Handling Dropouts in Federating Learning with Personal Data Management Systems

Julien Mirval, Luc Bouganim, Iulian Sandu Popa

Handling Dropouts in Federating Learning with Personal Data Management Systems

Julien Mirval[1,2,3][0009−0004−4580−5651], Luc Bouganim[2,3][0000−0002−2273−9987], and Iulian Sandu Popa[3,2][0000−0002−9937−4242]

[1] Cozy Cloud, "Le Surena" face au 5 Quai Marcel Dassault, 92150 Suresnes, France
{firstname.name}@cozycloud.cc
[2] Inria de Saclay, Campus de l'école Polytechnique, 1 rue Honoré d'Estienne d'Orves, 91120 Palaiseau, France
{firstname.name}@inria.fr
[3] Université de Versailles Saint-Quentin, 45 avenue des Etats-Unis, 78000 Versailles, France
{firstname.name}@uvsq.fr

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.

Keywords: Secure aggregation · Peer-to-peer · Reliability · Federated learning

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 [19], 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 [26].

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 [41], as evidenced by the large body of recent work in this area [40]. FL is defined as machine learning massively distributed on edge devices, where each participant holds its own private data. 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, 13], 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 SA protocols fail to achieve this objective. On one hand, approaches such as local differential privacy are based on adding

noise to protect privacy. This affects accuracy [9] or reliability to dropouts [51] 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 [40] (e.g., encryption-based [8, 21] or secret sharing-based [32, 17, 12]), 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 [12, 17]. 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 [12] 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 [45], where we designed a protocol that fulfill the above properties in an ideal setting, i.e., without considering the reliability issue. In [44], we focus on the reliability property, which is difficult to guarantee in a fully-decentralized setting and deserves a detailed study. In addition, 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). This paper is an extended version of [44] with a detailed state of the art and a deeper dive into the experimental platform used and the reflections that went into it, leading to more extensive experiments and performance graphs.

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 work w.r.t. the required properties in Section 2. We introduce, in Section 3, the consid-

ered architecture and threat model. Section 4 reminds the main design principles proposed in [45] 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 Work

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

2.1 Differential privacy-based Secure Aggregation

The concept of differential privacy, initially proposed in [22], centers on the premise that computations on two nearly identical datasets, differing by just one element, should yield results that vary minimally, within a small margin ϵ. Traditional differential privacy models relied on a central authority to collect and process data, ensuring privacy. However, this dependency on a trusted central entity led to the development of Local Differential Privacy (LDP), introduced in [35], which has its roots in the randomized response technique, formulated in [72]. It allows individuals to modify their data locally before sharing. Existing works address problems such as ML [9], FL [69, 1, 73], marginal statistics [76] or basic statistics based on range queries [16].

Despite the practical applications of differential privacy in various fields, as demonstrated in [24, 20], designing algorithms that strike a balance between data utility and privacy remains challenging. The inherent noise addition in LDP however accentuates the tension between utility and privacy protection since it requires more noise for the same level of protection as with classical DP [2]. Consequently, LDP is typically more suited to scenarios involving an extremely large number of participants to counterbalance the noise and where noisy data is acceptable.

Gossip-based protocols, known for their scalability, decentralization, and reliability, typically do not safeguard user privacy. In recent studies [9, 56], participants preserve privacy by sharing differentially private models or introducing noise in initial iterations and canceling it over time. Indeed, the absence of a central party of LDP fits well into fully decentralized networks like gossip. However, these techniques have a negative impact on utility and do not take advantage of high-quality PDMS data.

2.2 Anonymity-based Secure Aggregation

One method to process user data while preserving their privacy involves applying anonymization techniques before computation. This involves uniquely altering the data to prevent the extraction of personal information, yet still enabling computations. Techniques like k-anonymity [65], l-diversity [39], and t-closeness [36] have been proposed to anonymize datasets that contain information from multiple individuals. This allows for the analysis of such datasets without compromising the privacy of the users involved. However, this cannot be used in networks of PDMS because it could introduce privacy risks when grouping data from multiple nodes or if only applying it locally, or it will require strong data degradation.

Mixing techniques have also attracted a lot of interest from the research community. The *Encode-Shuffle-Analyze* framework [11] proposes a way to enable traditional database analysis on anonymous data. [28] proposes a summation protocol working with only a few anonymized messages. However, these techniques do not support nodes dropping out.

A line of work [38, 13, 37] achieves targeted statistical queries on decentralized networks of PDMS. It focuses on privacy-preserving targeting of participants in a context where small aggregates are computed. Because it only targets a sample of the target population for privacy reasons, this framework is not usable to include data from all relevant participants. Moreover, it neglects the impact of the size of the aggregate because it looks at statistical queries, not machine learning models.

Anonymity-based SA focuses on the anonymization of data pertaining to multiple users, intending to conceal each user into the masses. However, this necessarily implies a degradation of user data in case of failures and is incompatible with our objectives of resilience and maximizing the usage of the high-quality data stored by PDMS.

2.3 Trusted Execution Environment-based Secure Aggregation

Secure hardware has been used for a long time, but generally for extremely limited scopes. With the recent advances of *Trusted Execution Environments* (TEEs), new hopes have been given of widening the applications of secure hardware. TEEs are hardware solutions allowing to run a program on a machine in an isolated way. This isolation is characterized by a protection against tamper of said program and against observation of the processed data including from privileged software such as the operating system. Recently, TEEs have been included inside many different models of processors embedded in smartphones, computers and servers, opening new possibilities of ensuring security on a potentially insecure device. Intel Software Guard Extensions (Intel SGX) [18] and Arm TrustZone [5] are the leading examples of TEEs. TEEs enhance security in computing by creating an isolated processing environment [57] by encrypting and keeping track of the integrity of the RAM pages used by the code [30]. While

being executed inside the TEE, the isolated code can produce a proof of its identity (i.e., attestation) certifying that it is running inside a genuine TEE. Secure hardware on the user side could be used for tasks such as SQL aggregation [68], spatio-temporal aggregation [58], or even to speed-up FL [34] with asynchronous aggregation [48].

However, the security of TEEs is not perfect and several side-channel attacks have been showed. The first type of side-channel attacks target the inner mechanisms of TEEs [15, 60], breaking data confidentiality. Some [15, 59] even extract the attestation key of the TEE, which enables forging fake attestations. Another type of side-channel attack exploits the design of TEEs, using the untrusted rich execution environment running alongside the TEE to extract information manipulated by programs running in the TEE [61, 14]. Research work has been carried out to mitigate the impact of these attacks [50, 25]. While this shows that some security flows of TEEs can be corrected, the fallibility of TEEs cannot be ignored.

Ultimately, none of the existing solutions fully meet the requirements for SA using TEE in a PDMS environment. This environment must deal with the inherent challenges of a fully decentralized system, such as inevitable aggregator dropouts and the necessity for accurate FL. Current solutions rely too heavily on the assumption that TEEs are effectively tamper-proof when past research suggests that risks still exist and could lead to catastrophic privacy leaks if exploited.

