Edgelet Computing: Enabling Privacy-Preserving Decentralized Data Processing at the Network Edge

Ludovic Javet[1,2], Nicolas Anciaux[1,2], Luc Bouganim[1,2], Philippe Pucheral[1,2]
[(1)]Petrus team, Inria, France, [(2)]David Lab., U. Versailles St-Quentin-en-Yvelines, France
firstname.lastname@inria.fr

Abstract

The convergence of Opportunistic Networks and Trusted Execution Environments at the network edge presents a compelling opportunity for fully decentralized privacy-preserving data processing. Based on this convergence, we define the concept of Edgelet computing, a new paradigm for executing powerful and privacy-preserving distributed queries on personal devices. Our objective is to establish a robust, secure, and scalable execution framework with strong individual privacy guarantees. This paper first proposes a liability model tailored to decentralized executions on crowd members' devices, along with a query evaluation model that differs from the traditional database closed-world assumption. Second, it defines essential properties for ensuring the security, resiliency and validity of executions, and subsequently presents several methods and strategies for their enforcement. Through a comprehensive qualitative analysis and extensive evaluations, we showcase the relevance and effectiveness of the approach, demonstrating that Edgelet Computing holds potential for the emergence of novel and important classes of applications.

Keywords Crowd computing · Trusted Execution Environment · Opportunistic Network

1 Introduction

The edge of the Internet is expanding at a much higher pace than its core, with 29.3 billion connected devices by 2023 [17]. At the same time, the Internet reliable, server-based, and infrastructure-centric approach already exhibits its limits in terms of efficiency, privacy, and energy consumption. Alternatively, infrastructure-less and self-organizing networking protocols have emerged, from the legacy MANET protocol to the Opportunistic Network (OppNet) paradigm, from which a new people-centric vision of the Internet (IoP) can emerge [19]. However, while this paradigm has attracted a lot of attention from the research community, OppNets have never been deployed at a large scale, mainly due to the lack of killer applications [67].

A game changer is the generalization of Trusted Execution Environments (TEE) [57] at the extreme edge of the network: Intel SGX [22] is becoming ubiquitous on PCs and tablets, ARM's TrustZone [54] on smartphones (Figure 1.a) and even Trusted Platform Module on smart objects (Figure 1.b). TEEs protect code and data from untrusted execution environments and from the devices' owners. They are new building blocks for the organization of fully decentralized and secure computations among data scattered on multiple personal devices, without resorting to any central authority or infrastructure. Hence, powerful large-scale privacy-preserving computations are within reach in a different – and more flexible – manner than with homomorphic cryptography [14], local differential privacy [40], or secure multiparty computation protocols [7]. Our approach leverages the security features of TEEs to enable generic computations and scalable executions, processing cleartext data without any tradeoff between privacy and accuracy.

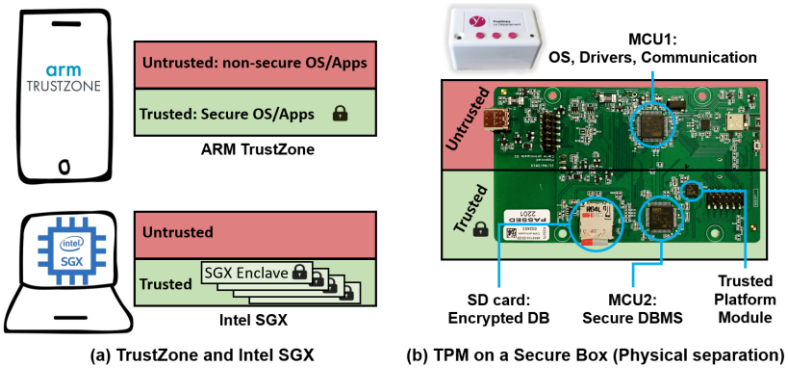

Figure 1: Examples of Trusted Execution Environments

The convergence between OppNets and TEEs, named hereafter *Edgelet computing*, leverages secure personal devices (called edgelets) and promises concrete applications for OppNets involving crowd participation, for example:

Opportunistic polling. During events that attract a large audience (e.g., conferences, concerts, museums, sporting events), the participants could contribute with their data (e.g., centers of interest, nationality, age) to global processing to improve their user experience in real-time (i.e., adapting the services to the characteristics of the audience). The proximity of individuals and their personal devices makes the use of traditional communication infrastructures unnecessary, if not inappropriate, and paves the way for Opportunistic Computing [18]. For example, estimating the number of people present at a demonstration is a difficult task, both for the police and for the demonstrators themselves. The deployment of an Edgelet application could enable opportunistic counting of individuals, providing accurate statistics (e.g., based on age or political conviction) without compromising their privacy.

Data altruism. Introduced in the EU Data Governance Act [55], this proposal encourages data subjects to consent to process their personal data for purposes such as scientific research or public services improvement (e.g., a health survey organized by Santé Publique France). Privacy protection is paramount in this context, as it is a key element for people to participate with their sensitive data. For instance, the DomYcile Project [24] is a real-case study that proposes a secure personal box for managing the medical records of elderly individuals receiving home assistance. DomYcile allows patient associations and health or statistical agencies to query ephemeral cohorts of several thousands of these boxes, each acting as an edgelet using short-range communications. Such a scenario, detailed in subsection 3.2 , could be generalized to all forms of Personal Data Management Systems [5] hosted by their holder.

These scenarios require performing complex processing over personal data, ranging from regular database aggregation queries to machine learning computations, in a highly distributed, failure-prone, and infrastructure-less context, with strong security guarantees. The specificity of this new Edgelet computing paradigm requires the definition of (1) a new liability model adapted to decentralized execution on sets of crowd members devices, which translates the shift of responsibility from cloud providers to crowd members compared to traditional cloud-based liability models and (2) a new query evaluation model where the traditional database closed-world assumption no longer applies: on the contrary, queries must be evaluated on database subsets built opportunistically, drastically changing the query evaluation process. The impact of such liability and query models on security, resiliency and validity is somehow antinomic and thus requires the definition of unusual appropriate solutions. Our objective is then threefold: we aim to build a framework that (1) is both generic and scalable, allowing complex computations on the data of thousands of individuals, (2) ensures security and respects the privacy of the people involved, and (3) is fault-tolerant in a fully decentralized environment, which is prone to failures and message loss.

The existing approaches relying on secure multiparty computation, local differential privacy or secure outsourced computations often struggle to strike an optimal balance between privacy preservation and computation accuracy. Secure multiparty computation [7], although ensuring collaborative privacy, faces difficult challenges in terms of scalability and computational efficiency, particularly when dealing with large, diverse and dynamic sets of participants. Local differential privacy [40], while protecting individual data values, may compromise the utility and accuracy of aggregate computations due to noise injection, and would not provide genericity (in the sense of supporting any type of computation). Secure computations outsourcing [2] would introduce, on the other hand, potential vulnerabilities within centralized authorities or third-party intermediaries which would not be acceptable in our context. In light of these limitations, the Edgelet computing paradigm aims to overcome these challenges by harnessing the capabilities of Trusted Execution Environments (TEE) to enable decentralized, secure, generic, and efficient computations that transcend the inherent trade-offs of existing solutions.

To achieve the objectives listed above and to match the field realities of our crowd contexts, we make the following contributions[1].

- First, we characterize the Edgelet computing paradigm: a new framework for implementing powerful processing on personal devices in a highly distributed, failure-prone, and infrastructure-less environment. We specify the key features of Edgelet computing, introduce a novel liability model associated with these features, and precisely state the problem addressed through three properties, namely Security, Resiliency and Validity.

[1] This paper builds on a CCGRID conference paper [37] with important additions, namely: (1) the Edgelet computing paradigm is more precisely characterized, leading to the definition of a new dedicated liability model (section 2.2) and a snapshot query model along with a strawman execution strategy (section 2.3); (2) new security mechanisms required to enforce the purpose honesty (section 3.2) and computation honesty (section 3.3) related to this liability model; (3) a new resiliency strategy (section 4.3) and its evaluation; (4) new evaluations comparing the proposed strategies in terms of exposure (section 6.1) and (5) the consideration of Stochastic Gradient Descent algorithms that were implemented and validated (section 6.3).

