Federated Learning on Personal Data Management Systems: Decentralized and Reliable Secure Aggregation Protocols

Julien Mirval, Luc Bouganim, Iulian Sandu Popa

Federated Learning on Personal Data Management Systems: Decentralized and Reliable Secure Aggregation Protocols

Julien Mirval
julien.mirval@cozycloud.cc
Cozy Cloud, Inria-Saclay
UVSQ, Université Paris-Saclay
France

Luc Bouganim
luc.bouganim@inria.fr
Inria-Saclay
UVSQ, Université Paris-Saclay
France

Iulian Sandu-Popa
iulian.sandu-popa@uvsq.fr
UVSQ, Université Paris-Saclay
Inria-Saclay
France

ABSTRACT

The development and adoption of personal data management systems (PDMS) has been fueled by legal and technical means such as smart disclosure, data portability and data altruism. By using a PDMS, individuals can effortlessly gather and share data, generated directly by their devices or as a result of their interactions with companies or institutions. In this context, federated learning appears to be a very promising technology, but it requires secure, reliable, and scalable aggregation protocols to preserve user privacy and account for potential PDMS dropouts. Despite recent significant progress in secure aggregation for federated learning, we still lack a solution suitable for the fully decentralized PDMS context. This paper proposes a family of fully decentralized protocols that are scalable and reliable with respect to dropouts. We focus in particular on the reliability property which is key in a peer-to-peer system wherein aggregators are system nodes and are subject to dropouts in the same way as contributor nodes. We show that in a decentralized setting, reliability raises a tension between the potential completeness of the result and the aggregation cost. We then propose a set of strategies that deal with dropouts and offer different trade-offs between completeness and cost. We extensively evaluate the proposed protocols and show that they cover the design space allowing to favor completeness or cost in all settings.

CCS CONCEPTS

• **Computer systems organization → Peer-to-peer architectures**.

KEYWORDS

Secure aggregation, peer-to-peer, reliability, federated learning.

ACM Reference Format:
Julien Mirval, Luc Bouganim, and Iulian Sandu-Popa. 2023. Federated Learning on Personal Data Management Systems: Decentralized and Reliable Secure Aggregation Protocols. In *35th International Conference on Scientific and Statistical Database Management (SSDBM 2023), July 10–12, 2023, Los Angeles, CA, USA.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3603719.3603730

SSDBM 2023, July 10–12, 2023, Los Angeles, CA, USA

ACM ISBN 979-8-4007-0746-9/23/07…$15.00
https://doi.org/10.1145/3603719.3603730

1 INTRODUCTION

New privacy-protection regulations (e.g., GDPR) and smart disclosure initiatives in the last decade have boosted the development and adoption of Personal Data Management Systems (PDMSs) [3]. A PDMS (e.g., Cozy Cloud [13], Nextcloud, Solid) is a data platform that allows users to easily collect, store, and manage into a single place data directly generated by the user's devices (e.g., quantified-self data, smart home data, photos) and data resulting from the user's interactions (e.g., social interaction data, health, bank, telecom). Users can then leverage the power of their PDMS to benefit from their personal data for their own good and for the benefit of the community [10].

As a result, the PDMS paradigm leads to a shift in the personal data ecosystem since data becomes massively distributed, on the user side. It also holds the promise of unlocking innovative usages. An individual can now cross her data from different data silos, e.g., health records and physical activity data. In addition, individuals can leverage their PDMSs by forming large communities of users sharing their data. This allows, for example, to compute statistics for epidemiological studies or to train a Machine Learning (ML) model for recommendation systems. In this context, it is natural to rely on a fully decentralized PDMS architecture (as opposed to central servers that raise several important issues such as cost, availability and scalability with the number of users), but this also poses new challenges.

Aggregation primitives are essential to compute basic statistics on user data and are also a fundamental building block for ML algorithms. In particular, Secure Aggregation (SA) is a central component of Federated Learning (FL), introduced in [21], as evidenced by the large body of recent work in this area [20]. However, to enable such new usages in the PDMS context, we need new solutions adapted to its specificity. First, PDMS users rely on large peer-to-peer systems for data sharing and computations [3, 9] thus requiring fully decentralized and scalable aggregation protocols, discarding data centralization on servers. Also, these protocols need to protect user privacy and adapt to varying selectivity (i.e., the consent of relevant participants). Ideally, the proposed protocol should provide an accurate result that takes advantage of the high-quality data available in PDMSs. Efficiency (i.e., protocol latency and total load of the system) is of prime importance given the potentially limited communication speed or computation power of PDMSs. Finally, given the scale of such decentralized aggregation, protocols must also be robust to node dropouts. To summarize, our goal is to design protocols that fulfill the following properties: **fully decentralized and highly scalable**, with the number of participants; **privacy-preserving**, i.e., protecting the confidentiality of

the contributed user data; **accurate**, i.e., no trade-off between accuracy and privacy (e.g., like in the data anonymization or differential privacy approaches); **adaptable**, i.e., adapting to a large spectrum of computation selectivity values (reflecting the subset of contributor nodes) and system configurations (network and cryptographic latency); and **reliable**, i.e., handling node dropouts (e.g., failures, voluntary disconnections or unexpected communication delays).

Ensuring these properties altogether is challenging and to the best of our knowledge, the existing distributed Secure Aggregation (SA) protocols fail to achieve this objective. On the one hand, approaches such as local differential privacy are based on adding noise to protect privacy. This affects accuracy [7] or reliability to dropouts [25] and requires a very large number of participants to reduce the impact of noise which contradicts an adaptive node selectivity (see Section 2). On the other hand, despite leveraging different cryptographic schemes in SA for FL [20] (e.g., encryption-based [6, 14] or secret sharing-based [8, 12, 18]), existing solutions employ a similar hybrid architecture wherein one or several highly available and powerful servers aggregate the data supplied by many user devices. Although some solutions consider the case of node dropouts, this applies to client devices and never to aggregation servers [8, 12]. In a Peer-to-Peer (P2P) PDMS system, all computations are performed by internal PDMS nodes (i.e., user devices). Hence, the data aggregators and data contributor nodes have the same constraints, i.e., limited computing power and availability. Such nodes cannot be expected to carry out heavy cryptographic operations [8] and can drop out during the computation. Fortunately, the P2P approach allows involving many nodes to perform a computation thus reducing the load on individual aggregators.

A first effort towards SA adapted to P2P systems was made in [22], where we designed a protocol that fulfill the above properties in an ideal setting, i.e., without considering the reliability issue. This work brings two major novelties. First, we focus on the reliability property, which is difficult to guarantee in a fully-decentralized setting and deserves a detailed study. Second, although our protocols apply to SA in general, we chose to study the more general case of FL, given its particular interest in the PDMS paradigm. The study of FL is also more challenging due to the potentially large size of the model, which increases the scalability problem. In our experiments, we consider model sizes from very small to very large, thus covering a wide range of use cases (including classical SA).

Our contributions are as follows. We analyze the impact of dropouts, be it contributor or aggregator nodes, on the other properties of an SA protocol designed for a P2P PDMS system. Node dropouts have a direct impact on accuracy (i.e., a single failure can make the final computation result useless) and on efficiency (i.e., it can introduce large latency). From this analysis, we derive the precise requirements of a reliable protocol and show that in a fully-decentralized context, reliability also introduces a tension between result completeness (i.e., the percentage of initial contribution in the final result, despite dropouts) and computation cost. We introduce the necessary building blocks to deal with these requirements. Then, we propose a variety of execution strategies offering different trade-offs between completeness and cost and allowing to cover a wide spectrum of dropout rates, contributor selectivity or trained model sizes. Our extensive experimental evaluation shows that the proposed strategies cover well the design space allowing to favor completeness or cost in all settings.

The rest of this paper is organized as follows. We discuss the related works w.r.t. the required properties in Section 2. We introduce, in Section 3, the considered architecture and threat model. Section 4 reminds the main design principles proposed in [22] and then introduces, as a starting point, a straw-man SA protocol which efficiently computes the required aggregation assuming an ideal world (i.e., there are no node dropouts). This allows to highlight the challenges induced by reliability issues. Section 5 presents the necessary building blocks to addresses the reliability related challenges. In Section 6, we propose four SA strategies that leverage those building blocks and allow for different trade-off between result completeness and aggregation cost. We extensively evaluate the proposed strategies in Section 7 and conclude in Section 8.

2 RELATED WORKS

Secure aggregation (SA) is an intense research area since many years leading to several approaches: SA based on cryptography, SA based on (local) differential privacy and gossip-based protocols and SA based on Trusted Execution Environments (TEE). However, these solutions are not adapted to the decentralized context and fail to cover all the required properties listed above.