2.4 Cryptography-based Secure Aggregation

Cryptographic solutions for SA have been proposed for nearly three decades. The early solutions were designed for wireless sensor networks [52], but the field has recently taken off again to meet the needs of federated learning. A recent survey [40] discusses about forty works for FL, grouped in either encryption-based SA or MPC-based SA.

Encryption-based Secure Aggregation

Encryption-based SA can be done using various schemes, including additively homomorphic encryption and masking. It involves encrypting private data using cryptographic keys, making the data unusable for individuals who do not have the proper key.

Additively Homomorphic Encryption (AHE) is a property of some cryptography systems which enables computations to be carried out directly on the ciphertext instead of having to decrypt it beforehand, preserving data confidentiality. They were first described in [55] as follows: Given a cryptosystem where E is the encryption function and D the decryption function, this system is said to be additively homomorphic if $E(m_1 + m_2) = E(m_1) \oplus E(m_2)$ where m_1, m_2 are messages. AHE enables highly secure computations but generally comes at a great cost penalizing efficiency. Some schemes have reported up to ten times worse performances [75] compared to the plain text alternative. Moreover, some

key management issues can arise which are difficult to manage in the presence of dropouts.

Masking consists of aggregating masks on top of private data, such that the data is kept private unless the aggregation reaches a point where masks cancel each other out. It is one of the key tools used in SecAgg [12, 62, 63]. Masks are generally much cheaper to generate compared to additively homomorphic cyphertexts and are therefore easier to scale. Nonetheless, handling dropouts implies managing masks and their removal, which can be impractical in a peer-to-peer context.

Multi-Party Computation-based Secure Aggregation

Unlike encryption-based SA, Multi-Party Computation (MPC)-based SA does not use cryptographic keys to preserve data confidentiality but instead uses private messages sent to other participants according to specific protocols. Best-known members of this category include garbled circuits or computations on secret shares.

Garbled circuits transform computations into logical circuits that are then "garbled" by each party so that they can privately exchange how the circuit should be evaluated without revealing their private inputs. The problem of schemes based on garbled circuits (e.g., [8, 21]) is that they require a lot of messages, which prevents scaling to large sets of participants and does not handle failures efficiently.

Computations on secret shares involve transforming private data into smaller pieces of secret that independently reveal nothing but can be recomposed into the original value. Using mathematical properties of these secret shares similar to homomorphic encryption enables the creation of more complex systems (e.g., [32, 27]). Some secret sharing schemes are natively redundant, making handling dropouts easier. In some cases, computations on secret shares are analogous to SA using masking.

We can see that Encryption-based SA, with techniques like additively homomorphic encryption and masking, provides robust security by allowing computations on encrypted data. However, these methods often come with trade-offs in terms of efficiency and complexity in key management. On the other hand, MPC-based SA offers an alternative approach, leveraging garbled circuits and computations on secret shares to maintain data confidentiality without the use of cryptographic keys. Despite their potential for scalability and handling dropouts, these methods also face challenges in terms of communication overhead and computational complexity.

Given the scarcity of resources in large networks of PDMS compared to what centralized service providers can offer, encryption-based SA appears as a better alternative. Indeed, the communication overhead introduced by MPC-based SA do not scale well to the number of participants that we envision in our context. More specifically, we favor masking techniques over homomorphic encryption as it reduces costs even further, by avoiding the use of expensive cryptographic operations.

2.5 Conclusion

In this section, we delved into the various secure aggregation techniques relevant to PDMS environments. Differential privacy-based SA works with noisy private data to protect individual data points but struggles with the trade-off between data utility and privacy. Anonymity-based SA, while useful in some contexts, can lead to data degradation and can be challenging to apply in PDMS networks. Trusted Execution Environment-based SA leverages secure hardware solutions but is not without its security vulnerabilities. Lastly, MPC-based SA offers interesting solutions in terms of security but faces challenges in efficiency, scalability, and handling node dropouts caused by the large communication overhead it introduces.

In summary, encryption-based SA, and particularly masking, stands out as the most fitting technique for PDMS networks, opening the way for solutions that are not only secure and privacy-preserving but also practical in terms of resource utilization and scalability. This explains its popularity in federated learning. However, existing methods using masks rely on high-end highly available aggregation servers which is not feasible in PDMS networks, and perform poorly for a large number of participants and dropouts (i.e., masks, although less costly to manage compared to MPC-based SA, must nonetheless be carefully handled to obtain accurate results).

In a nutshell, the existing secure aggregation methods cannot be applied in a 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; 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. Adapting secure aggregation techniques to the architectural specificity and the related constraints of the PDMS context is therefore necessary.

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 [64]. 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 $\mathcal{O}(log(N))$ 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 [41] 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 [40], 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 [13]), 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 [17] or forge false aggregation results [31] with the objective to compromise contributors' input confidentiality by bypassing the SA protocol. A few recent works (e.g., Prio [17], VeriFL [31]) 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 [45] 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.

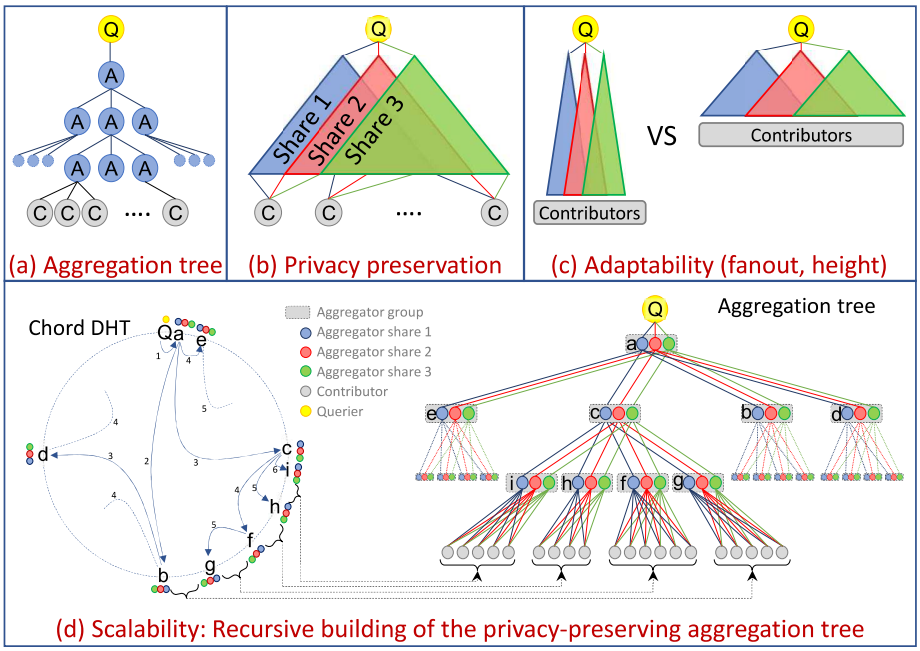


Fig. 1: Building the aggregation tree based on DHT

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 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 [38]: 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 [45]. 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) 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

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 $msg.parents[i]$

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 design 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.