- Second, we introduce an unusual threat model capturing the aforementioned key features of the Edgelet computing paradigm and propose robust security mechanisms to protect the data of individuals involved in distributed queries under this threat model.
- Third, we propose and analyze three different resiliency strategies producing valid results while tolerating the failures and message losses induced by the fully decentralized environment. The first strategy is an adaptation of the traditional backup-based resiliency strategies aiming at satisfying together Resiliency, Security, and Validity in an OppNet context, whatever the query. The second strategy capitalizes on the open-world assumption inherent to the Edgelet computing paradigm to improve the execution efficiency of a large class of queries. The third strategy tries to combine the generality and the efficiency of the two preceding strategies. The proposed execution strategies are evaluated and compared to assess their relevance.

This paper is structured as follows. Section 2 presents the Edgelet Computing paradigm with its associated crowd liability model and snapshot-compliant query model. It precisely states the problem at hand. Section 3 details the threat model considered and describes the security mechanisms to enforce the security property. Section 4 presents the three proposed execution strategies, focusing on the mechanisms required to ensure resilient and valid execution despite failures or message loss. Section 5 presents a qualitative evaluation of these strategies and proposes some guidelines to help choose the most adequate strategy depending on the context. Section 6 presents an extensive evaluation of the proposed strategies with the help of the Opportunistic Network Environment (ONE) simulator [41]. Related works are discussed in Section 7. We conclude and present research perspectives in the last section.

2 Edgelet Computing Paradigm

To tackle the privacy protection, data management, and distributed system issues related to this environment, we define in this section the *Edgelet computing* paradigm. First, we describe the considered underlying architecture, its components, and related assumptions. Second, we propose a new model of responsibility adapted to this architecture, to rely on the participants and their devices rather than a central entity. Third, we define a new query model to enable the processing of sample data (i.e., data from a set of individuals available at the time of evaluation) and propose a strawman query execution plan that supports it. We then study the impact of the Edgelet computing context on this execution, which helps us define three required properties to ensure the security, resiliency, and validity of the query execution.

2.1 Key Features of Edgelet Computing

The idea of using the computational resources of personal computers or devices is not new (cf. P2P architectures [6], e.g., BOINC [11]). More recently, [48] proposed to use resources from mobile personal devices, coupled with opportunistic communications to distribute opportunistically some large computations on a set of mobile devices. In this paper, we push this idea a step further by seeking to share both the personal mobile device resources and the personal data of the device owner, as we outlined in the scenarios presented above. To the best of our knowledge, this has never been proposed before probably because of the risks related to the protection of personal data.

As stated in the introduction, the Edgelet computing architecture involves the combination of Trusted Execution Environments and Opportunistic Networks. Here, we discuss the key features of the architecture and then explore how this combination results in a new responsibility model (leveraging the TEEs' security properties) and a new query model (due to the open-world nature of OppNets).

Trusted Execution Environments. Trusted Executions Environments enforce two security properties, namely (1) *data confidentiality*: data manipulated within a TEE device cannot be observed from the outside; and (2) *code integrity*: an attacker cannot influence the behavior of a program executed within a TEE. In this paper, we assume that each crowd member's personal device (named *edgelet node* or *edgelet* for short) is equipped with a TEE guaranteeing to its owner the two previous properties as long as the device is not physically compromised. Indeed, side-channel attacks compromising data confidentiality cannot be totally ignored (though highly difficult to conduct), placing the device in a "Sealed-Glass Proof" mode [65] (i.e., confidentiality is broken, but not integrity).

Opportunistic Networks. The communication infrastructure considered in our scenarios is sidestepping any classical WAN infrastructure (e.g., Internet) for cost or energy constraints, lack of connectivity, security, privacy, or freedom of expression concerns. The communications between edgelet nodes are short-range (e.g., Wi-Fi, Bluetooth) and asynchronous, i.e., there is no time bound on the message transmission delay. For simplicity we consider an epidemic diffusion of the messages, assuming

that messages are transferred from device to device following a store-carry-forward communication protocol [68]. Other more optimized protocols, e.g., exploiting moving patterns of users [36], are considered as future work.

As mentioned earlier, we want to explore the possibility of relying exclusively on secure personal devices to perform the required computations, i.e., without relying on any external infrastructure like central servers. Thus, similar to P2P and cross-device federated learning architectures, we assume processing to be directly distributed across edge devices, without requiring the deployment or maintenance of central servers. The execution of computations in the Edgelet architecture is then achieved in a fully decentralized manner.

To summarize, the Edgelet architecture is characterized by:

- *Edgelet nodes*: Each crowd member's personal device is equipped with a TEE, which provides secure storage and processing capabilities.
- *Opportunistic communications*: Edgelets communicate with each other through Opportunistic Networks whose messages are transmitted at short range with no time limit for their delivery.
- *Fully decentralized execution*: The processing of personal data is fully decentralized on edgelets and does not rely on central servers.

2.2 Crowd Liability Model

Responsibility models are usually introduced to help define the respective responsibilities of all actors involved in a given computing infrastructure. Such models guide judges, practitioners, and researchers when confronted with legal questions related to the protection of data and code. For instance, the *Shared Responsibility Model (SRM)* [61] states the responsibilities of cloud service providers and customers to secure all aspects of a cloud environment. To illustrate this, in an Infrastructure as a Service (IaaS) context, the customer is responsible for the data, application, and Operation System (OS) parts and the cloud provider for the rest of the infrastructure (virtualization, servers, storage, network), while in a Software as a Service (SaaS) context, the latter endorses also the responsibility of the application and OS.

The SRM is generic enough to apply to a large variety of application domains. Assuming personal data is managed in a SaaS context, the customer is the actor playing the data controller role in the GDPR sense (e.g., a company, an administration, an NGO), and the cloud provider plays the role of data processor. Yet, the individuals themselves are no longer in the loop after having given their consent to the data controller to process their data.

Conversely, in crowd computing applications managing personal data, the individual is at the heart of the infrastructure. However, we are not aware of any similar shared responsibility model adapted to crowd computing. Unsurprisingly, no one would agree to endorse the data processor or data controller responsibility in a fully decentralized computing context where each device is under the control of a distinct individual. The consequence is that crowd computing usually handles use cases where the shared data is not really sensitive (e.g., sensor data like temperature or noise captured by a smartphone) but cannot tackle use cases involving sensitive personal data (e.g., medical data) that require tangible security guarantees. In this section, we define such a shared responsibility model, adapted to the Edgelet computing context, that we call the *Crowd Liability Model (CLM)*.

Crowd liability conveys the idea that the data controller is the crowd (i.e., the result of the processing is expected to benefit, directly or indirectly, the crowd members who agree together to the why and the how of this processing); hence there are as many potential data processors as there are crowd members. The corollary of this idea is that each crowd member is expected to do their best to honestly play the fragment of the data controller and data processor roles assigned to them, but the participation in the processing of some dishonest crowd members or of corrupted devices owned by honest crowd members cannot be precluded.

To translate this idea into a responsibility model, we split the data controller into two roles, (1) the *Recipient* which is the Edgelet selected to issue the processing and fairly disseminate the result to the crowd, and (2) the *Regulator* (an external trusted entity or a set of crowd members) which assesses the *Purpose honesty* (i.e., the honesty of the processing purpose) and approves it on behalf of the crowd. Considering that crowd members usually do not have the technical skills to endorse the data processor role, we limit their responsibility to the usage of a genuine TEE-enabled device (*Edgelet physical integrity*) to contribute to the processing (i.e., the crowd member is not liable for potential corruption of their device but they become liable if they tamper with their TEE). Then, we introduce a *Trusted Assistant* role, which is played on behalf of the Edgelet node owners, and encompasses all technical principles embedded in each edgelet node to help the crowd members endorse the fragment of the data processor roles assigned to them. Basically, the Trusted Assistant is expected to guarantee the so-called *Computation honesty*, namely: (1) any decentralized execution strictly complies with the processing approved by the Regulator, (2) any (honest) crowd member can contribute to the computation without risk, even if they lack the technical ability to secure their

device, as long as they are using an up-to-date TEE and (3) any (dishonest) crowd member is defeated when attempting a massive attack.

Note that, as we detail in Section 3, point (3) of Computation honesty necessarily leads to (i) decentralizing operators and data to reduce the benefit of a successful attack by a dishonest crowd member and (ii) randomly selecting the crowd member devices that carry out the computations to reduce the risk of a successful attack by a dishonest crowd member. This further justifies the terminology used in our model, which spreads the liability across the crowd.

Table 1 below transcribes the CLM with the distribution of responsibilities according to tasks and roles.