SA based on cryptography. Cryptographic solutions for SA have been proposed since nearly three decades. The initial solutions were designed for wireless sensor networks [27], but the field has recently taken off again to meet the needs of FL. A recent survey [20] discusses about forty works for FL, grouped in four classes: SA using masking, SA using additively homomorphic encryption, SA using functional encryption and SA based on MPC (or secret shares). The proposed scheme may differ on the offered trade-off between the computation and the communication cost, or the tolerance to client dropouts. Regardless of the employed cryptographic scheme, all the existing solutions (e.g., [5, 8], Prio [12] and Drynx [16]) rely on a similar architecture wherein one or a handful of powerful and highly reliable servers collect encrypted user data and then apply costly aggregation algorithms based on masking (e.g., [30]), garbled circuits (e.g., [6, 14]) or secret sharing (e.g., [15, 18]).

SA based on cryptography with dropout support. Typical FL scenarios can involve a large number of (mobile) clients and hence, client dropouts are common during the aggregation process. Some of the above mentioned works support client dropouts, in particular, the methods based on masking [8] and MPC [20] but none considers that the aggregation servers can fail.

The existing methods cannot be applied in a fully-decentralized PDMS setting for two reasons: (i) scalability – a PDMS is not a high-end server that could deal with thousands of connections and related crypto operations, making the existing solutions not scalable with a large number of participants (e.g., the execution latency is linear [12], super linear [29] or quadratic [8] with the number of participants); and (ii) reliability – similar to the client devices (i.e., the data contributors), the aggregator nodes can drop out making the existing solutions inoperative in our context. Our protocols are also based on cryptography (i.e., using a basic secret share mechanism) but adapt to the architectural specificity and the related constraints of the PDMS context.

SA based on differential privacy and gossip protocols. Local differential privacy (LDP) has gained significant momentum in recent years due to its major advantage compared with classical DP, i.e., it does not require a trusted third party. Existing works address problems such as ML [7] , FL [1, 33, 34], marginal statistics [35] or basic statistics based on range queries [11]. However, LDP accentuates the tension between utility and privacy protection since it requires more noise for the same level of protection as with classical DP [2]. Hence, this can either affect utility or require a very large number of participants to reduce the impact of noise which contradicts adaptability to selective participation.

Gossip-based protocols are scalable, fully decentralized, reliable and have an adjustable accuracy. Unfortunately, classical gossip-based protocols do not protect the user privacy. In [7, 28], participants collectively learn a machine learning model in a privacy preserving way by gossiping differentially private models, impacting accuracy. In [23], participants introduce noise in the first iterations and gradually remove it in subsequent iterations. This approach makes such solutions unreliable w.r.t. node failures. Finally, we are not aware of gossip protocols tolerating selective participation and basic adaptations produce inaccurate results.

SA based on TEE. To overcome some of the limitations of cryptographic schemes or DP, several works propose using secure hardware at the user-side to address, e.g., SQL aggregation [32] or spatio-temporal aggregation [26]. This approach is generic w.r.t. the computation function but the existing solutions use a hybrid architecture (i.e., employ a supporting server), do not address the node selection problem and generally consider a tamper-proof attack model or a very small number of corrupted nodes.

In conclusion, none of the above classes of solutions can satisfy all the requirements of SA in a PDMS setting wherein the fully-decentralized nature of the system has to cope with unavoidable aggregator dropouts and the need for accuracy of FL.

3 SYSTEM OVERVIEW AND THREAT MODEL

3.1 System Architecture

P2P network. We envision a fully distributed P2P system relying only on PDMSs, thus requiring an efficient communication overlay. *Distributed Hash Tables* (DHT) are structured overlays which enable a logarithmic scalability with the number of nodes. Our protocol is currently built on top of the Chord DHT [31]. Each node has an *Id* obtained by hashing a static property of the node and stores a *fingertable* (FT) to route Chord messages. FT is a table with a number of entries equal to the size of the *Id* space in bits. If X is a node *Id*, the i^{th} entry of the FT contains the IP address of the node whose *Id* is closest but lower than $X + 2^i$. Routing is done by searching in the FT the closest entry to the target address and transmitting recursively the message until it reaches its target, with a worse case of $O(log(N))$ message complexity, where N is the number of DHT nodes.

Computation model. A model computation can be triggered by any node, i.e., *querier*. The querier broadcasts the computation and each node consents or not to contribute, and in the positive case is called *contributor*. The ratio between the number of contributors and total number of nodes defines the *selectivity* $\sigma \in [0, 1]$. Each node (contributor or not) may be a data processor and is then called *aggregator*. Each contributor trains the model locally for several epochs as described in [21] and sends it to the aggregators. Aggregators produce a new model based on the received contributions. The process can potentially repeat for several iterations.

3.2 Threat Model

As in the majority of SA works [20], we consider the classical *honest-but-curious* threat model, i.e., an attacker can access, but cannot alter, the data manipulated by the attacked nodes (called *leaking nodes*). A PDMS can hold the entire digital life of her owner and thus needs to be highly protected against privacy threats as indicated by recent works [4]. However, we consider that some PDMS owners have succeeded in tampering their PDMS since no security measure is unbreakable. Since attackers may collude and thus, de facto, control more than one PDMS, the worst-case attack is represented by the maximum number of colluding nodes controlled by a single "attacker", i.e., C leaking nodes. Additionally, each PDMS is equipped with a trustworthy certificate supplied by an offline PKI. Thus, any node can verify the authenticity of other participants by checking their certificate. This prevents Sybil attacks (i.e., forging nodes to master a large portion of the system). Finally, secure communication channels (e.g., TLS) are required since attackers can observe the communications between the nodes.

Our protocols should fully protect the confidentiality of the contributors' data and all the intermediary results, with high and tunable probability (see also [9]), the final result not being confidential. Also, we consider that being a contributor for a given computation is not a sensitive information.

Out-of-scope attacks. We do not consider the case of an attacker manipulating fully corrupted PDMSs. In a P2P system, such an attacker could perform poisoning attacks of the contributed data [12] or forge false aggregation results [17] with the objective to compromise contributors' input confidentiality by bypassing the SA protocol. A few recent works (e.g., Prio [12], VeriFL [17]) deal with these problems but existing solutions are still limited especially in our context because of their limited scalability or lack of genericity.

4 STRAW-MAN PROTOCOL

This section summarizes the main design principles proposed in [22] to fulfill the privacy, accuracy, adaptability and scalability properties. It then describes, as a starting point, a straw-man protocol in an ideal world, i.e., assuming there are no dropouts. Finally, it highlights the reliability issue by considering node dropouts and formulates precisely the problem at hand.

4.1 Background

Achieving a scalable aggregation process requires multiple aggregators, naturally arranged in a tree structure (see Fig. 1.a) wherein the intermediary nodes are aggregators and the leaves are contributors. The querier obtains the result from the tree root.

Privacy and accuracy: We use a secret sharing scheme without threshold for data confidentiality. Each contributor splits its private value into s shares, making it impossible to reconstruct the secret unless someone collects all s shares. Considering s shares for each contributor and partial aggregate results leads to build s separate (parallel) aggregation trees with exactly the same structure. This

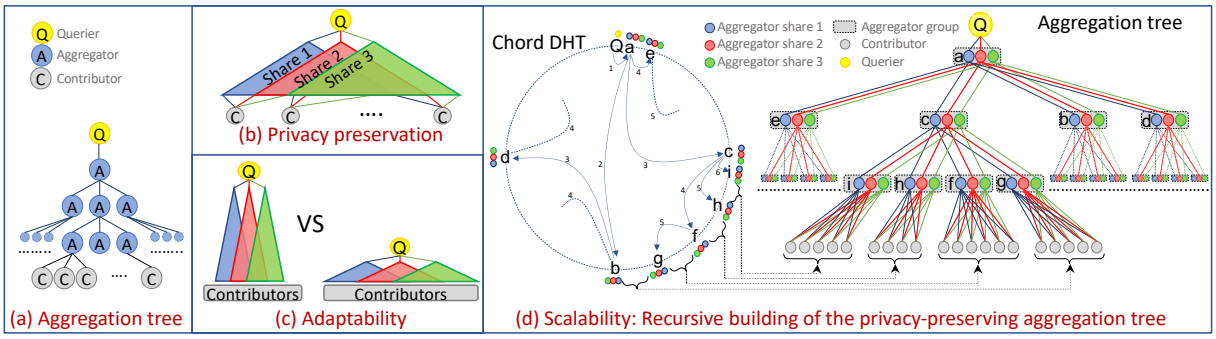

Figure 1: Building the aggregation tree based on DHT

precludes inferences from an attacker on any of the intermediate results (see Fig. 1.b). Each i^{th} share has the value $x_i = x/s + \epsilon_i$ such that $\sum_{i=1}^{n} \epsilon_i = 0$, where x is the private value. Thus, shares from different contributors are aggregated separately and if no share is missing (reliability is discussed in Section 5), the final result equals the exact sum of all private values and is computed by the querier; hence, our protocol provides, by construction, accurate results.