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 [42] , 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., $completeness = 0\%$.

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 Aggregation Strategies

Having laid out the main building blocks that we intend to study, we now move to the topic of assembling and composing them to form coherent strategies. Since our building blocks are never a panacea and always present trade-offs between aggregation cost (i.e., the latency, total work, and bandwidth of the protocol) and result completeness, the goal is to provide Pareto-efficient strategies for different cost minimization and completeness requirements. The strategies introduced below are not exhaustive of what can be built using these building blocks and more building blocks can be created to propose new strategies but, based on our experiments, the proposed strategies are the best combination of our building blocks.

We start by introducing the most extreme strategies called *LowCost* and *HighCpl* , which respectively try to minimize costs and maximize the result completeness. Then, using those strategies as starting points, we introduce two trade-off strategies called *Sync&Prune* and *Hybrid* that try to mitigate the weaknesses of extreme strategies. For each strategy, we describe its overall objective, its design principles, then describe the protocol through the choice of adequate building blocks.

6.1 *LowCost*

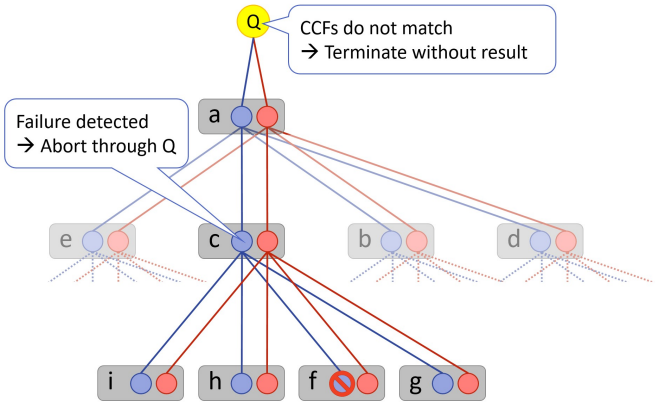


Fig. 2: *LowCost* strategy

Objective

LowCost is a radically optimistic strategy: it bets on an "almost ideal" world where little to no dropouts ever occur. It leverages the straw-man protocol explained in algorithm 1, correcting its main issues (unchecked validity and lack of termination) with low-cost mechanisms to make it reliable. By saving on checks and verifications, the strategy is able to effectively minimize costs.

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 desirable properties while still ensuring the protocol's validity and termination.

Protocol

Recall that contributors transmit their s shares simultaneously to the s leaf aggregators. In case of a 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. 2 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, unlike any form of *sync*.
– **Termination.** *LowCost* terminates either (i) gracefully after the querier receives all s shares with consistent footprints from its children or (ii) abruptly if a child is asked to resend data by its replaced parent aggregator. Abrupt termination is done by having replacement node alert the querier, and they then both propagate the alert for other nodes to stop. 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 *HighCpl*

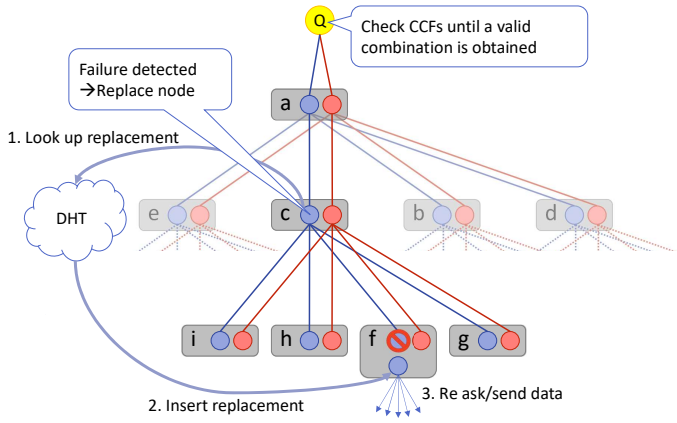


Fig. 3: *HighCpl* strategy

Objective

To have a complete view of the design space, we want to design a strategy that eagerly seeks to maximize completeness regardless of cost. We call this strategy

HighCpl. Despite its more pessimistic outlook on node dropouts compared to *LowCost*, *HighCpl* is designed to be optimistic about its ability to handle them, hence the unlimited efforts to detect and manage dropouts.

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, as illustrated in Fig. 3. 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 (a new version after a leaf aggregator replacement would anyway require a new synchronization). 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 maximize 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 but in favorable cases, it should lead to reduced latency and maximized completeness.

6.3 *Sync&Prune*

Objective

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

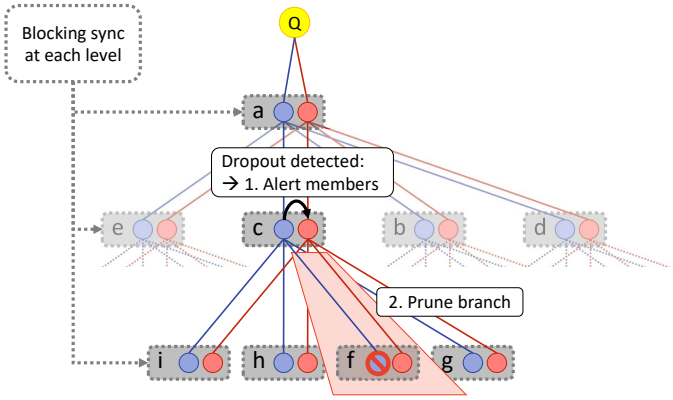


Fig. 4: *Sync&Prune* strategy

Design principles

Sync&Prune is based on the *LowCost* strategy. It 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* and pruning the branches containing inconsistent shares or dropouts, as shown in Fig. 4.

Protocol

- **Node replacement policy.** As for *LowCost*, given the 'send-only-once' principle, nodes are only replaced as long as their children never sent data.
- **Validity.** Non-blocking sync is not compatible with the 'send-only-once' principle because if nodes are to send data only once, they should take a bit of time to make sure they are sending valid data. Thus *Sync&Prune* employs *blocking sync* to ensure validity and does so at every level of the tree. Indeed, since dropout aggregators are 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 and branches producing invalid results.
- **Termination.** *Sync&Prune* terminates when the querier receives s shares from its child group or if it detects a dropout in the root group. Like in *LowCost*, lower-level nodes progressively terminate, from leaves to the root, after sending their *IntRes* to their parents. Whenever a branch of the tree is pruned, nodes of that branch are contacted using the remaining tree structure to inform them that they can terminate.

Discussion

Sync&Prune is expected to have a low cost due to the 'send-only-once' strategy. Syncing also adds an overhead, especially with the blocking variant, but we expect it to be low compared with the data transmission and the message decryption/encryption costs. The blocking sync is also expected to increase latency in comparison with *LowCost*. However, sync should fix the 0% completeness issue of *LowCost* by syncing and pruning sub-trees in case of dropouts, rather than aborting the whole execution.

