geq \sqrt{D} \cdot \epsilon \cdot (1 - \sin\alpha) \cdot \cos(\alpha + \beta) > 0.
\end{aligned}$$

This contradicts the fact that $\sum_{t=1}^{\infty} \gamma_t = \infty$. Therefore, the sequence u_t converges to zero. This convergence implies that the sequence $\|x_t\|^2$ is bounded, i.e., the vector x_t is confined in a bounded region containing the origin. As a consequence, any continuous function of x_t is also bounded, such as, e.g., $\|x_t\|^2$, $\mathbb{E}\left\|G(x_t, \xi)\right\|^2$ and all the derivatives of the cost function $Q(x_t)$. In the sequel, positive constants K_1, K_2, etc... are introduced whenever such a bound is used.

(Convergence). We proceed to show that the gradient $\nabla Q(x_t)$ converges almost surely to zero. We define

$$h_t = Q(x_t).$$

Using a first-order Taylor expansion and bounding the second derivative with K_1, we obtain

$$|h_{t+1} - h_t + 2\gamma_t \langle \textsc{Multi-Krum}_t, \nabla Q(x_t) \rangle| \leq \gamma_t^2 \|\textsc{Multi-Krum}_t\|^2 K_1 \text{ a.s.}$$

Therefore

$$\mathbb{E}\left(h_{t+1} - h_t | \mathcal{P}_t\right) \leq -2\gamma_t \langle \mathbb{E}\textsc{Multi-Krum}_t, \nabla Q(x_t) \rangle + \gamma_t^2 \mathbb{E}\left(\|\textsc{Multi-Krum}_t\|^2 | \mathcal{P}_t\right) K_1. \tag{2}$$

By the properties of (α, f)-Byzantine resiliency, this implies

$$\mathbb{E}\left(h_{t+1} - h_t | \mathcal{P}_t\right) \leq \gamma_t^2 K_2 K_1,$$

which in turn implies that the positive variations of h_t are also bounded

$$\mathbb{E}\left(\chi_t \cdot (h_{t+1} - h_t)\right) \leq \gamma_t^2 K_2 K_1.$$

The right-hand side is the summand of a convergent infinite sum. By the quasi-martingale convergence theorem, the sequence h_t converges almost surely, $Q(x_t) \to Q_\infty$.

Taking the expectation of Inequality 2, and summing on $t = 1, \ldots, \infty$, the convergence of $Q(x_t)$ implies that

$$\sum_{t=1}^{\infty} \gamma_t \langle \mathbb{E}\textsc{Multi-Krum}_t, \nabla Q(x_t) \rangle < \infty \text{ a.s.}$$

We now define

$$\rho_t = \left\|\nabla Q(x_t)\right\|^2.$$

Using a Taylor expansion, as demonstrated for the variations of h_t, we obtain

$$\rho_{t+1} - \rho_t \leq -2\gamma_t \langle \textsc{Multi-Krum}_t, \left(\nabla^2 Q(x_t)\right) \cdot \nabla Q(x_t) \rangle + \gamma_t^2 \left\|\textsc{Multi-Krum}_t\right\|^2 K_3 \text{ a.s.}$$

Taking the conditional expectation, and bounding the second derivatives by K_4,

$$\mathbb{E}\left(\rho_{t+1} - \rho_t | \mathcal{P}_t\right) \leq 2\gamma_t \langle \mathbb{E}\textsc{Multi-Krum}_t, \nabla Q(x_t) \rangle K_4 + \gamma_t^2 K_2 K_3.$$

The positive expected variations of ρ_t are bounded

$$\mathbb{E}\left(\chi_t \cdot (\rho_{t+1} - \rho_t)\right) \leq 2\gamma_t \mathbb{E}\langle \mathbb{E}\textsc{Multi-Krum}_t, \nabla Q(x_t) \rangle K_4 + \gamma_t^2 K_2 K_3.$$

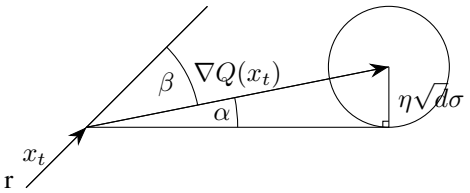


Figure 2: Condition on the angles between x_t, $\nabla Q(x_t)$ and the the GAR of MULTI-KRUM vector $\mathbb{E}$MULTI-KRUM$_t$, in the region $\|x_t\|^2 > D$.