The number of shares, s, is computed such that the probability to obtain s shares for an attacker, controlling C nodes, is inferior to α, a security threshold (e.g., $\alpha = 10^{-6}$). An attacker could cleverly locate her controlled nodes in the DHT to obtain the s shares of a group. We avoid this attack by reusing the concept of imposed location proposed in [19]: the node *Id* in the DHT is computed by hashing the public key from the PDMS certificate (see Section 3.2). The nodes are then uniformly distributed in the DHT space and the PDMS owner (here the attacker) cannot influence this placement. Consequently, the uniform distribution also applies to leaking nodes and the probability that an attacker controls an entire group is given by $(C/N)^s < \alpha$. Then s is minimal when $s = \lceil \log(\alpha)/\log(\frac{C}{N}) \rceil$.

Obviously, communications between nodes must use secured channels, to protect the integrity and confidentiality of the exchanged data and to ensure the provenance of that data.

Adaptability: The number of aggregators and their arrangement (i.e., the tree fan-out and its height) is tuned as a function of the number of contributors, the communication costs and the processing costs as discussed in [22]. This allows the protocol to always offer near-optimal performance (i.e., aggregation latency) and achieve adaptability w.r.t. the computation selectivity and PDMSs characteristics. Furthermore, our protocol can be configured to offer the desired trade-off between the latency and the total cost of the aggregation, which are conflicting objectives. At one extreme (see Fig. 1.c left), a binary tree ($f = 2$) distributes the query load on a maximum number of aggregator groups but increases the communications costs. At the other extreme (see Fig. 1.c right), a tree limited to a unique aggregator group ($f = \sigma \times N$) minimizes the communications costs, the total system load but concentrates most of that load on this unique aggregator group that becomes overloaded. Thus, in an "ideal" setting, the height of the tree is chosen to optimize the query latency without impacting too much the total load.

Scalability: The DHT realizes a de facto fully decentralized and efficient architecture for communication between nodes. Building and broadcasting s aggregation trees can be very costly since the trees can be large. We thus employ a divide-and-conquer approach to parallelize the construction and diffusion of the trees and use the finger table structure to minimize communications. Finally, we reduce the knowledge and the diffusion of the trees to the part required to perform the aggregation: a node of an aggregation tree only knows its parent and its children.

To simplify the description of the tree construction, we consider below that each node of the tree is a group of s nodes (see Fig. 1.d with $s = 3$). Assuming the querier knows the height h and the fan-out f of the aggregation tree (see above), it starts the tree creation by assigning the whole DHT to its successors. Recursively, each aggregator group in the tree (i.e., parent nodes) is assigned to a DHT region that it will subdivide and delegate to other aggregator groups in that region. When an aggregator oversees a DHT region, it looks for f nodes that are (almost) evenly spaced across the region. The node responsible for finding peers is a parent aggregator, while the selected nodes are child aggregators. Each child then becomes the parent of the region between itself and the next sibling. This process goes on until the height h is reached. At each step, s nodes are selected instead of one. To make this selection efficient, each node in the DHT maintains a cache with the addresses and certificates of the $s-1$ successor nodes that will form the aggregator group. At the last tree level, the tree leaves (i.e., the contributors) are found by using a localized DHT broadcast in the respective region. Fig. 1.d illustrates this process with three nodes per group (blue, red and green) by using letters to represent a group. The fan-out is 4 and the height is 3 (excluding the querier Q).

4.2 Straw-man Protocol in an Ideal World

This section details a straw-man aggregation protocol, assuming an ideal world in which there are no dropouts, in order to illustrate the reliability problem. For the sake of simplicity, the presented protocol considers that the aggregation trees were built up to the leaves, but without including the contributors.

Straw-man is detailed in Algorithm 1 by type of nodes considering the computation of the average of a private vector owned by each contributor. Contributors willing to participate establish a secure channel with each aggregator parent and then send shares of their private vector. The aggregators aggregate the received shares and send their results to their parents up to the root. The querier then performs the final aggregation to obtain the result. There are only two types of messages: (i) *Query()* messages containing (1) the query itself (line 25); (2) the sender certificate (line 26), and (3) the receiver parents to whom the shares must be sent (line 28). (ii)

Algorithm 1: Straw-Man protocol (average computation)

Message definition:

- *IntRes$_i$(Sum, NbContrib)*: intermediate result of child i.
- *Query()*: ask contributors for their potential contributions.

Input: s: number of shares; f: tree fan-out

Querier Node :

1 **on** *initialization* **do** $\overrightarrow{sum} \leftarrow \overrightarrow{0}$; $nbContrib \leftarrow 0$
2 **on** *IntRes$_i$(), $i \in [1..s]$* **do**
3 $\overrightarrow{sum}$ += $msg.\overrightarrow{sum}$; $nbContrib$ += $msg.nbContrib$
4 **if** *I received s intermediate results* **then**
5 $\overrightarrow{result} = \overrightarrow{sum}/(nbContrib/s)$ /* average */

Aggregator Nodes :

6 **on** *initialization* **do** $\overrightarrow{sum} \leftarrow \overrightarrow{0}$; $nbContrib \leftarrow 0$
7 **on** *IntRes$_i$(), $i \in [1..f]$* **do**
8 $\overrightarrow{sum}$ += $msg.\overrightarrow{sum}$; $nbContrib$ += $msg.nbContrib$
9 **if** *I received f intermediate results* **then**
10 **if** *I want to contribute* **then**
11 $\overrightarrow{sum}$ += $\overrightarrow{myData}$; $nbContrib$ += s
12 Send *IntRes*($\overrightarrow{sum}$, $nbContrib$) to my parent

Leaf Aggregator Nodes :

13 **on** *initialization* **do**
14 $\overrightarrow{sum} \leftarrow \overrightarrow{0}$; $nbContrib \leftarrow 0$
15 Broadcast the query to the assigned part of the DHT
16 Set a *Contribution Timeout* (to receive all contributions)
17 **on** *IntRes$_i$(), $i \in [1..f]$* **do**
18 $\overrightarrow{sum}$ += $msg.\overrightarrow{sum}$; $nbContrib$ += $msg.nbContrib$
19 **after** *Contribution Timeout expiration* **do**
20 **if** *there is no more pending messages* **then**
21 **if** *I want to contribute* **then**
22 $\overrightarrow{sum}$ += $\overrightarrow{myData}$; $nbContrib$+=s
23 Send *IntRes*($\overrightarrow{sum}$, $nbContrib$) to my parent

Potential Contributor Nodes :

24 **on** *Query()* **do**
25 **if** *I want to contribute* **then**
26 **if** *msg. sender is a PDMS (check certificate)* **then**
27 **for** $i \in [1..s]$ **do**
28 Send *IntRes*($\overrightarrow{share_i}$, 1) to $msg.parents[i]$

Intermediate results under the format ($\overrightarrow{sum}$, *nbContrib*) sent either by contributors (line 28, with *nbContrib* = 1) or (Leaf) Aggregators (lines 12 and 23).

After having broadcasted the query, the leaf aggregators set a *contribution timeout*, computed such that it allows to receive all contributions (line 16). The timeout is computed by considering the time to reach a contributor plus the time to prepare and send a contribution since we consider an ideal world with no delays. While sent in parallel, the contributions are decrypted sequentially by the leaf aggregators, which wait for the processing of any message (line 20) before sending the partial result (line 23). If a node selected as aggregator (leaf or not) in the tree wishes to contribute, it can simply add its private data to the partial aggregate it computes add s to the count of share contributions before sending it to its parent.

4.3 Analysis and Problem Formulation

Although the straw-man protocol is simple and efficient in an ideal world, it can deliver an incorrect result or simply block in the presence of node dropouts. Indeed, one share of a contributor may not be received because the contributor drops out after sending some shares or because the corresponding message was delayed. In both cases, the result is incorrect. Furthermore, if an aggregator fails before sending its intermediate result, the condition in line 9 will never be true, thus blocking the protocol. A single aggregator dropout is indeed sufficient to thwart a graceful protocol termination since all the ancestors of the dropout node will hang on indefinitely waiting for the data to arrive.

The problem addressed in this paper is to devise protocols robust to dropouts, i.e., ensure the reliability property with three complementary goals despite failures and delays:

(1) *validity*: the protocol must deliver a correct result;
(2) *termination*: the protocol must not block;
(3) *completeness*: the protocol should maximize completeness defined as the percentage of the initial contributors actually accounted for in the final result.

Termination and validity are mandatory while maximizing completeness is a desirable objective, but may incur a significant overhead. An ideal protocol should minimize this overhead and maximize the completeness of the result, which are unfortunately conflicting goals. Indeed, maximizing completeness requires synchronization between the parallel aggregation trees and the ability to redo the work done by a dropped out aggregator. In addition, this overhead increases the latency of the protocol which can lead to increased dropouts, with, potentially, a snowball effect.

5 HANDLING DROPOUTS

This section proposes solutions to handle dropouts during the aggregation protocol. We first introduce the dropout model and detection. Then, we discuss possible approaches to react to dropouts, guarantee validity and termination.