6.4 *Hybrid*

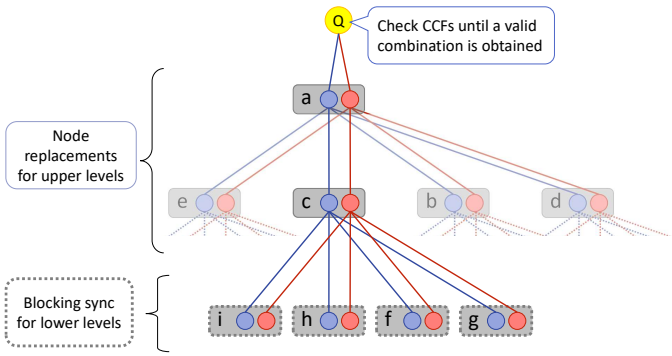


Fig. 5: *Hybrid* strategy

Objective

Hybrid is the second trade-off strategy attempting to maximize completeness and maintain a reasonable cost: by relaxing *HighCpl*'s dropout intolerance, *Hybrid* should avoid overpaying to replace nodes.

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 a 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 requires their children to re-send their *IntRes*. Note that when leaf aggregators drop out before they send their intermediate results, because of the 'send-only-once' policy, will prevent replacements from reproducing the intermediate results. In this case, the adopted behavior is to fall back on the *Sync&Prune* strategy of telling parent to prune the part of the tree containing the leaf group, as illustrated in Fig. 5.
- **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 if they drop out after receiving data from contributors, the sync at their parent's level induces pruning as in *Sync&Prune* , which negatively impacts completeness. However, the aggregator replacements in the upper levels avoid pruning a large number of contributors and thus favor 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. 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.

Although the protocol of this strategy applies the *Sync&Prune* for leaves and contributors and *HighCpl* for upper aggregators, the depth threshold between those strategies could be adjusted. It is done automatically and at the branch level by groups being pruned out of the tree, but it can also be enforced at the protocol level by having contributors and any levels of aggregators above follow the *Sync&Prune* strategy, with extreme values of this threshold devolving either into *Sync&Prune* when the threshold in at a minimum, or into *HighCpl* when the threshold is at its maximum. However, the replacements have the biggest impact on performances for leaf aggregators, because that is where there is the most nodes to replace and where the most data is exchanged. Therefore, setting the threshold just above the leaf aggregators yields the best result, with higher threshold having only marginal benefits while slightly decreasing performances.

As mentioned for the node replacement policy, leaf aggregator groups can be pruned when some members send data while others are being replaced. This

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

Table 1: Design and building blocks

could lead to extreme cases where the parent group of a leaf group has almost no children left alive. A new security issue arises when only one child group is left: an attacker learns more than expected due to its nodes being present at adjacent depths of the tree and different group positions but observing the same data. This can be mitigated by limiting the number of replacements a parent group can do before being pruned.

6.5 Summary

Table 1 summarizes the design and the behavior of each strategy. By eagerly replacing nodes, *HighCpl* favors completeness at 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 sub-trees. Finally *Hybrid* leverages the best of both *Sync&Prune* and *HighCpl*.

7 Performance Evaluation

In this section, we present the evaluation platform and used metrics, then describe the experimental parameters and the system security configuration. We present and analyze the experimental results varying the dropout rate and the other parameters, and then conclude with an analysis of the best-fitted strategy depending on the context.

7.1 Experimental Platform

Our main goal is to evaluate the four proposed strategies in a large P2P PDMS system wherein the nodes are structured leveraging a Chord DHT overlay [64]. However, peer-to-peer networks are notoriously hard to study: they rely on many participants across various geographical regions, using different physical networks and a wide range of hardware [23, 47, 7]. The most notable problems experiments on peer-to-peer networks are facing are:

– **Reproducibility**. The large number of moving pieces that compose a peer-to-peer network makes experimenting hard to reproduce. Good simulators enable reproducible results using deterministic randomness. Moreover, they should offer a way to perform fair ablation studies by keeping all things equal when only varying a parameter.

- **Scalability**. The study of peer-to-peer networks often relies on having thousands of nodes connected. Each node may need to independently simulate some local properties on top of doing any computations required by the protocol or application built on the network. Managing all these operations can be challenging at scale.
- **Application-specificity**. Having a detailed simulation of every network mechanism is not of equal importance depending on what is being tested: while it makes sense for new or modified versions of existing components, it may not always be true when testing higher-level applications built on peer-to-peer networks. Working at the appropriate level can greatly enhance scalability and reproducibility.

Existing Simulators

Intending to test the strategies we proposed in Section 6, we follow the same general approach as in the related work on P2P systems [64, 53, 74, 33], i.e., our results are based on a simulator[43] which creates a logical DHT between simulated nodes. Indeed, simulators were used to evaluate the performance of the state-of-the-art structured DHTs (such as Chord [64] and CAN [53]) and systems that leverage P2P DHTs, such as in the distributed information retrieval area [54, 67, 66, 74, 33].

State-of-the-art peer-to-peer network simulators [47] focus either on scalability or fidelity. P2PSim [29] provides discrete-events simulation in a peer-to-peer network while offering several options to model the underlying physical network connecting peers. However, the amount of events that this application generates makes it hardly usable above a few thousand nodes. In PeerSim [46], up to hundreds of thousands of nodes can be simulated, however with a much lower fidelity since it only simulates query-cycles (from the moment a node queries some data in the network to the moment it receives it). In both cases, building more complex protocols on top of that simulator is ill-advised, because their focus is too specific to peer-to-peer communications rather than peer-to-peer and decentralized applications.

With the recent popularity of federated learning, the amount of tooling available to monitor and experiment on those systems has grown. The goal of those simulators is generally to experiment with new components of some FL workflow or to deploy them to production. FLINT [70] offers a platform that can provide many useful analytics and tools for users trying to transition from a centralized ML setting to a federated learning setting, as well as a decision framework to understand how worthy it can be to switch. However, it is focused on the cloud-based FL, not suitable for peer-to-peer usage, and favors efficiency over fairness by grouping devices by capabilities to accelerate training. More research-oriented platforms like Flower [10] on the contrary, focus on large-scale networks of heterogeneous devices, with up to 15 million devices simulated using only a pair of high-end GPUs. However, the dropouts model it considers resembles that of *SecAgg* [12], which is incompatible with our setting, as explained in Section 2.4.

Neither of the highly detailed or FL-focused simulators can however simultaneously address the complex dropout model that we are targeting in our strategies, the specific peer-to-peer communication overlay, and the application to large-scale FL that we require. This acts as our motivation to develop our simulator, tailored to our needs.

Simulator Design

The first step of designing a simulator is to define the scope. If it is too wide, the simulation gains in fidelity negatively impact scalability, and conversely. Our goal is to make the simulator as reusable as possible, to enable others to test and experiment with peer-to-peer federated learning on networks of PDMS. For this reason, we reduce the scope of the simulator to the core phase of our protocol, namely the aggregation phase. This allows experimenters using different components (such as constructing the tree overlay or disseminating models to contributors) to still benefit from the insights provided by our simulator while using their custom logic. This follows the approach used by framework like Flower [10] to focus on the aggregation while being compatible with many methods to organize the rest of the computations.