	Recipient	Regulator	Trusted Assistant	Crowd Members
Result dissemination	X			
Purpose honesty		X		
Computation honesty			X	
Edgelet physical integrity				X

Table 1: Crowd Liability Model (CLM)

Figure 2 summarizes the Edgelet Computing paradigm: (1) The Regulatory Agent checks the honesty of the processing (Query) purpose; (2) the Recipient (in blue) disseminates the Query to the Crowd members through the OppNet. Each crowd member ensures the physical integrity of its Edgelet and the Trusted Assistant ensures that the computation takes place as decided in the Query Execution Plan. The decentralized query execution then takes place between Data processors, manipulating personal data from Data contributors. Finally, the result is sent to the Recipient which fairly disseminates it to the crowd. Edgelet Computing thus differs from Edge Computing [28] since it relies exclusively on the edges to organize the distributed computations, without the need for any infrastructure (see also Section 7). Hence, the CLM Trusted Assistant is implemented exclusively on edgelets, handling both processing and communications, with code integrity guaranteed by TEEs.

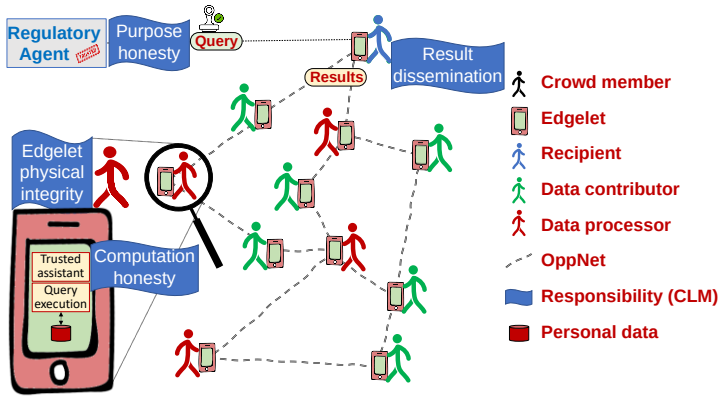

Figure 2: Overview of Edgelet Computing

2.3 Snapshot-compliant Query Model

We consider distributed computations involving personal data hosted in (potentially large) sets of edgelet nodes from smartphones and tablets to more specific smart objects. Moreover, contrary to participatory sensing or sensor networks that focus on stream queries over elementary data, we consider rich data (e.g., healthcare folders, and spending habits) and advanced processing (e.g., database statistics, data mining, and machine learning). We assume that edgelets data can be queried as a shared database with a uniform schema. More precisely, each device may host a set of database schemas, typically one per application domain. The database schemas may be defined by a government agency (e.g., Ministry of Health), a private consortium (e.g., a group of banks and insurance), or an NGO. Consequently, for a given query expressed on a given database schema, the universe of edgelets data E can be seen as a horizontal partitioning of the corresponding global database.

The computations under consideration (called *Query* hereafter) must cope with the uncertainty inherent to the Edgelet setting, making the traditional database closed-world assumption irrelevant. In fact, only a subset of the participants is available

for execution, and only a subset of the data arrives at its destination on time. Thus, considering an open-world query model for edgelets leads us to introduce the notion of *snapshot-compliant query*.

***Snapshot-compliant query**:* Given E the universe of edgelets data and Q a query targeting a dataset $D \subseteq E$, the representativeness of the snapshot D for Q is defined by a set P of predicates over elements of E (e.g., *age* > 65) and by a cardinality (e.g., $|D| = 2000$). We denote by $\zeta_Q(E)$ the set of all representative snapshots of E enforcing P and $|D|$. The Query Q is *snapshot compliant* if the result of Q considering any snapshot of $\zeta_Q(E)$ is equivalent to the Recipient. In other words, a snapshot-compliant query, evaluated on any representative snapshot (enforcing P and $|D|$), will give a satisfactory result for the Recipient.

We consider that a query is expressed by a *Query Execution Plan* (QEP), which is a directed graph where vertices materialize the operators to be computed and edges represent the dataflow among them, with messages sent through the OppNet. The simplest form of a QEP is a tree with *Data Contributors* (DCs) at the leaves, which are the edgelets of crowd members who gave their consent to contribute to the query with their data. Other operators of the QEP are *Data Processors* (DPs), i.e., edgelets that contribute to the processing of the contributed data to produce the final result for the Recipient (R), the root of the QEP. Thus, the DPs either consume the outputs of a set of DCs or the outputs of other DPs.

Strawman execution. Let us introduce the basis for the query model by considering a strawman execution. To satisfy a snapshot-compliant query, we only need to define two types of data processors:

- The *Snapshot Builder* (SB) whose role is to build a dataset, representative from the data transmitted by the DCs.
- The *Computer* (C) whose role is to perform the computation required by the query on the representative snapshot built by SB.

Then, the strawman QEP proceeds in three steps as illustrated in Figure 3.

1. Each Data Contributor DC_i sends its data to the Snapshot Builder.
2. SB builds a snapshot $D = \bigcup_1^n DC_i$ compliant with the set of predicates P and the cardinality $|D|$ and sends it to the Computer.
3. C performs the computation on D and sends the final result to the Recipient.

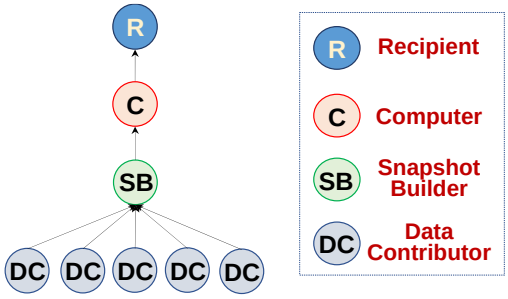

Figure 3: Strawman Query Execution Plan

Note that, since we consider OppNets, we do not consider pipeline communication between edgelets because (1) it generates too many messages, overloading the OppNet and (2) it is very difficult to manage since there is no bound on the transmission delays. Thus, we consider that a DP works in a blocking mode, i.e., it waits for all its input before processing it and producing the output. Similarly, a DC produces all its output in a single message. Hence, any message is sent atomically through the OppNet, i.e., the payload is either totally received by the recipient or not at all. Furthermore, implementing the Snapshot-compliant Query Model in an OppNet context makes it difficult to manage failures, as communications are completely asynchronous and, consequently, the use of reliable failure detectors is impossible (see the Resiliency property below).

2.4 Problem Statement

Ensuring security, fault tolerance, and query validity in our context is difficult because of the conflicting impact of snapshot compliance and crowd liability. Typically, crowd liability inherently requires that the evaluation be distributed across a larger set of devices for security reasons (see the end of subsection 2.2), whereas snapshot compliance would be more easily achieved by using a single crowd member device for processing (typically if a single Data Processor evaluates SB and thus qualifies the snapshot D used for the current query Q, a local check is all that is needed to ensure compliance with the set of predicates P

and the desired cardinality $|D|$). This strawman execution hence does not answer any of the challenges posed by the considered context in terms of security, resiliency, and validity. Based on these observations, we then define the properties required to tackle the problem.

Security. Even if edgelets are secured by TEEs, they can still be attacked (see subsection 3.1). For instance, an execution relying on a single device, as is the case for SB and C in the strawman execution, gives access to the full dataset, that could be fully leaked in case of a successful attack on SB or C. We thus have to circumscribe this risk by decomposing operators (e.g., using horizontal and/or vertical partitioning) into sub-operators assigned to different edgelets. The liability shift to the crowd must be carried out in a context where a few attacked devices can jeopardize the security of the whole system with no way to detect them, leading to the following property:
***Security property*:** Each edgelet integrates mechanisms enforcing the CLM's Purpose honesty and Computation honesty on behalf of the Crowd Members.

Resiliency. Edgelets are personal devices and are therefore susceptible to failure and/or voluntary shutdown. In addition, these devices communicate via OppNets that do not guarantee bounds on communication delays. It is thus necessary to proactively introduce mechanisms to withstand these failures or communication problems. A resilient execution should therefore include backup or replication operators. The objective here is to guarantee that the execution has a significant chance of success, especially when the execution is decomposed on a large set of edgelets.
Since reliable failure detectors cannot exist in OppNets due to the unpredictability of message delays, it makes it difficult to predict the time to build a snapshot from random contributors and to execute a query. The system's liveness must then be guaranteed based on fault presumptions only and on the probability of success for queries associated with a deadline (i.e., a maximum time allowed for executions):
***Resiliency property*:** Given a probability of fault presumption p_f for any edgelet, a *query deadline*, and an expected probability of success p_s, a query Q must be completed before the deadline with a probability greater than p_s.