5.1 Dropout Model and Detection

We consider the most difficult failure model wherein any node can dropout at any moment (i.e., we cannot benefit from graceful disconnections). For simplicity, in all cases, we consider that dropout nodes cannot reintegrate the ongoing computation after a dropout. When a dropout node recovers, it reintegrate the DHT or can participate in new queries.

We consider that the dropout probability is the same for any contributor or aggregator node. That is, at each time instant (e.g., every second) during the protocol execution, every node can dropout with some fixed probability, thus, the longer the protocol duration, the higher the risk of a dropout and hence the observed dropouts. In this model, there is no way to detect a dropout with certainty; a dropout can only be assumed after a timeout, i.e., node A may presume node B's dropout because A was expecting a message from B and did not receive it after a given timeout.

Let us note that the aggregation trees form a temporary additional overlay on top of the DHT overlay. Hence, to detect dropouts, we use a common DHT mechanism to maintain its consistency, i.e., health check (HC) messages. Specifically, any aggregator is periodically monitored by its parent using HC messages. HC are sent over the secure channels already required to secure the communications in the aggregation tree. HC are equivalent to ping messages so they imply a low network overhead.

5.2 Replacing (or not) a Dropped Out Node

The natural reaction to the dropout of an aggregator A is to trigger a node replacement as follow: the parent P detecting the dropout of node A randomly selects a free node, say R, from its DHT fingertable (i.e., a node that has not been previously selected as aggregator in the current tree) and supplies R the necessary information (e.g., A's children, the members of A's group, A's status, etc.) allowing the node to take the place of A. This information can be easily kept up-to-date by P when A answers P's health check requests. If the dropout occurs before A has received any data from its children, the replacement is cheap, entailing only the creation of secure communication channels between R and P, A's children and the members of A's group. In the other cases (i.e., A has done part or all of its assigned work), the replacement induces a significant overhead (R must require A's children to re-send their data, potentially redo the aggregation and re-send the aggregated data to P). Thus, all the strategies described in Section 6 replace any node dropping out before receiving any data while the replacement policy in the other cases depends on the strategy, with an impact on the overhead/completeness trade-off. Obviously, contributors that drop out cannot be replaced. Thus, a synchronization between the parallel aggregation trees must take place to ensure validity if some contributors or some –not replaced– aggregators dropped out.

5.3 Ensuring Validity

This section introduces two complementary mechanisms for ensuring result validity. The first is based on recording and checking the contributors' footprints in each of the parallel aggregation trees. The second uses inter-tree synchronization between aggregators in the same group allowing for contributors' convergence between the parallel aggregation trees.

*5.3.1 Check Contributors Footprint (**CCF**).* Validity is ensured as long as the s last *IntRes* messages (see Algorithm 1), computed by the s parallel aggregation trees and received by Q, contain the secret share contributions of the exact same list of contributors. To this end, we employ a hashing scheme similar to a Merkle Hash Tree , i.e., computing incrementally a hash of the identifier lists of the contributors whose shares are aggregated. We add a new field, *CF*, to *IntRes* messages which contains, for leaf aggregators, a hash of all contributors IPs that are included in that intermediate result. Contributors thus send *IntRes(MyShare, 1, hash(MyIP)*). Then aggregators, leaf or not, sort the incoming *CF*s, hash that sorted list to produce their own *CF* and send it with their intermediate result. The process repeats to all intermediate levels up to the querier Q. Therefore, Q can ensure the production of a valid result iff all the *CF*s received together with the s intermediate results are equal. Thus *CF* can be considered as a *version identifier* of a given *IntRes*. *CCF* allows for efficient detection of inconsistent shares but not for a convergence of those shares. Hence, the lack of even a single share leads to invalidating the entire aggregation, i.e., $completeness = 0\%$.

*5.3.2 Inter-tree Synchronization (**sync**).* To correct inconsistencies between the parallel aggregation trees, we need a synchronization mechanism to eliminate from the *IntRes* message the contributions of any contributor that provided less than s secret shares. Note that this can arrive either because of a contributor dropout but also as a result of an aggregator dropout which is not replaced. This synchronization called *sync* can be applied in a blocking or non-blocking manner as described below.

Blocking sync. A blocking sync is performed between the aggregators in a same group (e.g., the blue, red and green nodes of any group in Fig. 1.d), which synchronize their contributing children list to produce an *IntRes* containing only the data from children nodes in the intersection of those lists. If the children data was synchronized before aggregation, the resulting *IntRes* is then consistent (i.e., will have the same *CF*). Each aggregator in a group waits for all its children data and then broadcasts the list of contributing children to the other aggregators in its group. After receiving $s-1$ lists, each aggregator produces through intersection the final list, aggregates the corresponding shares and sends the result to its parent.

Non-blocking sync. The idea is to allow aggregators to send the aggregated shares up the tree without synchronization with the other $s-1$ leaf aggregators in the group, but just informing them of the actual *Children List (CL)* used to compute the *IntRes*. A leaf aggregator who receives a CL must react in different ways, depending on its own status: (a) if it has not sent any *IntRes* message, it must ignore the data from children that are not in the received CL; (b) if it has already sent an *IntRes* with its own CL (CL_{last}), it computes $CL_{new} = CL_{last} \bigcap CL$. If $CL_{new} \neq CL_{last}$, the aggregator sends a new *IntRes* message based on CL_{new} data and informs the other aggregators of its group, sending CL_{new}. Thus, if the querier receives inconsistent *IntRes* messages, it detects it through the *CF* inconsistency and just has to wait for new *IntRes* messages that will eventually become consistent.

For a leaf aggregator group (e.g., the f group in Fig. 1.d), the sync eliminates the contributors that provided only a part of the s shares. In the upper tree levels, the sync eliminates entire tree branches and therefore, possibly a significant number of contributors (e.g., if c-blue drops out and is not replaced, the sync at the a group in Fig. 1.d leads to prune the whole c sub-tree since there are no means to retrieve the blue shares in this sub-tree). Thus, sync operations may hurt completeness. A blocking sync guarantees that all the group aggregators send consistent *IntRes* up the tree and thus potentially entails a lower cost (i.e., both bandwidth and computation cost) than a non-blocking sync wherein aggregators may send several *IntRes* messages (i.e., eventual consistency). However, in strategies that replace dropped out aggregators, a replaced aggregator may require another sync with the members of its group, thus reducing the interest of a blocking sync. Moreover, blocking syncs may increase query latency since any slowdown in one of the parallel aggregation tree will impact the others. Finally, we should underline that a blocking sync does not require CCF if applied in all groups, whereas this is required for non-blocking sync.

5.4 Ensuring Termination

Ensuring query termination is straightforward insofar as dropouts are detected (see Section 5.1). The protocol can gracefully terminate if the querier receives a consistent set of s *IntRes*. In this case, the querier 'broadcasts' a termination message (i.e., which is propagated recursively down the s aggregation trees). Depending on the aggregation strategy (see Section 6) a dropout can also trigger termination. For instance, in a straw-man-like protocol a single dropout can invalidate the entire result. Hence, on detecting a dropout, an aggregator informs the querier which sends termination to all nodes. Finally, nodes within sub-trees can receive an early termination message (i.e., before the protocol end) following a sync at the sub-tree root group requiring pruning.

6 PROPOSED AGGREGATION STRATEGIES

Several strategies can be envisioned around the building blocks introduced above leading to different trade-offs between aggregation cost (i.e., the latency, total work and bandwidth of the protocol) and result completeness. In this section we first present two extreme strategies called *LowCost* and *HighCpl* trying each to push in one direction of the conflicting overhead/completeness objectives. Then, we introduce two more strategies called *Sync&Prune* and *Hybrid* which target interesting trade-offs between completeness and cost. For each strategy, we describe its overall objective, its design principles, then describe the protocol through the choice of the adequate building blocks, concluding with a short discussion which is completed by experimental results in Section 7.

6.1 Low-Cost Protocol

Objective. *LowCost* bets on an "almost ideal" world and is thus really optimistic in terms of dropouts. It leverages the straw-man protocol, correcting its main issues (validity and termination) with low-cost mechanisms to make it reliable.

Design principles. We observe that the straw-man protocol is near-optimal since (i) the data is sent up the tree only once by each node, and (ii) there is no sync between the parallel aggregation trees. The idea is to keep these good properties while still ensuring the protocol validity and termination.

Protocol. Recall that contributors transmit their s shares simultaneously to the s leaf aggregators. In case of contributor dropout, it is unlikely, but not impossible, that the shares are transmitted completely to, e.g., s-1 aggregators and incompletely to the last one.

Node replacement policy. An aggregator is replaced only if its dropout occurs before the node receives any data from any of its children. For instance, node c in Fig. 1.d can be replaced only if it drops out before receiving any data from i, f, h or g. Replacing c after this would violate the 'send-only-once' principle since one or several of its children would need to re-send data.

Validity. It is ensured by leveraging the low cost CCF mechanism which does not require any inter-tree communication.