Given this scope, we can see that modeling communications at a low level is not necessary: once the tree overlay is fixed to transmit data, DHT routing is only used to find replacement nodes. Because of this limited impact, a probabilistic approach to find the number of hops needed to route messages between nodes can save a lot of computing during the simulation.

To make the simulation easy to analyze, we used an event-driven simulator: every action in the protocol is modeled as a message that is chronologically inserted in a queue. For variable duration interactions, a first message is used to indicate the start of the interaction and a second message that can be postponed indefinitely while the node is busy indicates the end. The queue can be stored on the hard drive for later analysis.

The simulator is designed with modularity in mind, as we need to test multiple building blocks and plan on having others experiment with new blocks. To this end, the simulator is conceived to be easily parameterized, and adding new components is straightforward. Building blocks can then be mixed and matched according to the experimental setting.

Finally, many statistics are collected at every step of the simulation. These statistics can either be aggregated when analyzing many executions of the protocol, or they can be maximally detailed when needed. The type of statistics collected can be adjusted for each experiment, to avoid exporting too much data if this is not needed.

7.2 Evaluation Metrics

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 distributed protocols. At 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. While estimating bandwidth is simple by monitoring the exchanged messages and the size of their content, estimating absolute time values (i.e., in seconds) for work and latency is more challenging since such measures are highly dependent on the context (underlying network topology, PDMS node heterogeneity, network congestion, etc.). For the network, we consider network links with latency and bandwidth based on average values of domestic Internet in France [49] 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. We note that in practice, no participant can have the exact value of completeness for a given result: although the number of contributions in the final aggregate is computed, the number of nodes that intended to contribute can never be known since some of them may drop out before telling anyone about their intentions.

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).

7.3 Experimental and Security Parameters

Besides the simulator calibration described above and summarized in purple in Table 2, the other experimental parameters are divided into four classes as follows.

- **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 used the revised version of the formula presented in Section 5.2 and allowed only one replacement. 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. All these values are quite high since it must be the same attacker that controls all these nodes (see also [13]).
- **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 allow 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 node dropouts 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 [45] 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, [12, 62] consider the size of 1MB.
- **Number of runs and box plots.** Given the randomness of simulated dropouts, and despite the pre-computation of dropouts (see above), there is a lot of variability between simulation runs. Indeed, a single dropout can have dramatically different outcomes depending on where in the tree and when it occurs. We aggregate the results of 50 runs to obtain statistically representative results. In addition, we use box plots which help visualize 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 the 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 in the interquartile range) which are directly represented as points. Finally, the line connecting the mean values is also represented.

7.4 Performance with Varying Dropout Rate

Now that the experimental setup has been described, we present the performance of our strategies under specific scenarios, varying parameters one by one. We start by varying the dropout rate, as it is the most important parameter to understand how dropouts affect the system.

Fig. 6, 7, 8 and 9 depict respectively the completeness, latency, bandwidth and work by node, varying the dropout rate D for the four studied strategies. All strategies, when using the same parameters, perform almost identically when there are no dropouts since the overhead of each strategy (e.g., blocking sync) is mostly negligible compared to the transmission cost of the 1MB model through the tree. This setting provides a nice baseline to which we can then compare strategies in different settings.

Description (*notation*)	**Values (*default*)**
Network latency [49]	30ms
Network bandwidth [49]	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

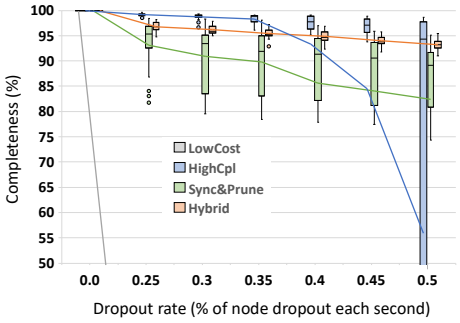


Fig. 6: Completeness, 1 MB model

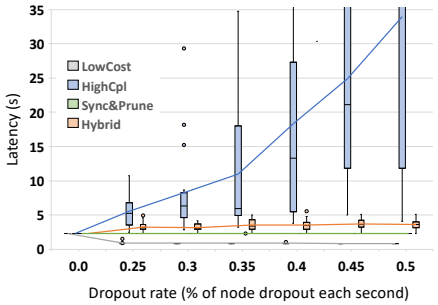


Fig. 7: Latency, 1 MB model

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 can contribute data. Fig. 8 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 generally prevent it from succeeding. Specifically, under default parameters and few dropouts (not shown in the figures), we obtained 45% of *LowCost* successful executions with $D = 0.01\%$, only 10% when $D = 0.02$ and 0% with higher dropout rates.

HighCpl has the best completeness but 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. 7 and 8). *HighCpl* gives up replacing nodes after

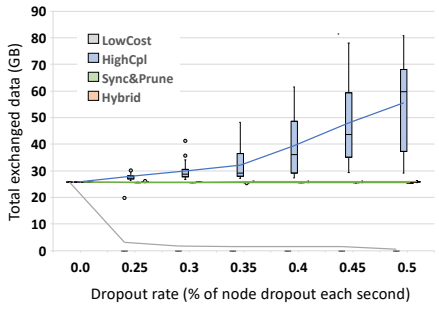


Fig. 8: Bandwidth, 1 MB model

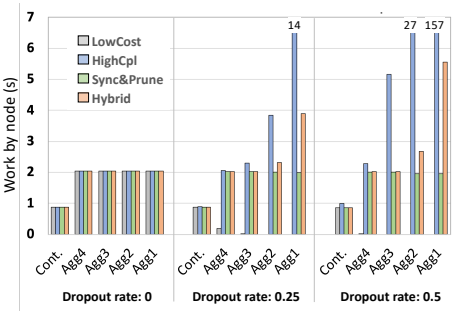


Fig. 9: Work by node, 1 MB model

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: during the time needed to replace some nodes, even more nodes drop out, preventing the querier from obtaining valid aggregates. This makes it unsuitable for large dropout rates.

Sync&Prune has stable costs but lower completeness

Opposite to *HighCpl*, *Sync&Prune* starts out with a lower completeness and 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. However, *Sync&Prune* has a more stable performance on all metrics as the dropout rate increases, because pruning trees is cheaper than resending data that may become invalid: it puts an upper bound on costs at the expense of initially lower completeness. 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.

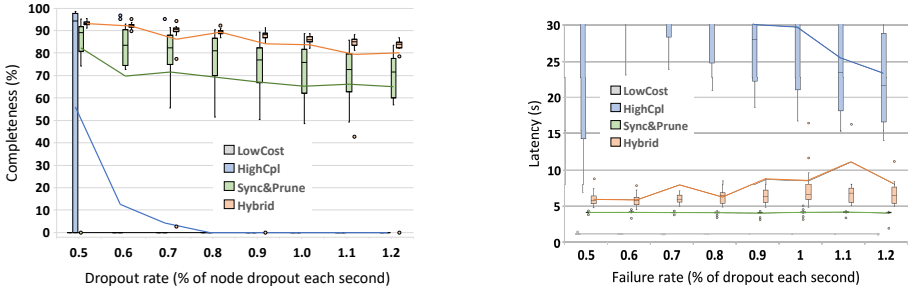
Hybrid combines the best of the two previous strategies

Hybrid has 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 and instead, leaf groups and their contributors are pruned. It is however a lot better for high dropout rates because leaf aggregators are never replaced, preventing the conflicting CCFs that cause 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, inducing an increased latency, but this has a limited impact due to blocking **sync**.