Validity. Enforcing the two previous properties leads to a much more complex QEP (with several SBs, several Cs as well as with backup edgelets and/or replicated operators) which could then result in an invalid computation if inconsistencies occur at some point. It is therefore necessary to ensure that the execution is valid, either by adding synchronization mechanisms before the computation, or verification mechanisms after the computation. The snapshot consistency must be preserved all along the query processing despite presumed faults and message loss between Data Processors to guarantee a consistent result:
***Validity property*:** The Edgelet query execution result must be identical to a centralized query execution over at least one snapshot of ζ_Q(E). Formally, $\forall D_i \in \zeta_Q(E)$, $\exists D_j \in \zeta_Q(E)$ / $Q_E(D_i) = Q_C(D_j)$, with Q_E (resp. Q_C) denoting the Edgelet (resp. centralized) execution of a query Q.

To summarize, we have defined three properties that are particularly challenging to tackle together given their mutual impact. Security is addressed in Section 3 while Resiliency and Validity are addressed in Section 4.

3 Security Enforcement

This section is devoted to the security mechanisms and algorithms required for enforcing the Crowd Liability Model (CLM), specific to the Edgelet computing paradigm. The two cornerstones of the CLM are *Purpose honesty*, which deals with the approval by the crowd of the why and the how of the processing, and *Computation honesty*, which ensures the trustworthiness of the processing during its decentralized execution. These two principles are respectively addressed in subsections 3.2 and 3.3 . However, since the CLM defines new roles and responsibilities and relies on specific security assumptions, a dedicated threat model must be defined first.

3.1 Dedicated Threat Model

A dedicated threat model is required to capture (1) the shift of responsibility from a usual central entity (i.e., the data controller in the GDPR) to the crowd and (2) the TEE trustworthiness. We define this model as follows.

Ingenuous Recipient.

- *Role*: initiates the processing of a QEP and receives the final results. Does not take part in the execution.
- *Played by*: depending on the use case, one or more crowd members, medical workers, statistical agencies, etc.
- *Trust:* We do not want to impose the Recipient to be equipped with a TEE-enabled device to avoid reducing the targeted

use cases, explaining why we exclude it from the actual processing of the QEP. However, we do not question the good faith of the Recipient. Thus, even if it receives the results of the processing in clear text, we do not consider inference attacks from its side (i.e., crossing the results of multiple queries).

Wolf in sheepfold Participants.

- *Role*: contributes with data (*Data Contributor*) and/or processing power (*Data Processor*) in a QEP.
- *Played by*: any participant equipped with a TEE-enabled device.
- *Trust*: as said in subsection 2.1 , side-channel attacks on a TEE-enabled device cannot be totally precluded despite their complexity (requires tampering with the device). We assume a large majority of honest participants (the lambs) and a few "sealed glass-proof" ones (the wolves).

Regulator.

- *Role*: reviews and approves the processing to be performed. This role is not devoted to the query execution itself.
- *Played by*: either an external entity (e.g., a privacy regulatory agency like the CNIL in France) or a representative group of crowd members engaged in a collective validation process.
- *Trust*: full.

3.2 Purpose Honesty

In this subsection, we focus on the initialization of queries and how the Recipient can prove its honesty. To this end, we propose to build on the manifest-based framework [42], so that the recipient can declare the why and how of processing. Here is its implementation in our context:

First, the Recipient specifies a manifest describing the query to be performed. This manifest consists of four elements: (1) the general purpose of the processing expressed in natural language, (2) the Query Execution Plan, (3) the set of representativeness predicates P of the targeted dataset D as well as its cardinality $|D|$ and (4) the source code of the operators to be executed on each edgelet node. Subsequently, the Recipient submits the manifest to the Regulator for verification of its compliance with the expected privacy practices. Once verified, the Regulator signs the manifest and returns it to the Recipient. This certified manifest will be used for the edgelet assignment protocol presented in the next subsection, after which the query manifest will be ready for dissemination in the Opportunistic Network.

Let us now examine how the threat model and the manifest-based framework are applied to the motivating example presented in the introduction, namely, the DomYcile project [24]. In this project, nearly 8,000 patients are each equipped with a secure home box where their medical and social records are stored. These boxes are not connected to the Internet for subscription cost, security/privacy, and acceptability reasons and are only accessible at the patient's home by healthcare workers. They leverage their mobility to transmit messages from their smartphones to the boxes via Bluetooth, applying a store-carry-forward strategy and thus implementing an OppNet. Assuming that a Recipient (e.g., patient associations, statistical agencies, medical workers) wants to query a cohort of consenting elderly patients to obtain statistical results in the spirit of [34] (Share EU project [64]), the query deployment, illustrated in Figure 4, proceeds as follows:

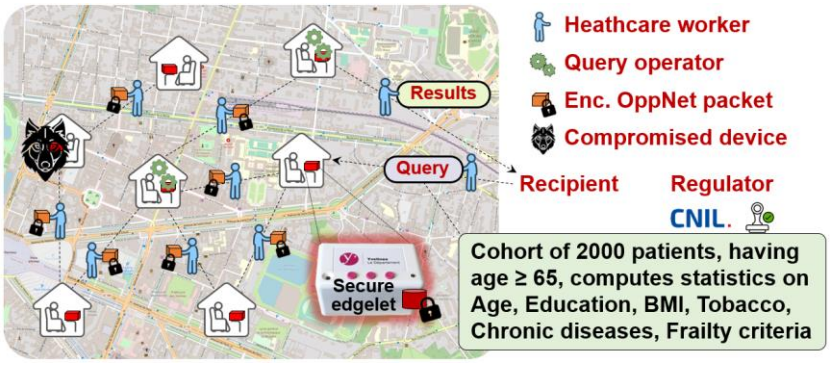

Figure 4: Edgelet Query in the DomYcile Project

1. The Recipient specifies the manifest containing the four elements mentioned above, including the general purpose as follows: "Cohort of 2000 patients, having age ≥ 65, computes statistics on Age, Education, BMI, Tobacco, Chronic

diseases, Frailty criteria".

2. The Recipient sends the manifest to the Regulator, e.g., the CNIL (French regulatory agency), which sends back a signed version of the manifest if approved.
3. The Recipient randomly assigns QEP operators to a subset of edgelets (see subsection 3.3.2), making the query manifest ready for dissemination.
4. Healthcare workers broadcast the query through their OppNet.
5. Patients willing to contribute their data to the query will then act as Data Contributors. Edgelets assigned by the Recipient will themselves act as Data Processors.

The challenge is now to protect the confidentiality of the computations despite potential attacks on the edgelets (wolfs), which is the topic of the next subsection.

3.3 Computation Honesty

Now that the certified manifest is disseminated to all edgelets in the network, we want to ensure the CLM's Computation honesty reminded here for readability: (1) each decentralized execution must strictly conform to the QEP approved by the Regulator, (2) each honest crowd member must be able to confidently participate to the processing despite potential corruption of their own device, and (3) each dishonest crowd member is defeated in their attempt to perform a massive attack. For the two first points, the question is how to verify that the execution is done in the same order as specified by the certified QEP and how to ensure that the processing performed on the data does not deviate from the original code. This is addressed in subsection 3.3.1 whereas point (3) is the focus of subsection 3.3.2 .

3.3.1 Global and local integrity of the processing

To enforce points (1) and (2), we leverage the code integrity property provided by the TEE to build a Trusted Assistant, a logical entity taking responsibility for the Computation honesty on behalf of the edgelet owners. Concretely, the Trusted Assistant runs on each edgelet node a piece of code, called *Core* hereafter, which is part of the TEE Trusted Computing Base, a code base guaranteed genuine at boot time. Using remote attestations [42, 45] (cryptographic proof of the authenticity of the TEE), the Core enforces point (1) of CLM's Computation honesty by guaranteeing that the predecessors of Data Processors in the QEP are legitimate and produce valid intermediate results. In addition, the Core attests to the genuineness of the QEP operator's code assigned to the participating edgelets by verifying the signature of the Regulator on the certified manifest. Note that even if the edgelets' operating system is corrupted by malware or viruses, the isolation property of the TEE ensures that the executed code cannot be altered. Hence, point (2) of CLM's Computation honesty is also enforced.