Termination. LowCost terminates either (i) gracefully after the querier receives all s shares from its children or (ii) abruptly if any aggregator dropout is detected. Note also that due to the 'send-only-once' principle, any node can safely leave the protocol after it has sent its *IntRes* to its parent (i.e., a progressive termination from the leaves to the root).

Discussion. This basic protocol minimizes cost and is reliable. However, it has a binary behavior w.r.t. completeness. That is, completeness drops to 0% if (i) a single fatal aggregator dropout occurs or (ii) a single contributor drops out after sending only a sub-set of its s shares (thus leading to different versions in the parallel subtrees). This makes the completeness of *LowCost* extremely sensitive to dropouts. The reason is that there is no inter-tree sync mechanism allowing a convergence between the s parallel aggregates.

6.2 High-Completeness Protocol

Objective. To have a complete view of the design space, we also need a strategy that eagerly searches to maximize completeness regardless of cost. We call this strategy *HighCpl*.

Design principles. To achieve this objective *HighCpl* adopts the opposite behavior compared with *LowCost*: (i) any tree node (contributor or aggregator) can re-send its data whenever required (e.g., following a node replacement) and (ii) *HighCpl* propagates the data upward in the tree as fast as possible to maximize the chances of diffusion and then uses non-blocking sync for convergence between trees with eventual consistency thanks to CCF.

Protocol. *Node replacement policy.* In *HighCpl*, any dropped out aggregator is replaced as soon as the dropout is detected by its parent node regardless if the dropout node has already sent one or several times its data up the tree. The replacement node asks *IntRes* from its children after replacement and sends the aggregate to its parent if that aggregate has a different version (*CF*) compared with the last sent aggregate (recorded by the parent). This may happen, for instance with the replacement of a leaf aggregator with some dropped out contributors.

Validity. Non-blocking sync is required at the leaf aggregator level to ensure validity despite contributor dropouts (and a leaf aggregator replacement requires a new synchronization anyway).For the other levels, no sync is required since aggregators are replaced in case of dropouts and any new version sent at the leaf aggregator level triggers new computations up to the tree root.

Termination. HighCpl can terminate after the querier receives s consistent shares from its children.

Discussion. *HighCpl* searches to maximizes completeness through systematic dropout replacements, subsequent data re-sends, and minimalist non-blocking sync. The consequence is obviously an increased overhead since the same data can be transmitted and aggregated multiple times.

6.3 Sync-and-Prune Protocol

Objective. The extreme behavior of *LowCost* and *HighCpl* may make them impractical to use in real case scenarios. *Sync&Prune* offers an adapted trade-off between completeness and cost, trying to minimize overhead but without completely hurting completeness

Design principles. *Sync&Prune* leverages the same 'send-only-once' principle to minimize cost like in *LowCost*. However, different from *LowCost*, *Sync&Prune* allows for convergence between the parallel trees by using sync.

Protocol. *Node replacement policy.* Same as *LowCost* given the 'send-only-once' principle.

Validity. Non-blocking sync is not compatible with the 'send-only-once' principle. Thus *Sync&Prune* employs blocking sync to ensure validity at all tree levels. Indeed, since dropout aggregators are

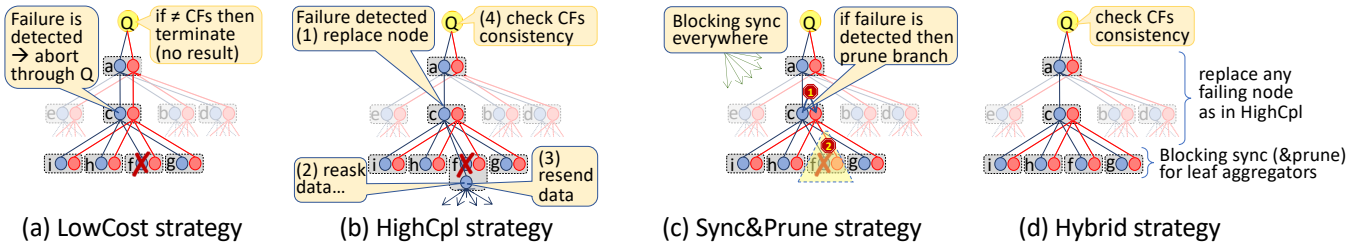

Figure 2: Envisioned strategies

not replaced, synchronization is required at all levels to ensure a consistent result between the parallel trees. Syncing at each tree level allows *Sync&Prune* to progressively prune the tree branches corresponding to dropped out aggregators from leaves to the root.
Termination. Sync&Prune terminates when the querier receives s shares from its child group or if it detects a dropout in the root group. Lower level nodes progressively terminate, from leaves to the root, after sending their *IntRes* to their parents.
Discussion. *Sync&Prune* is expected to have a low cost due to the 'send-only-once' strategy. Syncing also adds an overhead but we expect it to be low compared with the data transmission and the message decryption/encryption cost.

6.4 Hybrid Protocol

Objective. *Hybrid* is a second, less extreme strategy attempting to maximize completeness and maintain a reasonable cost.
Design principles. The idea is to have a hybrid approach combining principles from *HighCpl* and *Sync&Prune*. In *Hybrid*, the contributors employ a 'send-only-once' strategy like in *Sync&Prune*, while the aggregators re-send data whenever necessary like in *HighCpl*. The rationale is: (i) a significant part of the query cost comes from the data transmission at the contributors' level given the large number of contributors; and (ii) in the upper part of the tree, replacement induces less costs and provides comparatively more benefits, in terms of completeness.
Protocol. *Node replacement policy.* In *Hybrid*, a leaf aggregator that drops out is replaced only if the dropout occurs before the node receives any data from its contributors to comply with the 'send-only-once' principle for contributors. On the other hand, any dropout aggregator in the upper levels is systematically replaced and require their children to re-send their *IntRes*.
Validity. To minimize overhead, *Hybrid* uses a blocking sync strategy at the leaf aggregators. Thus, leaf aggregators send a single version of *IntRes* to their parent. Moreover, since leaf aggregators are not replaced in case of dropout, the sync at the leaf aggregator parent level induces pruning as in *Sync&Prune*. However, the aggregator replacements in the upper levels avoid pruning a large number of contributors and thus favors completeness. Also, replacements in the upper levels can generate multiple versions.
Termination. As in *HighCpl*, *Hybrid* can terminate after the querier receives s consistent *IntRes* from its child group. The contributor nodes and leaf aggregators nodes terminate after sending all the s shares to their parent(s).
Discussion. This hybrid strategy inherits the best of the two worlds: maximized completeness through replacement in the upper levels and limited cost due to 'send-only-once' at the contributors' level.

Strategy	send once	use sync	blocking sync	use CCF	replace agg.
LowCost	yes	no	n/a	yes	no
HighCpl	no	leaf agg.	no	yes	yes
Sync&Prune	yes	all levels	yes	no	no
Hybrid	contrib.	leaf agg.	yes	yes	yes

Table 1: Design and building blocks

However, it also inherits, but at a smaller scale, the limitations of the two approaches: some loss in completeness because of the non-replacement of leaf aggregators as well as some overhead generated by data re-sends in the upper levels.

6.5 Summary

Table 1 and Fig. 2 summarize the design and the behavior of each strategy (with only 2 shares for readability). By eagerly replacing nodes, *HighCpl* favors completeness to the expense of cost while *LowCost* does not make any significant effort to favor completeness. *Sync&Prune* still favors low overhead but avoids the binary behavior of *LowCost* by pruning subtrees. Finally *Hybrid* leverages the best of both *Sync&Prune* and *HighCpl*.

7 PERFORMANCE EVALUATION

We present the evaluation platform and used metrics in Section 7.1, then describe in Section 7.2 the experimental parameters and the system security configuration. We present and analyze the experimental results varying the dropout rate and the other parameters in Sections 7.3 and 7.4, then conclude with an analysis on the best fitted strategy depending on the context in Section 7.5.

7.1 Experimental Platform and Metrics

Our main goal is to evaluate the four proposed protocols in a large P2P PDMS system wherein the nodes are structured leveraging a Chord DHT overlay [31]. To this end, we follow the same general approach as in the related works on P2P systems [**? ? ?**], i.e., our results are based on a simulator which creates a logical DHT between simulated nodes[1]. Besides, we cannot quantitatively compare our protocols to other SA strategies (see Section 2) given the lack of similar secure aggregation solutions in P2P.