The two terms on the right-hand side are the summands of convergent infinite series. By the quasi-martingale convergence theorem, this shows that ρ_t converges almost surely.

We have

$$\begin{aligned}\langle \mathbb{E}\text{MULTI-KRUM}_t, \nabla Q(x_t)\rangle &\geq \Big(\|\nabla Q(x_t)\| - \eta(n,f)\cdot\sqrt{d}\cdot\sigma(x_t)\Big)\cdot\|\nabla Q(x_t)\| \\ &\geq \underbrace{(1-\sin\alpha)}_{>0}\cdot\rho_t.\end{aligned}$$

This implies that the following infinite series converge almost surely

$$\sum_{t=1}^{\infty}\gamma_t\cdot\rho_t < \infty.$$

Since ρ_t converges almost surely, and the series $\sum_{t=1}^{\infty}\gamma_t = \infty$ diverges, we conclude that the sequence $\|\nabla Q(x_t)\|$ converges almost surely to zero. □

We conclude the proof of **(i)** by recalling the definition of MULTI-KRUM, as the instance of $m - Krum$ with $m = n - f - 2$. □

3 AGGREGATHOR: Strong Byzantine Resilience and Slowdown

Let n be any integer greater than 2, f any integer s.t $f \leq \frac{n-3}{4}$ and m an integer s.t $m \leq n - 2f - 2$. Let $\tilde{m} = n - 2f - 2$.

Theorem 2 (Byzantine resilience and slowdown of AGGREGATHOR)**.**
(i) AGGREGATHOR *provides strong Byzantine resilience against f failures.*
(ii) In the absence of Byzantine workers, AGGREGATHOR *has a slowdown (expressed in ratio with averaging) of* $\Omega(\sqrt{\frac{\tilde{m}}{n}})$.

Proof. **(i).** If the number of iterations over MULTI-KRUM is $n - 2f$, then the leeway, defined by the coordinate-wise distance between the output of BULYAN and a correct

gradient is upper bounded by $\mathcal{O}(\frac{1}{\sqrt{d}})$. This is due to the fact that BULYAN relies on a component-wise median, that, as proven in [4] guarantees this bound. The proof is then a direct consequence of Theorem 1 and the properties of Bulyan [4]

(ii) is a consequence of the fact that m-Krum is the average of m estimators of the gradient. In the absence of Byzantine workers, all those estimators will not only be from the "correct cone", but from correct workers (Byzantine workers can also be in the correct cone, but in this case there are none). As SGD converges in $O(\frac{1}{\sqrt{m}})$, where m is the number of used estimators of the gradient, the slowdown result follows. □

References

[1] BLANCHARD, P., EL MHAMDI, E. M., GUERRAOUI, R., AND STAINER, J. Machine learning with adversaries: Byzantine tolerant gradient descent. In *Neural Information Processing Systems* (2017), pp. 118–128.

[2] BOTTOU, L. Online learning and stochastic approximations. *Online learning in neural networks 17*, 9 (1998), 142.

[3] EL-MHAMDI, E.-M., AND GUERRAOUI, R. Fast and secure distributed learning in high dimension. *arXiv preprint arXiv:* (2019).

[4] EL MHAMDI, E. M., GUERRAOUI, R., AND ROUAULT, S. The hidden vulnerability of distributed learning in Byzantium. In *Proceedings of the 35th International Conference on Machine Learning* (Stockholmsmässan, Stockholm Sweden, 10–15 Jul 2018), J. Dy and A. Krause, Eds., vol. 80 of *Proceedings of Machine Learning Research*, PMLR, pp. 3521–3530.

[5] HAYKIN, S. S. *Neural networks and learning machines*, vol. 3. Pearson Upper Saddle River, NJ, USA:, 2009.

[6] MÉTIVIER, M. *Semi-Martingales*. Walter de Gruyter, 1983.

[7] XIE, C., KOYEJO, O., AND GUPTA, I. Generalized Byzantine-tolerant sgd. *arXiv preprint arXiv:1802.10116* (2018).