HighCpl suffers from too many aggregate versions

Fig. 9 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 $k = 5$ 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 the largest 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.

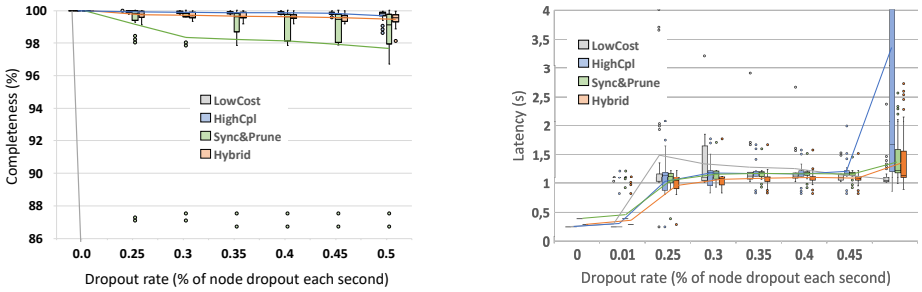


(a) Completeness (b) Latency

Fig. 10: Performances under extreme dropout rates

Hybrid and *Sync&Prune* tolerate extreme dropout rates

Fig. 10a 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. However, costs for *Hybrid* grow faster than those of *Sync&Prune*, as shown in Fig. 10b. 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. Reaching this limit also explains why *HighCpl*, which replaces more nodes, aborts the execution sooner as the dropout rate increases.



(a) Completeness (b) Latency

Fig. 11: Performances for 1KB model

***HighCpl* and *Hybrid* work well with small models**

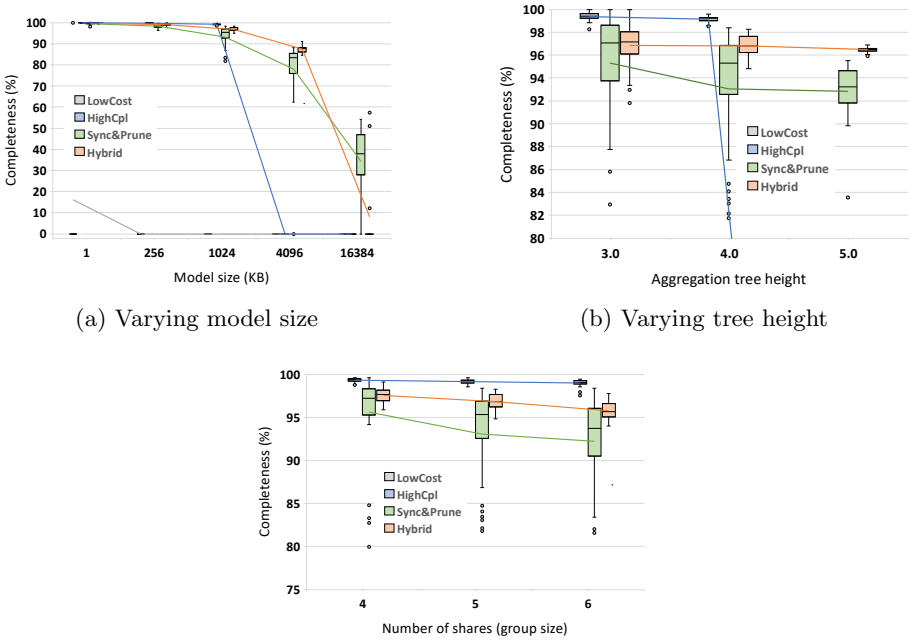
Fig. 11a and Fig. 11b show the obtained completeness and latency with a small model of only 1KB. Small models change the protocol bottlenecks since the model transmission becomes less impacting. Moreover, since the overall latency is much smaller, there are fewer dropouts and thus better completeness for all strategies. *HighCpl* and *Hybrid* obtain almost 100% completeness, much better than *Sync&Prune*, with low overhead. Fig. 11a 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 are generally the reason for the higher variability of completeness in the executions of *Sync&Prune*. We can finally see in Fig. 11b that latency sharply increases as soon as dropouts start start to happen, even for the smallest dropout rates where we can see a pack of outlier at around 1 second latency. This occurs because dropouts are detected after a timeout, whose duration is significant compared to the overall latency: active nodes will consider a peer as dropped out if it did not respond to the most recent health check in up to 10 times the average duration of a health check, which in this case should take $\approx$ 60ms. This leads to a 600 ms delay, which can be observed by the difference of latency between runs without dropouts and runs with higher dropout rates.

7.5 Scalability and Security

Now that we have extensively studied the robustness of our strategies, we want to focus on the scalability and security aspects, as they are key issues for decentralized federated learning.

Very large models require less dropouts

For a model size of 16 MB, none of the strategies reaches completeness above 50% as shown in Fig. 12a. We can see that *Sync&Prune* starts outperforming *Hybrid*,



(a) Varying model size

(b) Varying tree height

(c) Varying security

Fig. 12: Completeness for different parameters variations

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 higher 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 the 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. We use the tree height as a proxy for the number of contributors because it has more impact on performances: the selectivity in a given network can vary almost without affecting performances until the tree is saturated and the depth increases, where we observe sharp changes in metrics. Using the tree height prevents gaps in the plots due to the effect of rounding the depth and reduces the complexity of the

analysis. *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 an increasing number of contributors showing that they can fully benefit from the distributed nature of the execution. The seemingly increased stability comes from the fact that some runs being further away from the performances of most runs, which labels them as outliers and exclude them from the box and puts a point instead. The fact that it barely affects boxes demonstrates the small impact of large trees on completeness, where failures happen most where they have the least impact: on leaves.

Larger groups (better security) are well supported

Finally, we study the impact of the security parameter on all strategies. We can see in Fig. 12c 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.

7.6 Best Fit Strategy

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

(a) At max 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

(b) At 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

(c) At 95% completeness

Fig. 13: Least costly strategy for various completeness threshold

In this last section, we first choose some typical parameter settings that best exemplify realistic scenarios, then define different objectives that guide us

in comparing the strategies. Finally, we report in Fig. 13a, 13b and 13c 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 of the context of federated learning (i.e., it is pertinent to aggregate from fewer 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 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. 13a, 13b and 13c 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. 13a, *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 a few settings on Fig. 13c.

***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.13c), 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 a 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 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 settings, trained model size, or contributors' selectivity. Our results showed that these protocols can offer high completeness results at a reasonable cost in a wide range of settings.