Our experimental evaluation is focused on the tension between cost and completeness in the proposed protocols for different parameters impacting security and/or performance. With respect to cost, our simulator captures the typical metrics for evaluating

[1]The simulator is available on https://github.com/JulienMirval/dissec_cozy/tree/master/simulation

distributed protocols. At the the network level, we measure the required **bandwidth** (or bytes per query) at the node or system levels. The consumed bandwidth is of particular interest especially in the FL context wherein transmitted model parameters can have a significant size (see Table 2). The required amount of **work** (or CPU time per query) at the node or system levels is equally important. Finally, we also need to estimate the **latency** to process a query. For the network, we consider network links with latency and bandwidth based on average values of domestic Internet in France [24] and add random noise to these values to be more representative of PDMSs heterogeneous connections. For the local work on each PDMS impacting the total work and the latency, we consider the most costly operations during the protocols, i.e., the cryptographic operations. To calibrate the simulator (see Table 2), we measured on a standard laptop computer equipped with Intel i5-9400H CPU @ 2.50GHz the cost of classical asymmetric encryption for signing and verifying messages, which is required to open the secure channels between communicating nodes. We also measured the time required by contributors/aggregators to process their data (e.g., encryption/decryption of a model of different sizes using AES256).

7.2 Experimental and Security Parameters

System setup and security. We consider a large P2P system of $N = 10^6$ nodes. We consider that the most powerful attacker can control up to C nodes. Our goal is to avoid data leakage with a very high probability (e.g., a value of the security threshold $\alpha < 10^{-6}$). To determine the number of shares s, we use a revised version of the formula given in Section 4.1 to account for the node replacements, i.e., $\sum_{i=0}^{1} \binom{s+1}{s+1-i} (\frac{C}{N})^{s+1-i} (1-\frac{C}{N})^{i} < \alpha$. For simplicity, we set s and deduce, depending on α and N, the maximum number of controlled nodes C. With $\alpha = 10^{-6}$, the group size of 4, 5 or 6 allows to tolerate up to, respectively, 21K, 44K or 72K colluding nodes.

Dropout rate and simulation of dropouts. We vary the dropout rate from none up to extreme values, considering the most interesting, medium range values. The no dropout case allows providing a lower bound for the cost metrics. The extreme dropout rates are not representative of a real system setup (e.g., 1% dropout rate means that all the nodes drop out –for that query– after 100 seconds!) but allows observing trends and limitations of each strategy. Note also that dropouts during a query are only related to that query (i.e., nodes may be still working correctly, e.g., for the DHT overlay). Finally, we should stress that nodes dropout are pre-computed before the query execution and independently of the strategy to produce the exact same dropouts at the same moment and allow a fair comparison of the different strategies.

System scalability. We use a fan-out of 8 for the aggregation trees since this value offers the best trade-off between latency and total work in our setting (see [22] for the fan-out tuning detail). To measure the system scalability, we consider different values for selectivity and model size. The selectivity determines the number of contributors for a query and consequently, the aggregation tree height (the tree height of 3, 4 and 5 corresponds, with a fan-out of 8, to a selectivity of respectively 0.05%, 0.4% and 3.2%). For the model size, we considered very small (1KB) to large (16 MB) models to cover a wide range of FL applications. For instance, [8, 29] consider the size of 1MB.

Description (*notation*)	Values (*default*)
Network latency [24]	30ms
Network bandwidth [24]	6MB/s
Asymmetric cryptographic operation	10ms/op
Local processing including symmetric crypto	5ms/MB
Number of PDMS nodes (N)	10^6
Group size (s), based on α, N, C	4, 5 or 6 (5)
Maximum number of replacement per group	1
Percentage of node dropouts per second (D)	0% to 1.2% (0.25%)
Aggregation tree fanout (f)	8
Aggregation tree height (H)	3, 4, or 5 (4)
ML model size (M)	1KB to 16MB (1MB)
Number of runs (to account for variability)	50

Table 2: Simulation parameters

Number of runs and box plots. We aggregate the results of 50 runs to obtain statistically representative results. In addition, we use box plots which helps visualizing this variability and the distribution of runs. In the figures, the lower and upper whiskers of the box respectively represent the min and the max for the plotted metric, and the lower and upper edges of the box represent first and third quartile respectively. To make the boxes easier to read, we exclude outliers (i.e., points that are further away than 1.5 the interquartile range) which are directly represented as points. Finally, the line connecting the mean values is also represented.

7.3 Performance with Varying Dropout Rate

Fig. 3, 4, 5 and 6 depict respectively the completeness, latency, bandwidth and work by node, varying the dropout rate D All strategies perform identically when there are no dropouts since the overhead of advanced strategies (e.g., blocking sync) is negligible compared to the transmission cost of the 1MB model.

***LowCost* cannot handle even a few dropouts**: As soon as some nodes drop out, *LowCost* fails to obtain any result. The strategy uses an abrupt termination (see Section 6.1), sometimes even before contributors have a chance to contribute data. Fig. 5 shows that despite early termination, some contributors do manage to send their shares, which consumes some bandwidth even though these data are never used. In the rest of the study, *LowCost* is often ignored because we focus on settings where dropouts prevent it from succeeding.

***HighCpl* has the best completeness but it degrades rapidly with large dropout rates**: *HighCpl* degrades rapidly when $D \geq 0.4$ with a variability between runs that explodes. Indeed, the overhead induced by dropout handling increases significantly with the dropout rate (see Fig. 4 and 5). *HighCpl* gives up replacing nodes after having exhausted the maximum number of replacement nodes. On the one hand, this allows maintaining a path between the remaining contributors and the root of the tree, which in turn allows for aggregates to move faster up the aggregation path. On the other hand, because data are eagerly sent along the parallel trees without explicit synchronization mechanisms, it can generate a lot of aggregate re-sends before the final CCF convergence, impacting work, latency and bandwidth. Hence, *HighCpl* is subject to a snowball effect making it unsuitable for large dropout rates.

***Sync&Prune* has stable cost but lower completeness**: Opposite to *HighCpl*, *Sync&Prune* starts out with a lower completeness and

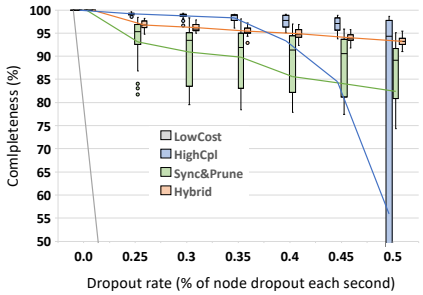

Figure 3: Completeness, 1 MB model

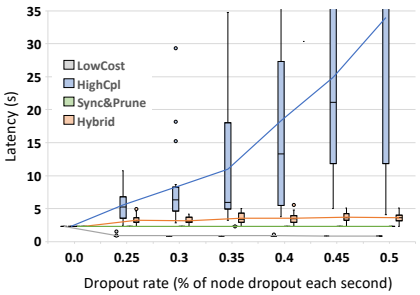

Figure 4: Latency, 1 MB model

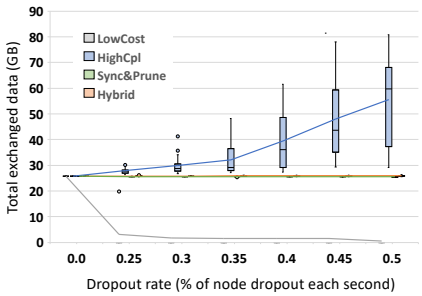

Figure 5: Bandwidth, 1 MB model

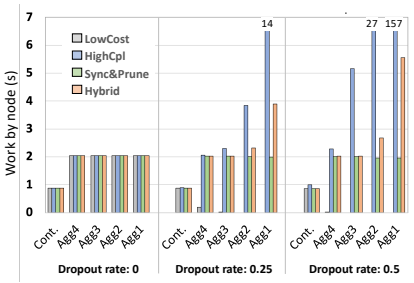

Figure 6: Work by node, 1 MB model

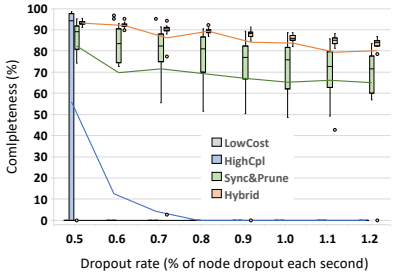

Figure 7: Extreme dropout rates

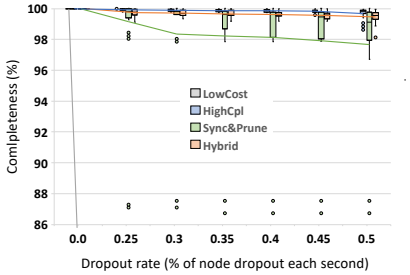

Figure 8: Completeness, 1 KB model

higher dispersion. The rationale is that for *Sync&Prune*, completeness is determined by the location of dropouts in the tree, with dropouts higher in the tree severely affecting it. Also, *Sync&Prune* has a more stable performance on all metrics as the dropout rate increases. Since the strategy ensures that any node works at most once, dropouts reduce the load of their parents. Thus, this strategy is beneficial for cost, but has a negative impact on completeness.