These two integrity guarantees provided by the Trusted Assistant via the TEEs reinforce the crowd's confidence in the Edgelet computing architecture. As a result, each individual can freely consent to contribute their sensitive data with the reassurance that the decentralized execution will follow precisely the precepts indicated by the certified manifest. As we will see below, the next challenge is to protect executions from dishonest crowd members.

3.3.2 Resistance to massive attacks

As mentioned before, we do not exclude the possibility of side-channel attacks on edgelets compromising the confidentiality of the manipulated data. Our objective is then to empower the Trusted Assistant with mechanisms that minimize the risks of massive data leakages in this particular case.

The Recipient is in charge of initiating the query, that is, assigning the QEP operators to the participating edgelets, and then launching the processing. However, the threat model makes no assumption about the integrity of the Recipient(s). Even if we do not question their good faith, we acknowledge that their information system may have been corrupted by some attacker. Thus, the task of assigning operators to edgelets must be protected to prevent a critical operator from being assigned to a corrupt accomplice edgelet, e.g., the operator manipulating the data of a targeted contributor or the operator manipulating a maximum amount of sensitive data. We, therefore, propose a random assignment protocol auditable by the Trusted Assistant.

Random assignment. We assume that the set of all crowd members' edgelet is known by the Recipient (e.g., they register to join a community) and that their IDs form a hash ring (as in a Chord DHT). We also assume that each crowd member tacitly consents to contribute to a query execution, which is far less engaging than consenting to contribute their personal data to the query. Under these assumptions, the assignment protocol is as follows (see the pseudocode below):

1. The Recipient first checks if the Manifest is genuine (lines 1&2). Then it computes a hash of the manifest (signed by the Regulator and publicly known) as a seed for the random process (line 3) and assigns the first operator to the edgelet having the ID immediately greater than this hash in the ring (line 5). The first hash is rehashed to assign the second operator and so forth until all operators have been assigned (lines 4-7). Finally, the assignment is signed (line 8).
2. The Trusted Assistant, which implements Core on all edgelets, verifies first the signature of the Manifest and the assignment (lines 1&2), then checks the chain of hash in order to detect any fraudulent assignment for the operator intended for them (lines 3-13). Assuming that each edgelet knows at least the ID of its predecessor in the ring, it is sufficient to check that the hash is strictly larger.

```
Random_Assignment (run by the Recipient)
Input:  M a manifest; M.S the Regulator's signature of M; M.OP the set of operators to be assigned to edgelets;
        R a ring with sorted identifiers of registered edgelets
Output: A={(op, id)} an assignment of operators to edgelets; A.S a signature of this assignment
1. if M.S is not a valid signature for M then
2.   return ERROR
3. seed ← hash(M)                                    // compute a hash of the manifest
4. for each op in M.OP do
5. |  id ← find_edgelet_id(R, seed)                  // find an edgelet ID satisfying prev(id) < seed and id > seed
6. |  seed ← hash(seed)                              // rehash the current hash value
7. |  A ← add_assignment(A, op, id)                  // add (op, id) to A
8. A.S ← sign_assignment(A, Recipient's private key)  // Sign the assignment A with the Recipient's private key
9. return A, A.S


Verify_Assignment (run by the Trusted Assistant, with edgelet id = my_id)
Input:  M a manifest; M.S the Regulator's signature of M; A={(op, id)} an assignment of operators to edgelets; A.S a
        signature of this assignment
Output: boolean (true if the assignment is correct, false otherwise)
1. if M.S is not a valid signature for M or A.S is not a valid signature for A then
2.   return false
3. my_op ← op in A such that id = my_id   // the operator assigned to "my" edgelet
4. seed ← hash(M)                                  // compute a hash of the manifest
5. prev ← previous_edgelet_id             // id of the previous edgelet in the ring
6. for each op in A do
7. |  if op == my_op then                 // if the operator is the one assigned
8. |    if prev < seed and my_id > seed then        // if my_id is immediately greater than this hash
9. |      return true
10.|    else
11.|      return false                    // invalid assignment detected
12.|  seed ← hash(seed)                   // rehash the current hash value
13. return false                          // invalid assignment detected
```

How does this assignment work when successive hashes lead to collisions? Note that this problem is frequent when there are few edgelet nodes present in the DHT ring, and it becomes highly unlikely otherwise. However, assigning multiple roles to a single node is undesirable for both resiliency and privacy reasons, hence the need for a countermeasure. When a collision occurs, we propose that the Recipient assigns the direct successor of this node in the ring, as the latter has a lower probability (squared) of also being in collision. The Trusted Assistant can still detect a fraudulent assignment, but this time each edgelet must know the IDs of its two predecessors. The proposed method is therefore adjustable to any dimension of the network, it will just be necessary to increase the number of known predecessors.

Next, let us try to restrict the leakage to dishonest crowd members, those for which the edgelet is physically attacked. Trivially, if no Data Processor is compromised, the TEE confidentiality property guarantees that no sensitive data can leak by construction. However, the distributed executions we are considering require intermediate results to be transmitted between the Data Processors (e.g., the representative snapshot *D* sent by the Snapshot Builder to the Computer). The messages sent in the OppNet then need to be encrypted to counter any malicious interception (e.g., by the untrusted smartphone of a healthcare worker).

Messages encryption. We assume that the QEP transmitted by the recipient contains the certificates of the assigned data processors, consisting of the IDs of the nodes in the DHT and their public encryption keys. Since the QEP is constructed statically, each node can easily encrypt its output based on its successors in the execution tree. Thus, when an edgelet e_i needs to send a message *m* to e_j, it generates a symmetric key k_s to encrypt its message and uses e_j's public encryption key pk_j to

encrypt k_s. The packet sent in the OppNet is then: {enc(m,k_s), enc(k_s, pk_j)}. Following this procedure, all messages in transit in the network are encrypted, making them unreadable to anyone other than their recipients. Therefore, since no cryptographic material is ever shared among edgelets and the TEE integrity property still holds even in sealed-glass proof mode, the potential leakage is reduced to the data processed by a compromised edgelet. Note also that any change in the operator's ordering would make the messages indecipherable and the execution would fail, reinforcing point (1) of the CLM's Computation honesty.
Finally, we need to restrict the amount of data manipulated by each edgelet, so that in the event of a physical attack on assigned edgelets, leakage is limited to a small proportion of the data required for a query.

Horizontal and vertical partitioning. We observe that computations of interest are often distributive (e.g., MapReduce, Spark), enabling the decomposition of processing into sub-operators. Thus, we propose to distribute the operators of the Data Processors (Snapshot Builder and Computer) among different edgelet nodes. This decomposition can help minimize the amount of data exposed at each edgelet by horizontally partitioning the dataset. This can also preclude the concomitant exposure, in the same edgelet, of data items that become sensitive when combined (e.g., a quasi-identifier) by vertically partitioning the dataset. Note that such distributive executions can also help minimize the workload (e.g., when energy consumption matters) by exploiting the inherent Edgelet computing parallelism. Figure 5 presents both types of partitioning, horizontal and vertical, on the query example of Figure 4. Each Data Contributor performs a hash function of its ID to select the Snapshot Builder to send its data to, so that each partition processes only a fragment of the dataset (here, a tenth) with different Computers depending on the statistics to be computed (each one only sees the attributes strictly necessary for its computation). Note that a Computing Combiner operator must be added in the QEP to combine the outputs of all sub-operators.

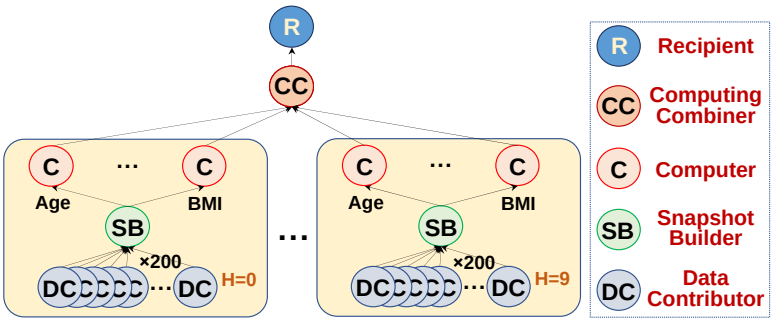

Figure 5: Horizontal and Vertical Partitioning

Although these partitioning strategies are decided by the Recipient in the design of the QEP, it is the responsibility of the Regulator to approve them. Depending on the application context, the privacy criteria selected may be different (e.g., medical records, spending habits) and involve more or less distribution of the data. Note that since each individual is free to consent to contribute their data, it is in the Recipient's interest to design a privacy-friendly QEP in order to encourage people to participate.