We plan to extend this work in several directions and shortly discuss two of them. First, our current aggregation protocols consider a honest-but-curious threat model. Although this model is widely used in the secure aggregation context, it would be useful to consider more advanced attack models, wherein the attacker can completely control some system nodes. However, this has profound implications such as attacks on the DHT, on the tree construction mechanism and on the assumed uniformly random selection of nodes to create groups as well as the node replacement mechanism. We believe that the techniques proposed in [13], namely collaborative probabilistic proofs could probably be applied even if node dropouts make the problem more challenging. Second, the aggregation process can be prone to data poisoning and backdoor attacks [6, 71]. The typical solution based on filtering malicious contributions cannot work with secure aggregation since contributors' inputs are private. Hence, rightful inputs cannot be distinguished from purposefully crafted data intended to corrupt a model training. Existing work [17] provides privacy-preserving ways to ensure data fits a criterion but using a centralized architecture. By supposing that PDMS data is signed by the entity that inserts it, we envision including these signatures into our CCF mechanism and aggregating them to allow participants to verify that the aggregate does not contain maliciously crafted data.

Acknowledgments

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

References

1. Agarwal, N., Suresh, A.T., Yu, F.X., Kumar, S., *et al.*: cpSGD: Communication-Efficient and Differentially-Private Distributed SGD. In: NeurIPS (2018)
2. Alvim, M.S., Chatzikokolakis, K., Palamidessi, C., Pazii, A.: Local Differential Privacy on Metric Spaces: Optimizing the Trade-Off with Utility. In: IEEE CSF (2018)

3. Anciaux, N., Bonnet, P., Bouganim, L., Nguyen, B., *et al.*: Personal Data Management Systems: The Security and Functionality Standpoint. Information Systems (2019)
4. Anciaux, N., Bouganim, L., Pucheral, P., Sandu-Popa, I., *et al.*: Personal Database Security and Trusted Execution Environments: A Tutorial at the Crossroads. PVLDB (2019)
5. ARM, Building a Secure System using TrustZone Technology, (2008). `https://perma.cc/WE2E-WV2A`
6. Bagdasaryan, E., Veit, A., Hua, Y., Estrin, D., *et al.*: How to backdoor federated learning. In: International conference on artificial intelligence and statistics (2020)
7. Basu, A., Fleming, S., Stanier, J., Naicken, S., *et al.*: The state of peer-to-peer network simulators. ACM Computing Surveys (CSUR) (2013)
8. Bater, J., Elliott, G., Eggen, C., Goel, S., *et al.*: SMCQL: Secure Query Processing for Private Data Networks. PVLDB (2017)
9. Bellet, A., Guerraoui, R., Taziki, M., Tommasi, M.: Personalized and Private Peer-to-Peer Machine Learning. In: AIStat (2018)
10. Beutel, D.J., Topal, T., Mathur, A., Qiu, X., *et al.*: Flower: A friendly federated learning research framework. arXiv preprint arXiv:2007.14390 (2020)
11. Bittau, A., Erlingsson, Ú., Maniatis, P., Mironov, I., *et al.*: Prochlo: Strong privacy for analytics in the crowd. In: Proceedings of the 26th symposium on operating systems principles (2017)
12. Bonawitz, K., Ivanov, V., Kreuter, B., Marcedone, A., *et al.*: Practical Secure Aggregation for Privacy-Preserving Machine Learning. In: ACM CCS (2017)
13. Bouganim, L., Loudet, J., Sandu Popa, I.: Highly Distributed and Privacy-Preserving Queries on Personal Data Management Systems. The VLDB Journal (2023)
14. Brasser, F., Müller, U., Dmitrienko, A., Kostiainen, K., *et al.*: Software Grand Exposure: SGX Cache Attacks Are Practical. In: 11th USENIX Workshop on Offensive Technologies (2017)
15. Bulck, J.V., Minkin, M., Weisse, O., Genkin, D., *et al.*: Foreshadow: Extracting the Keys to the Intel SGX Kingdom with Transient Out-of-Order Execution. In: 27th USENIX Security Symposium (2018)
16. Cormode, G., Kulkarni, T., Srivastava, D.: Answering Range Queries Under Local Differential Privacy. PVLDB (2019)
17. Corrigan-Gibbs, H., Boneh, D.: Prio: Private, Robust, and Scalable Computation of Aggregate Statistics. In: NSDI (2017)
18. Costan, V., Devadas, S.: Intel SGX Explained, (2016). Cryptology ePrint Archive.
19. Cozy Cloud, Cozy Cloud (See `https://cozy.io/fr/`), (2023)
20. Ding, B., Kulkarni, J., Yekhanin, S.: Collecting Telemetry Data Privately. In: Advances in Neural Information Processing Systems (2017)
21. Dong, Y., Chen, X., Li, K., Wang, D., *et al.*: FLOD: Oblivious Defender for Private Byzantine-Robust Federated Learning with Dishonest-Majority. In: ESORICS (2021)
22. Dwork, C.: Differential Privacy. In: Automata, Languages and Programming (2006)
23. Ebrahim, M., Khan, S., Hasan Mohani, S.S.U.: Peer-to-Peer Network Simulators: an Analytical Review. Asian Journal of Engineering, Sciences & Technology (2012)
24. Erlingsson, Ú., Pihur, V., Korolova, A.: RAPPOR: Randomized Aggregatable Privacy-Preserving Ordinal Response. In: Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security. CCS '14 (2014)
25. Eskandarian, S., Zaharia, M.: ObliDB: Oblivious Query Processing for Secure Databases. Proc. VLDB Endow. (2019)