***Hybrid* combines the best of the two previous strategies**: *Hybrid* has a stable completeness, work and bandwidth throughout the dropout rate spectrum. Compared with *HighCpl*, the completeness is lower for low dropout rates because contributors never resend their data, but a lot better for high dropout rates because leaf aggregators are never replaced, preventing the snowball effect. To our surprise, *Hybrid* has only about $0.8 - 1.7\%$ larger communication cost than *Sync&Prune* but up to 20% higher latency. The rationale is that at the leaf aggregators level, where the vast majority of the protocol bandwidth is consumed, both protocols behave the same. In upper levels, *Hybrid* can resend data, but this has also a limited impact due to blocking **sync**.

***HighCpl* suffers from too many aggregate versions**: Fig. 6 shows the distribution of the work across the tree layers, i.e., the average work per node in each tree level, for each strategy. We observe first that the work per aggregator in all levels is similar when the dropout rate is low. Thanks to our tree structure, the load is effectively and fairly distributed across system nodes.

Contributors have less work to do in this setting since they transmit $s = 5$ shares while aggregators receive and process $f = 8$ shares and send one more to their parent. We also observe that the snowball effect in *HighCpl* mainly affects the aggregators closer to the root since those nodes receive a large number of different aggregate versions. *Hybrid* also concentrates the load on higher level aggregators but manages to keep this overhead in a reasonable range thanks to the synchronization at the leaf aggregators.

***Hybrid* and *Sync&Prune* tolerate extreme dropout rates**: Fig. 7 presents the behavior of our strategies with extreme dropout rates ($D \geq 0.5$). With dropout rates $D > 0.7$, *HighCpl* fails to obtain any result for most of its executions, but *Sync&Prune* and *Hybrid* only have a small reduction in completeness because they are pruning the nodes and branches that prevent the execution from finishing. We note that despite wider boxes for *Sync&Prune*, *Hybrid* has some outlying executions (even with no results) that fail to complete. Indeed with extreme dropout rates, the maximum number of replacements can be reached leading to pruning entire sub-trees and thus, reducing drastically completeness.

***HighCpl* and *Hybrid* work well with small models**: Fig. 8 shows the obtained completeness with a small model of only 1KB. Small models change the protocol bottlenecks since the model transmission becomes less impacting. Moreover, since the overall latency is much smaller (not shown given to the space limitation), there are less dropouts and thus better completeness for all strategies. *HighCpl* and *Hybrid* obtain almost 100% completeness with low overhead (not shown). Fig. 8 shows also something interesting: we can observe a set of outlier executions for *Sync&Prune* with a completeness of about 87% (i.e., missing a fraction of 1/8 of the results). This is explained by the dropout of an aggregator in the second level group happening after its children have sent some data and is generally the reason for the higher variability of completeness in the executions of *Sync&Prune*.

7.4 Scalability and Security

Very large models require less dropouts: For a model size of 16 MB, none of the strategies reaches a completeness above 50%

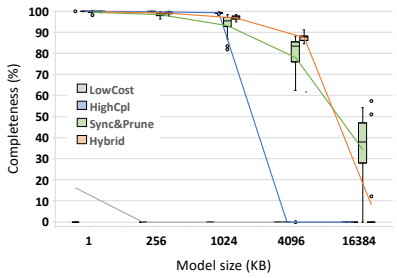

Figure 9: Varying model size

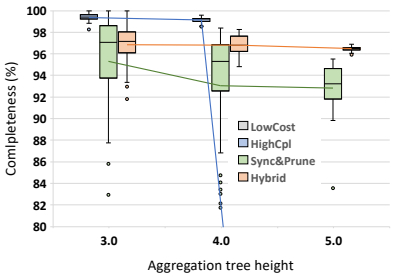

Figure 10: Varying tree height

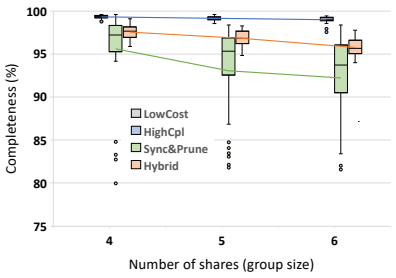

Figure 11: Varying security

LowCost HighCpl Sync&Prune Hybrid

Height	Size	0	0.01	0.25	0.5	1
3	1 K	100%	100%	100%	100%	100%
4	1 K	100%	100%	100%	100%	99%
3	1 M	100%	100%	99%	99%	96%
4	1 M	100%	100%	99%	93%	84%
3	4 M	100%	100%	97%	78%	59%
4	4 M	100%	100%	87%	68%	28%

CRITERIA: Best completeness

Figure 12: Strategy / Completeness

LowCost HighCpl Sync&Prune Hybrid

Height	Size	0	0.01	0.25	0.5	1
3	1 K	100%	100%	100%	99%	97%
4	1 K	100%	95%	99%	98%	91%
3	1 M	100%	80%	89%	90%	87%
4	1 M	100%	100%	93%	83%	84%
3	4 M	100%	100%	81%	60%	59%
4	4 M	100%	100%	78%	51%	28%

CRITERIA: Least total work, at least 80% of best completeness

Figure 13: Strategy / 80% Completeness

LowCost HighCpl Sync&Prune Hybrid

Height	Size	0	0.01	0.25	0.5	1
3	1 K	100%	100%	100%	99%	97%
4	1 K	100%	95%	99%	98%	99%
3	1 M	100%	100%	97%	99%	96%
4	1 M	100%	100%	97%	93%	84%
3	4 M	100%	100%	97%	76%	59%
4	4 M	100%	100%	87%	68%	28%

CRITERIA: Least total work, at least 95% of best completeness

Figure 14: Strategy / 95% Completeness

as shown in Fig. 9. We can see that *Sync&Prune* starts outperforming *Hybrid*, which only manages to finish a few low completeness (10%) executions. This is linked to the snowball effect caused by aggregator replacements: even if most nodes are not replaced (i.e., the leaf aggregators), the replacements upper in the tree are enough to create conflicts in the aggregate versions, which cannot be resolved since models transmission takes time, thereby leading to more dropouts. *Sync&Prune* is outperforming here because it only sends once the data that has been explicitly agreed upon by all group members, and prunes branches that fail to come to an agreement. More generally, we can conclude that very long queries (due to the transmission of 16 MB model) are not compatible with the default dropout rate (with too many dropouts) while 4 MB models are correctly supported.

***Hybrid* and *Sync&Prune* support large aggregation trees**: Another important aspect of scalability is the number of participants contributing to the query, which is characterized by the selectivity σ in our system (appearing as the tree height). *HighCpl* is once again failing when the height is greater than 4, taken down by the snowball effect of 8 times more nodes constantly resending new aggregate versions and never converging to a stable version. *Sync&Prune* and *Hybrid* on the other hand are scalable with increasing number of contributors showing that they are able to fully benefit from the distributed nature of the execution.

Larger groups (better security) are well supported: Finally, we study the impact of the security parameter on all strategies. We can see in Fig. 11 that increasing the security only has a mild impact on completeness for *Sync&Prune* and *Hybrid* while *HighCpl* remains mostly unaffected. The opposite is however happening in terms of work per node. Increasing the group size makes synchronization harder for *Sync&Prune* and *Hybrid*, resulting in more frequent pruning. The security parameter also impacts *HighCpl* by requiring more work to make the parallel trees come to an agreement.

7.5 Best Fit Strategy

In this last section, we first choose some typical parameters settings that best exemplify realistic scenarios, then define different objectives that guide us in our comparisons of the strategies. Finally, we report in Fig. 12, 13 and 14 the region of the parameter space where each strategy performs best. We study two heights for the trees, corresponding to aggregating data from 500 and 4000 contributors. It includes the default height as well as a smaller height since they are representative for the context of federated learning (i.e., it is pertinent to aggregate from less contributors more frequently). We choose three model sizes: one for tiny models (e.g., for training simple models or computing aggregate statistics), and the default value of 1MB and then 4MB (which correspond to models used in a wider range of applications, ranging from image classification to natural language processing). The last parameter that we vary is the dropout rate in order to observe the robustness of each strategy.

Our first objective is simply the highest completeness. Since a marginal increase in completeness can be costly, and unfairly advantage a strategy, we select, for the second and third objective, the strategy that incurs the least work after a pre-selection of strategies reaching a completeness of 80% and 95% of the strategy with the best completeness. We present the results in Fig. 12, 13 and 14 in the form of a table where rows correspond to a couple of height and model size and columns to a dropout rate. The color of each cell corresponds to the strategy that best fits the given objective while the value is the averaged completeness of 50 executions.

***LowCost* best fits with very few or no dropouts**: Without dropouts, *LowCost* always wins. All strategies have 100% completeness in this case but *LowCost* has the lowest cost since parallel trees are not synchronized. The same happens with very few dropouts and small models.

***HighCpl* maximizes completeness but is costly**: In Fig. 12, *HighCpl* is the strategy that offers the best completeness in a majority of cases, i.e., except for bigger models or extreme dropout rates. It is also the best strategy in few settings on Fig. 14.