4 Resiliency and Validity Enforcement

The execution plans presented earlier are relevant from a logical point of view but are not resilient to failures. Indeed, the slightest failure or unavailability among the assigned edgelets results in the complete failure of the query. Our objective here is then to propose execution strategies that make Edgelet processing resilient to failures and to message losses in the Opportunistic Networks context.

In the following subsections, we present three alternative implementation strategies for enforcing the *Resiliency* property and discuss their impact on *Validity* and *Security* (a more detailed analysis of the exposure of personal data when the confidentiality property of some TEEs is breached is conducted in Sections 5 and 6). First, we consider an extension of traditional resilience solutions based on backups (subsection 4.1). Second, we present an alternative approach, in which, instead of replicating input data contributions, we tolerate an open (over-)set of data contributors typical of the edgelet context and take advantage of this "overcollection" of data to ensure Resiliency (subsection 4.2). Third, we present a "hybrid" approach in which the backup and overcollection modes can coexist for better efficiency (subsection 4.3).

4.1 Backup-Based Strategy

In this subsection, we take a conservative approach to Resiliency, recovering from failures in a general way, independent of query plans, and study its impact on the enforcement of the Validity and Security properties.

4.1.1 Enforcing Resiliency

No reliable failure detector exists in our context, and every Data Processor (SB, C, and CC in Figure 6) is a potential Single Point of Failure (SPF). In the Backup-based approach, we simply try to recover from failures, whatever the Data Processor presumed faulty, the benefit of which is to make the handling of Resiliency independent of the form of the QEP. We use timeouts to presume faults and secure the execution of all SPFs by means of backups, as usual [69].

We distinguish between Passive and Active Backups. A Passive Backup replicates the input data of its corresponding SPF, called primary, and is activated and processes this data only in case the primary is presumed faulty. Thus, the data transferred to a Passive Backup is not exposed since it remains encrypted until the backup node is actually required. Conversely, an Active Backup executes in parallel with its primary. Despite a reduced latency, Active Backups incur a higher resource consumption and offer less security (greater exposure of data). Consequently, all SPFs are passively replicated (see Figure 6), except the Computing Combiner which must be actively replicated; otherwise, the Recipient would be forced to take part in the processing, at least to activate the Computing Combiner Backup(s), which would damage the Ingenuous Recipient assumption (see subsection 3.1).

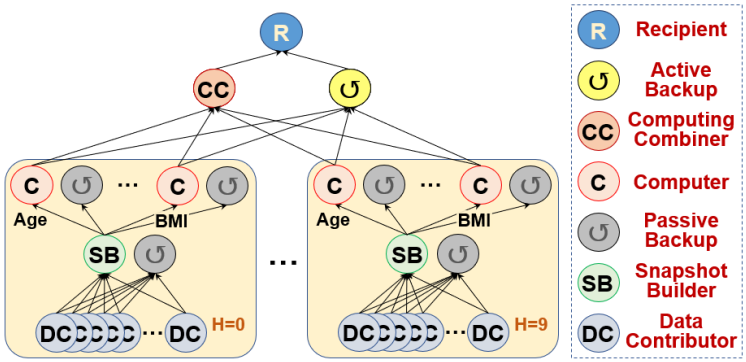

Figure 6: Backup-based Strategy

Figure 6 presents the Backup strategy for the QEP of the motivating example with horizontal and vertical partitioning. For this execution plan to be fault tolerant, each SPF in each partition must survive (i.e., Snapshot Builder and Computers), either by means of a primary node or one of its backups (note that in the figure, only one backup per SPF is shown). The number b, which represents the number of backups per primary node in each partition, is then determined by the following inequality: $\left(1-{p_f}^{(1+b)}\right)^{|SPF|\times n} \geq p_s$, with p_f the probability of fault presumption, p_s the expected probability of success, and $|SPF|$ the number of SPFs in each of the n horizontal partitions of the QEP (n=10 in Figure 6). As we will also see in the other strategies, the Computing Combiner is considered separately from the other Data Processors due to its particular role as a proxy for the Recipient. Following the same principle, the number of backups b_{cc} for the Computing Combiner results in the inequality $\left(1-{p_f}^{(1+b_{cc})}\right) \geq p_s$.

We assume that a start date T_s is set in the query manifest from which all Data Contributor edgelets start responding to the query. This date is then known by the Data Processor edgelets assigned for the query. Given the maximum delay δ for a sent message to arrive at its destination node (with a very high probability, close to 1), the maximum execution time of the query to successfully terminate can be estimated, which is referred to as the query deadline calibration. To this end, intermediate timeouts must be set at each Data Processor node so that the backups of their predecessor node(s) can be appropriately activated at runtime. For example, in Figure 6, the Computing Combiner and its active backups must allow sufficient time for all predecessor Computers to recursively activate the Snapshot Builder backups. Readers interested in a more detailed presentation of the query deadline calibration are referred to Appendix 1.

4.1.2 Enforcing Validity

Satisfying the Validity property requires that the data flow between the operators is consistent w.r.t. at least one of the ζ_Q(E) snapshots. However, with failure or message loss, a Snapshot Builder and its backups may build different snapshots, each belonging to $\zeta_Q(E)$. Therefore, three situations must be distinguished to guarantee that the query result is equivalent to the one obtained with a centralized execution over at least one snapshot built by the Snapshot Builder or one of its backups. This particular snapshot is hereafter called the *reference* snapshot.

1. Without any partitioning, a single Snapshot Builder feeds a single Computer. Hence, a reference snapshot can be identified whatever the execution. It is either the snapshot built by the Snapshot Builder primary if there is no fault presumption, or the snapshot built by one of the activated backups otherwise.
2. Similarly, with horizontal partitioning, the reference snapshot is simply the union of each partition's snapshot (backup or primary).
3. With vertical partitioning, the Snapshot Builder feeds several Computers. Some of these Computers may consider the primary snapshot, while some others may use the backup snapshot in case of fault presumption. This leads to inconsistency, i.e., a result built over a snapshot that does not belong to any snapshot of $\zeta_Q(E)$.

Figure 6 illustrates this third situation, where each Computer evaluates a different statistic but must consider the same snapshot belonging to $\zeta_Q(E)$. To solve this problem, several solutions can be envisioned but all of them incur a significant overhead. We sketch below two of them that address this issue in a different way:

Consensus: Synchronize the snapshot between the primary Snapshot Builder and its backups thanks to a consensus protocol (Figure 7.a). [9] proposes an effective consensus protocol for OppNets that matches the Edgelet context, at the price of a distributed consensus. This consensus stage requires active Snapshots Builders backups in order to agree on a representative reference snapshot. Indeed, if the collected data had to remain encrypted, it would become difficult, if not impossible, to verify the representativeness predicates.

- *QEP restructuring*: Rearrange the QEP so that the parallel branches are serialized, one after the other (Figure 7.b), thus avoiding inconsistencies due to multiple successors, but at the price of losing QEP parallelism and increasing drastically the recursive runtimes. This solution implies extending the deadline, otherwise, the intermediate timeouts would be too short and almost all the backups would be triggered, which would harm the privacy of the contributors since their data would be exposed several times.

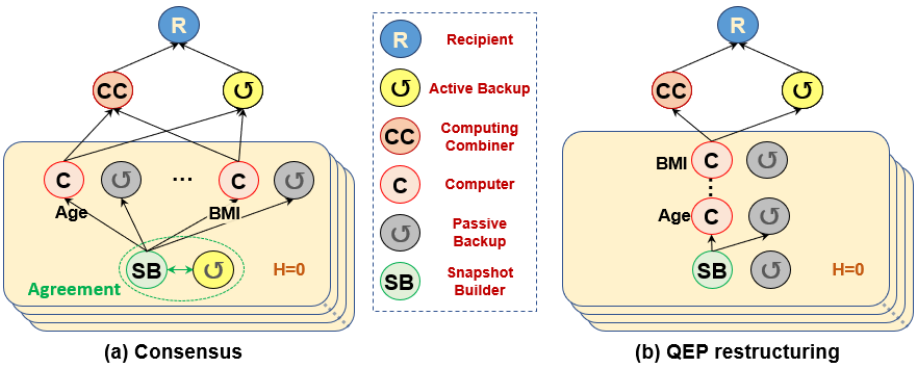

Figure 7: Methods to guarantee the Validity property

4.2 Overcollection-Based Strategy

This subsection introduces a very different way to handle the problem of Resiliency, which integrates the OppNet context by design. Contrary to the backup-based strategy, message delays or loss are no longer considered as faults that must be recovered but rather as legitimate behavior.