26. European Commission, Proposal for a Regulation on European data governance (Data Governance Act). [eur-lex], (2020).
27. Fereidooni, H., Marchal, S., Miettinen, M., Mirhoseini, A., *et al.*: SAFELearn: Secure Aggregation for Private Federated Learning. In: IEEE SPW (2021)
28. Ghazi, B., Manurangsi, P., Pagh, R., Velingker, A.: Private aggregation from fewer anonymous messages. In: EUROCRYPT (2020)
29. Gil, T., Kaashoek, F., Li, J., Morris, R., Stribling, J.: P2PSim, a simulator for peer-to-peer protocols, (2003).
30. Gueron, S.: Memory Encryption for General-Purpose Processors. IEEE Security Privacy **14** (2016)
31. Guo, X., Liu, Z., Li, J., Gao, J., *et al.*: VeriFL: Communication-Efficient and Fast Verifiable Aggregation for Federated Learning. IEEE Trans. Inf. Forensics Secur. (2021)
32. Gupta, P., Li, Y., Mehrotra, S., Panwar, N., *et al.*: Obscure: Information-Theoretic Oblivious and Verifiable Aggregation Queries. PVLDB (2019)
33. Hayek, R., Raschia, G., Valduriez, P., Mouaddib, N.: Summary management in P2P systems. In: EDBT (2008)
34. Huba, D., Nguyen, J., Malik, K., Zhu, R., *et al.*: Papaya: Practical, private, and scalable federated learning. Proceedings of Machine Learning and Systems (2022)
35. Kasiviswanathan, S.P., Lee, H.K., Nissim, K., Raskhodnikova, S., *et al.*: What Can We Learn Privately? SIAM Journal on Computing (2011)
36. Li, N., Li, T., Venkatasubramanian, S.: t Closeness: Privacy Beyond k-Anonymity and l-Diversity. In: ICDE (2007)
37. Loudet, J., Sandu-Popa, I., Bouganim, L.: DISPERS: Securing Highly Distributed Queries on Personal Data Management Systems. PVLDB (2019)
38. Loudet, J., Sandu-Popa, I., Bouganim, L.: SEP2P: Secure and Efficient P2P Personal Data Processing. In: EDBT (2019)
39. Machanavajjhala, A., Kifer, D., Gehrke, J., Venkitasubramaniam, M.: L-Diversity: Privacy beyond k-Anonymity. ACM Transactions on Knowledge Discovery from Data (2007)
40. Mansouri, M., Önen, M., Jaballah, W.B., Conti, M.: SoK: Secure Aggregation Based on Cryptographic Schemes for Federated Learning. PETS (2023)
41. McMahan, B., Moore, E., Ramage, D., Hampson, S., *et al.*: Communication-Efficient Learning of Deep Networks from Decentralized Data. In: PMLR (2017)
42. Merkle, R.C.: A Digital Signature Based on a Conventional Encryption Function. In: CRYPTO (1987)
43. Mirval, J.: FLP2P Peer-to-peer Simulator, `https://github.com/cozy/dissec_cozy/tree/master/simulation`.
44. Mirval, J., Bouganim, L., Sandu Popa, I.: Federated learning on personal data management systems: Decentralized and reliable secure aggregation protocols. In: SSDBM (2023)
45. Mirval, J., Bouganim, L., Sandu-Popa, I.: Practical Fully-Decentralized Secure Aggregation for Personal Data Management Systems. In: SSDBM (2021)
46. Montresor, A., Jelasity, M.: PeerSim: A scalable P2P simulator. In: P2P (2009)
47. Naicken, S., Livingston, B., Basu, A., Rodhetbhai, S., *et al.*: The state of peer-to-peer simulators and simulations. ACM SIGCOMM (2007)
48. Nguyen, J., Malik, K., Zhan, H., Yousefpour, A., *et al.*: Federated learning with buffered asynchronous aggregation. In: AISTATS (2022)
49. nPerf, Baromètre des connexions Internet fixes en France métropolitaine, (2020). `https://perma.cc/DP8V-5ABT`

50. Oleksenko, O., Trach, B., Krahn, R., Silberstein, M., *et al.*: Varys: Protecting SGX Enclaves from Practical Side-Channel Attacks. In: USENIX ATC, Boston, MA (2018)
51. Pilet, A.B., Frey, D., Taıani, F.: Robust Privacy-Preserving Gossip Averaging. In: SSS (2019)
52. Przydatek, B., Song, D., Perrig, A.: SIA: Secure Information Aggregation in Sensor Networks. In: SenSys (2003)
53. Ratnasamy, S., Francis, P., Handley, M., Karp, R.M., *et al.*: A Scalable Content-Addressable Network. In: ACM SIGCOMM (2001)
54. Reynolds, P., Vahdat, A.: Efficient Peer-to-Peer Keyword Searching. In: Middleware (2003)
55. Rivest, R.L., Adleman, L., Dertouzos, M.L.: On data banks and privacy homomorphisms. Foundations of secure computation (1978)
56. Sabater, C., Bellet, A., Ramon, J.: An Accurate, Scalable and Verifiable Protocol for Federated Differentially Private Averaging. JMLR (2022)
57. Sabt, M., Achemlal, M., Bouabdallah, A.: Trusted Execution Environment: What It is, and What It is Not. In: Trustcom (2015)
58. Sandu-Popa, I., Ton-That, D.H., Zeitouni, K., Borcea, C.: Mobile Participatory Sensing with Strong Privacy Guarantees using Secure Probes. GeoInformatica (2021)
59. van Schaik, S., Kwong, A., Genkin, D., Yarom, Y.: SGAxe: How SGX Fails in Practice, (2020). `https://sgaxeattack.com/`
60. van Schaik, S., Minkin, M., Kwong, A., Genkin, D., *et al.*: CacheOut: Leaking Data on Intel CPUs via Cache Evictions. In: SP (2021)
61. Schwarz, M., Weiser, S., Gruss, D., Maurice, C., *et al.*: Malware Guard Extension: Using SGX to Conceal Cache Attacks. In: DIMVA (2017)
62. So, J., Güler, B., Avestimehr, A.S.: Turbo-Aggregate: Breaking the Quadratic Aggregation Barrier in Secure Federated Learning. JSAIT (2021)
63. So, J., He, C., Yang, C.-S., Li, S., *et al.*: Lightsecagg: A Lightweight and Versatile Design for Secure Aggregation in Federated Learning. MLSys (2022)
64. Stoica, I., Morris, R., Karger, D., Kaashoek, M.F., *et al.*: Chord: A Scalable Peer-to-Peer Lookup Service for Internet Applications. ACM SIGCOMM (2001)
65. Sweeney, L.: k-anonymity: A model for protecting privacy. Int. J. Uncertain. Fuzziness Knowl.-Based Syst. (2002)
66. Tang, C., Dwarkadas, S.: Hybrid Global-Local Indexing for Efficient Peer-to-Peer Information Retrieval. In: NSDI (2004)
67. Tang, C., Xu, Z., Dwarkadas, S.: Peer-to-Peer Information Retrieval using Self-Organizing Semantic Overlay Networks. In: ACM SIGCOMM (2003)
68. To, Q., Nguyen, B., Pucheral, P.: Private and Scalable Execution of SQL Aggregates on a Secure Decentralized Architecture. ACM TODS (2016)
69. Triastcyn, A., Faltings, B.: Federated Learning with Bayesian Differential Privacy. In: IEEE BigData (2019)
70. Wang, E., Chen, B., Chowdhury, M., Kannan, A., Liang, F.: FLINT: A Platform for Federated Learning Integration. Proceedings of Machine Learning and Systems (2023)
71. Wang, H., Sreenivasan, K., Rajput, S., Vishwakarma, H., *et al.*: Attack of the tails: Yes, you really can backdoor federated learning. NIPS (2020)
72. Warner, S.L.: Randomized Response: A Survey Technique for Eliminating Evasive Answer Bias. JASA (1965)

73. Yang, G., Wang, S., Wang, H.: Federated Learning with Personalized Local Differential Privacy. In: IEEE ICCCS (2021)
74. Yang, Y., Dunlap, R., Rexroad, M., Cooper, B.F.: Performance of Full Text Search in Structured and Unstructured Peer-to-Peer Systems. In: INFOCOM (2006)
75. Zhang, C., Li, S., Xia, J., Wang, W., *et al.*: {BatchCrypt}: Efficient homomorphic encryption for {Cross-Silo} federated learning. In: USENIX ATC (2020)
76. Zhang, Z., Wang, T., Li, N., He, S., *et al.*: CALM: Consistent Adaptive Local Marginal for Marginal Release under Local Differential Privacy. In: ACM CCS (2018)