***Sync&Prune* is efficient and reaches 80% best completeness**: When being more tolerant about completeness, *Sync&Prune* often becomes preferable because of its lower cost. With 80%, it even outperforms in almost all cases. *Hybrid* performs better for extreme dropout rates because maintaining the tree prevents pruning the most impacting aggregators. With 95%, *Sync&Prune* can still outperform in some cases (Fig.14), although in those cases, all strategies perform well anyway or oppositely, *Sync&Prune* is the only strategy that gets a result.

***Hybrid* is efficient and reaches 95% best completeness**: *Hybrid* fulfills its role as a trade-off strategy for every objective. It mostly outperforms in settings where *HighCpl* suffers from the snowball effect and *Sync&Prune* already has degraded performances. Moreover, for high completeness (i.e., the 95% objective), *HighCpl* has smaller overhead compared with *Sync&Prune*.

8 CONCLUSION

Personal Data Management Systems arrive at a rapid pace allowing users to share their personal data within large P2P communities, which opens exciting perspectives. Federated learning is a prime example that could benefit from this abundant, diverse and complete source of personal data to train high quality ML models. However, this requires new protocols that protect the users' privacy and are adapted to the fully-decentralized nature of the PDMS ecosystem. To this end, we proposed a set of secure aggregation protocols for federated learning which are fully-decentralized, scalable, accurate and reliable. We analyzed the secure aggregation problem in the P2P PDMS context and showed that reliability is a key aspect raising a tension between the potential completeness of the result and the aggregation cost. We then proposed four protocols having different trade-offs between completeness and cost. We extensively evaluated these protocols for a wide range of settings of the dropout rates, security setting, trained model size, or contributors' selectivity. Our results showed that these protocols can offer high completeness results at reasonable cost in a wide range of settings.

ACKNOWLEDGEMENT

This work has been supported by the ANR 22-PECY-0002 IPOP (Interdisciplinary Project on Privacy) project of the Cybersecurity PEPR.

REFERENCES

[1] Naman Agarwal, Ananda Theertha Suresh, Felix X. Yu, Sanjiv Kumar, et al. 2018. cpSGD: Communication-Efficient and Differentially-Private Distributed SGD. In *NeurIPS*.
[2] Mário S. Alvim, Konstantinos Chatzikokolakis, Catuscia Palamidessi, and Anna Pazii. 2018. Local Differential Privacy on Metric Spaces: Optimizing the Trade-Off with Utility. In *IEEE CSF*.
[3] Nicolas Anciaux, Philippe Bonnet, Luc Bouganim, Benjamin Nguyen, et al. 2019. Personal Data Management Systems: The Security and Functionality Standpoint. *Information Systems* (2019).
[4] Nicolas Anciaux, Luc Bouganim, Philippe Pucheral, Iulian Sandu Popa, et al. 2019. Personal Database Security and Trusted Execution Environments: A Tutorial at the Crossroads. *PVLDB* (2019).
[5] Yoshinori Aono, Takuya Hayashi, Lihua Wang, Shiho Moriai, et al. 2017. Privacy-Preserving Deep Learning via Additively Homomorphic Encryption. *IEEE Trans. Inf. Forensics Secur.* (2017).
[6] Johes Bater, Gregory Elliott, Craig Eggen, Satyender Goel, et al. 2017. SMCQL: Secure Query Processing for Private Data Networks. *PVLDB* (2017).
[7] Aurélien Bellet, Rachid Guerraoui, Mahsa Taziki, and Marc Tommasi. 2018. Personalized and Private Peer-to-Peer Machine Learning. In *AIStat*.
[8] Keith Bonawitz, Vladimir Ivanov, Ben Kreuter, Antonio Marcedone, et al. 2017. Practical Secure Aggregation for Privacy-Preserving Machine Learning. In *ACM CCS*.
[9] Luc Bouganim, Julien Loudet, and Iulian Sandu Popa. 2023. Highly Distributed and Privacy-Preserving Queries on Personal Data Management Systems. *The VLDB Journal* (2023).
[10] EU Commission. 25 October 2020. Proposal for a Regulation on European Data Governance (Data Governance Act), COM/2020/767. [eur-lex].
[11] Graham Cormode, Tejas Kulkarni, and Divesh Srivastava. 2019. Answering Range Queries Under Local Differential Privacy. *PVLDB* (2019).
[12] Henry Corrigan-Gibbs and Dan Boneh. 2017. Prio: Private, Robust, and Scalable Computation of Aggregate Statistics. In *NSDI*.
[13] Cozy Cloud. 2023. *Cozy Cloud (See https://cozy.io/fr/).*
[14] Ye Dong, Xiaojun Chen, Kaiyun Li, Dakui Wang, et al. 2021. FLOD: Oblivious Defender for Private Byzantine-Robust Federated Learning with Dishonest-Majority. In *ESORICS*.
[15] Hossein Fereidooni, Samuel Marchal, Markus Miettinen, Azalia Mirhoseini, et al. 2021. SAFELearn: Secure Aggregation for Private Federated Learning. In *IEEE SPW*.
[16] David Froelicher, Juan Ramón Troncoso-Pastoriza, Joao Sa Sousa, and Jean-Pierre Hubaux. 2020. Drynx: Decentralized, Secure, Verifiable System for Statistical Queries and Machine Learning on Distributed Datasets. *IEEE Trans. Inf. Forensics Secur.* (2020).
[17] Xiaojie Guo, Zheli Liu, Jin Li, Jiqiang Gao, et al. 2021. VeriFL: Communication-Efficient and Fast Verifiable Aggregation for Federated Learning. *IEEE Trans. Inf. Forensics Secur.* (2021).
[18] Peeyush Gupta, Yin Li, Sharad Mehrotra, Nisha Panwar, et al. 2019. Obscure: Information-Theoretic Oblivious and Verifiable Aggregation Queries. *PVLDB* (2019).
[19] Julien Loudet, Iulian Sandu Popa, and Luc Bouganim. 2019. SEP2P: Secure and Efficient P2P Personal Data Processing. In *EDBT*.
[20] Mohamad Mansouri, Melek Önen, Wafa Ben Jaballah, and Mauro Conti. 2023. SoK: Secure Aggregation Based on Cryptographic Schemes for Federated Learning. *PETS* (2023).
[21] Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, et al. 2017. Communication-Efficient Learning of Deep Networks from Decentralized Data. PMLR.
[22] Julien Mirval, Luc Bouganim, and Iulian Sandu-Popa. 2021. Practical Fully-Decentralized Secure Aggregation for Personal Data Management Systems. In *SSDBM*.
[23] Yilin Mo and Richard M Murray. 2016. Privacy Preserving Average Consensus. *IEEE TACON* (2016).
[24] nPerf. [n. d.]. Baromètre des Connexions Internet Fixes en France Métropolitaine. https://perma.cc/DP8V-5ABT.
[25] Amaury Bouchra Pilet, Davide Frey, and François Taïani. 2019. Robust Privacy-Preserving Gossip Averaging. In *SSS*.
[26] Iulian Sandu Popa, Dai Hai Ton That, Karine Zeitouni, and Cristian Borcea. 2021. Mobile Participatory Sensing with Strong Privacy Guarantees using Secure Probes. *GeoInformatica* (2021).
[27] Bartosz Przydatek, Dawn Song, and Adrian Perrig. 2003. SIA: Secure Information Aggregation in Sensor Networks. In *SenSys*.
[28] César Sabater, Aurélien Bellet, and Jan Ramon. 2022. An Accurate, Scalable and Verifiable Protocol for Federated Differentially Private Averaging. *Mach. Learn.* (2022).
[29] Jinhyun So, Başak Güler, and A Salman Avestimehr. 2021. Turbo-Aggregate: Breaking the Quadratic Aggregation Barrier in Secure Federated Learning. *IEEE JSAIT* (2021).
[30] Jinhyun So, Chaoyang He, Chien-Sheng Yang, Songze Li, et al. 2022. Lightsecagg: A Lightweight and Versatile Design for Secure Aggregation in Federated Learning. *MLSys* (2022).
[31] Ion Stoica, Robert Morris, David Karger, M Frans Kaashoek, et al. 2001. Chord: A Scalable Peer-to-Peer Lookup Service for Internet Applications. *ACM SIGCOMM* (2001).
[32] Quoc-Cuong To, Benjamin Nguyen, and Philippe Pucheral. 2016. Private and Scalable Execution of SQL Aggregates on a Secure Decentralized Architecture. *ACM TODS* (2016).
[33] Aleksei Triastcyn and Boi Faltings. 2019. Federated Learning with Bayesian Differential Privacy. In *IEEE BigData*.
[34] Ge Yang, Shaowei Wang, and Haijie Wang. 2021. Federated Learning with Personalized Local Differential Privacy. In *IEEE ICCCS*.
[35] Zhikun Zhang, Tianhao Wang, Ninghui Li, Shibo He, et al. 2018. CALM: Consistent Adaptive Local Marginal for Marginal Release under Local Differential Privacy. In *ACM CCS*.