4.2.1 Enforcing Resiliency

As an alternative to securing every SPF in a QEP thanks to backups, we suggest over-collecting the dataset of interest so that the QEP may survive the loss of parts of it. To explain the intuition, let us consider a query over a sample dataset (e.g., *2000* individuals with *age > 65*) where Data Processors (i.e., Snapshot Builder and Computers) execute distributive operators. Instead of executing the operators on single edgelets, we distribute (using hashing) its execution over *n+m* edgelets where each one processes a partition of the original dataset, with *n* the minimum number of partitions to be collected and *m* the overcollection parameter (see Figure 8).

The Overcollection ratio must be adapted to the presumed fault probability p_f of the OppNet to reach the expected success rate p_s for a query. Given that the probability of success of one partition is $p_p = (1-p_f)^{|SPF|}$, with $|SPF|$ the number of SPFs in each partition, the number *m* of additional partitions is determined by the inequality $\sum_{i=n}^{n+m}\binom{n+m}{i} \times {p_p}^i \times (1-p_p)^{(n+m-i)} \geq p_s$, i.e., at least *n* of the *n+m* partitions must succeed. Concerning the Computing Combiner, we assign Active Backups exactly like those of the Backup strategy.

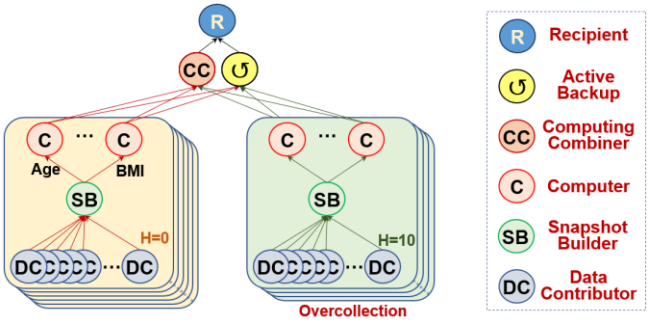

Figure 8: Overcollection-based Strategy

4.2.2 Enforcing Validity

If all QEPs can satisfy the Validity property when executed in backup mode (in some cases by adding a consensus among backups), this does not hold true when they are executed in Overcollection mode. Indeed, (1) the complete QEP must be reorganized to handle a partitioned dataset and (2) a reference snapshot $D \in \zeta_Q(E)$ must remain identifiable despite the arbitrary loss of subparts of this dataset during the processing. To tackle point (1), a brute-force solution is to reorganize the QEP in a set of sub-QEPs, each performing independent processing over a partition of the collected dataset with the Computing Combiner assembling the final result (see Figure 8). This solution applies only if the commutativity rules between operators allow to push all distributive operators down to the sub-QEPs and to push all non-distributive operators up to the Computing Combiner. A QEP satisfying this condition is said *reshapable*. Under this assumption, addressing point (2) can be easily achieved. The reference snapshot *D* is simply the union of all partitions that contributed to the QEP computation up to the Computing Combiner. Assuming that each of the *n+m* partitions locally satisfies the set of representativeness predicates *P* and has a cardinality $|D|/n$, the Validity property is trivially preserved as long as less than *m* partitions are lost.

4.2.3 Relaxing Validity

Most data-intensive queries of interest in our context are inherently distributive, as evidenced by the various implementations in MapReduce or Spark. However, some of the corresponding QEPs cannot be reshaped following the Brute-Force approach and then cannot combine Overcollection and Validity. This is notably the case of general interest machine learning algorithms because they are iterative or need to exchange partial results computed over different data partitions. In these cases, a reference snapshot $D \in \zeta_Q(E)$ can no longer be identified in case of message loss (e.g., two iterations may consider a different snapshot state). On the other hand, strict Validity is not a prerequisite for these algorithms which usually produce an approximate result. We thus suggest another basic preliminary method to handle these cases, called Iterative Brute-Force.

To execute an algorithm $\mathcal{A}$ with Iterative Brute-Force, each edgelet implementing a sub-QEP Computer iterates on (1) a *local convergence* phase where it computes $\mathcal{A}$ on its local partition and improves its local knowledge, initialized by a parameter of the sub-QEP, and broadcasts this knowledge to all others sub-QEPs, and (2) a *synchronization phase* where it receives the

knowledge of the other sub-QEPs it has heard of and integrates them in its own knowledge. Right before the query deadline, the knowledge is sent to the Computing Combiner which combines all received pieces of knowledge and sends the final result to the Recipient.
The main question is when to stop the processing. Fixing the number of iterations a priori (with a minimal number of received messages) is not meaningful in the OppNet context where message delays, then edgelet progression, are unpredictable. Expecting local convergence is also risky due to the instability of the synchronization phase among sub-QEPs. For instance, two edgelets with fast communications could converge locally quickly (without having even received any message from others) and decide to end prematurely their computation. Thus, we enforce the progression of the algorithm on all edgelets thanks to a *Heartbeat*, that is each iteration is cadenced by a clock[2], whatever the local state of the processing (i.e., a Computer moves to the next iteration even if few or no messages were received). Finally, the local result is delivered when the deadline is imminent.

```
Iterative Brute-Force
Computer Edgelet (local_partition, initial_knowledge)
  knowledge ← initial_knowledge
  Heartbeat until (query_deadline − 1 round)
    Local conv: knowledge ← 𝒜(local_partition) and broadcast knowledge to all
    Synchro: knowledge ← received knowledge of others
  Send final knowledge to Computing Combiner
Computing Combiner
  combine all received knowledge and send it to Recipient
```

We illustrate this method using three classical algorithms: Apriori, K-means, and Stochastic Gradient Descent (SGD). We demonstrate its effectiveness in terms of the proximity of results to centralized executions in subsection 6.3 . While the first two algorithms have specific tasks, the SGD algorithm is employed to solve many well-known machine-learning models in the literature [15, 71]. Its relevance in the Edgelet context further emphasizes the applicative potential of this approach.

Apriori [59]: mines frequent itemsets to learn association rules.
- *knowledge*: frequent itemsets and their support (initially empty).
- *Local convergence*: $\mathcal{A}$ first computes the local support of all frequent itemsets in its partition then iteratively computes the local support of itemsets that are frequent in other sub-QEPs it has heard of.
- *Synchronization*: adds frequent itemsets of others in knowledge.
- *Computing Combiner*: sums the local supports of the common itemsets found in all received knowledge.

K-means [23]: forms k clusters minimizing the intra-cluster variance.
- *knowledge*: current centroids (initially, k randomly chosen centroids)
- *Local convergence*: until local convergence or heartbeat, $\mathcal{A}$ assigns each element of its own partition to the cluster having the nearest centroid and recomputes the centroids of the new clusters, updating its knowledge.
- *Synchronization*: computes, on a cluster basis, the barycenter of all centroids received from other sub-QEPs, and integrates the result in knowledge.
- *Computing Combiner*: computes the barycenter of all centroids received.

SGD [43]: adjusts the weights of a model to minimize its objective function.
- *knowledge*: vector of weights *w* (initially random).
- *Local convergence*: until local convergence or heartbeat, $\mathcal{A}$ computes the gradients associated with the *w* vector for a small sample of its data (mini-batch) in order to iteratively update the model.
- *Synchronization*: computes the average of all *w* vectors received from other sub-QEPs it has heard of, and integrates the result in knowledge.
- *Computing Combiner*: computes the average of all *w* vectors received.

[2] Note that the heartbeat mechanism needs a clock on each edgelet (to know when it is supposed to finish a computation iteration and send the knowledge to others), but those clocks do not need to be synchronized between edgelets.

4.3 Hybrid-Based Strategy

As the name suggests, the Hybrid strategy proposes a new method that combines the previous mechanisms, namely Backup and Overcollection, with the advantage of being able to adapt and benefit from each method depending on the situation.

4.3.1 Enforcing Resiliency

We have seen that the Backup strategy causes inconsistency problems when the QEP includes vertical partitioning. To avoid these problems, the idea of the Hybrid method is to assign backups only to Computers and use the Overcollection principle to compensate for the probability of fault presumption of the Snapshot Builders (see Figure 9). In fact, integrating backups within partitions is a smarter resiliency solution than continuously adding overcollection partitions, especially when the degree of vertical partitioning is high (since each SPF must survive for the partition to succeed). In this configuration, the probability of success of one partition is determined by the equation $p_p = (1 - p_f) \times (1 - p_f^{(1+b)})^{(|SPF|-1)}$, with p_f the probability of fault presumption, $|SPF|$ the number of SPFs in the partition (i.e., Snapshot Builder and Computers) and b the number of backups per Computer. As with the Overcollection strategy, the success for the $n+m$ partitions is constrained by the inequality $\sum_{i=n}^{n+m} \binom{n+m}{i} {p_p}^i (1 - p_p)^{(n+m-i)} \geq p_s$, with p_s the expected probability of success. Once again, we use Active Backups to make the Computing Combiner fault tolerant.

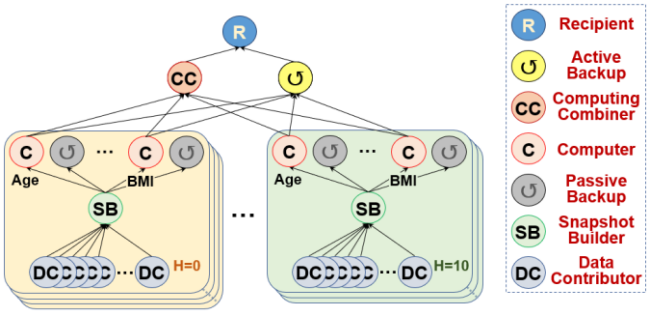

Figure 9: Hybrid-based Strategy

Note that both parameters b and m can be adjusted to achieve the success rate p_s, thus we can design different optimization strategies depending on whether it is better to have more backups or more overcollected partitions. For example, we can choose them to minimize the number of nodes involved in the QEP or to minimize the number of messages sent over the network.

4.3.2 Enforcing Validity

Similar to the "pure" Overcollection strategy, the Hybrid execution plans use only one Snapshot Builder per partition. This mechanism ensures by construction that the snapshot is unique for each partition. Thus, even if the QEP incorporates vertical partitioning, the Computers are guaranteed to be consistent because they process the same snapshot for each partition. However, the Validity is not so trivially guaranteed when the queries are iterative and the Computers of multiple partitions have to exchange information before producing their final result. Indeed, as explained in subsection 4.2.3 , the strict Validity property would require that all Computers in all partitions exchange the same information (data, aggregates, or hints) at each iteration to form a consistent view. But if we follow the same approach as before, i.e., relaxing the property to tolerate partial and non-uniform exchanges between Computers, we then lose interest in having added backups to the Computers since each Computer will only need to receive information from a subset of the others, and therefore there is no necessity to activate the backups. This particular issue will be further detailed in the next section.

5 Qualitative Analysis

This section first makes a qualitative analysis of the exposure risk associated with each resiliency strategy (backup, overcollection, and hybrid). We then compare these strategies based on their respective scope and provide some guidelines to select the most appropriate strategy.

5.1 Exposure Risk

For the backup strategy, we notice that the risk of exposure of individual data is not the same whether the backups are passive or active. Indeed, as we explained previously, Passive Backups only decrypt data if they are activated, unlike Active Backups which decrypt them anyway. This protection is possible because, even if the edgelet is compromised, the integrity property of the code is still ensured (see subsection 2.1) and guarantees that data can be decrypted only at the reception of an activation message from a successor node. Moreover, we consider that the activation messages are signed with the private key of the successor nodes, making them impossible to forge. Hence, besides horizontal and vertical partitioning, Passive Backups provide an additional layer of protection against confidentiality attacks on edgelets. Their use on all QEP operators is however not always possible, especially with the Computing Combiner which must be actively replicated, and with the Snapshot Builders when the QEP integrates vertical partitioning and a consensus protocol, as explained above.

Ensuring resiliency through Overcollection changes the way data is exposed. On the one hand, the fact that there are no backups ensures by design that individual data is only exposed once, as only the primary nodes will be able to decrypt the data and, in case of compromise, the data leakage is restricted to the compromised edgelets (see subsection 3.3.2). However, the Overcollection mechanism necessitates the involvement of more Data Contributors in the query to ensure the success of the execution. Therefore, the disadvantage of the individual exposure in the Backup strategy is replaced by a higher collective exposure.

Finally, since Hybrid execution considers a single Snapshot Builder per partition, it avoids any backups on Snapshot Builders (active or passive), which limits the exposure of individual data involved in the representativeness predicates to a single node (instead of $1 + b$ for the Backup strategy). Moreover, the mix between backup and overcollection mechanisms allows for a trade-off between individual and collective exposure. Determining which exposure is superior is challenging since it depends on the context. We will see in subsection 6.1 that this exposure is often an order of magnitude lower than Backup alone or Overcollection alone.

5.2 Guidelines

The goal of this subsection is to guide a potential Recipient toward the right execution strategy when designing a computation dedicated to the Edgelet computing paradigm. This choice depends on the type of computation, the method for defining a representative snapshot, and the expected query deadline. We can draw some guidelines from the statements summarized in Table 2.

<table>
<tr><th>Resiliency</th><th>Type of computation</th><th>Snapshot definition</th><th>Validity</th><th>Confidentiality</th><th>Success Rate</th></tr>
<tr><td rowspan="4">Backup-based Strategy</td><td>Horizontally partitionable</td><td>P partitionable</td><td>By construction</td><td rowspan="3">Activated backup exposed</td><td rowspan="3">Requires large query deadline</td></tr>
<tr><td>Vertically partitionable</td><td>Any P</td><td>Consensus</td></tr>
<tr><td>Others</td><td>Any P</td><td>By construction</td></tr>
<tr><td colspan="3">Iterative</td><td colspan="2">Unrealistic</td></tr>
<tr><td rowspan="3">Over collection-based Strategy</td><td>QEP reshapable</td><td>P partitionable</td><td>By construction</td><td rowspan="2">Over collected data exposed</td><td rowspan="2">Supports very small query deadline</td></tr>
<tr><td>Iterative reshapable</td><td>P partitionable</td><td>Ground validity</td></tr>
<tr><td colspan="3">P not partitionable</td><td colspan="2">Invalid</td></tr>
<tr><td rowspan="2">Hybrid-based Strategy</td><td>QEP reshapable</td><td>P partitionable</td><td>By construction</td><td>Activated backup and over collected data exposed</td><td>Supports small query deadline</td></tr>
<tr><td colspan="3">Iterative or P not partitionable</td><td colspan="2">Inappropriate</td></tr>
</table>

Table 2: Taxonomy of Execution Strategies

In this table, *Horizontally*/*Vertically partitionable* refers to the property of the computation to be distributed among several Data Processors, as explained in subsection 3.3.2 . Both forms of partitioning greatly make sense when conceiving a computation dedicated to the Edgelet computing paradigm, either to minimize the amount of data exposed at each Data

Processor or to avoid the exposure in the same Data Processor of information sensitive when combined or even to minimize the Data Processor workload when energy consumption matters. While vertical partitioning does not impose additional constraints on the processing, horizontal partitioning requires that the set of representativeness predicates P be itself partitionable, i.e., that it can be applied to each partition independently (e.g., *age > 65* is partitionable while median(*age*) < *10* is not).

QEP reshapable refers to the capacity to reorganize distributive and non-distributive operators in the QEP to be computed. This is a prerequisite for exploiting the Overcollection-based strategy for this QEP. *Ground validity* means that the Validity property defined in subsection 2.4 cannot be enforced; hence, it is up to the Recipient to assess empirically the accuracy of the final result, as we will do in subsection 6.3 for Apriori, K-means, and SGD. Finally, the *Confidentiality* column expresses the additional amount of data exposed by each strategy compared to an ideal strategy without resiliency (i.e., without backups nor overcollection).

Based on this table, we can draw the following conclusions. First, if the QEP implementing the computation cannot be reshaped or if *P* is not partitionable, the Backup strategy is the only solution since applying overcollection would be equivalent to completely duplicating the query. Second, if the computation is iterative, the Overcollection strategy turns out to be the only solution as well. Indeed, using backups, a consensus (or an equivalent mechanism) would be required at each iteration to ascertain that all participants consider *in fine* the same reference snapshot. Otherwise, only a Ground validity can be expected, but Overcollection outperforms Backup and Hybrid in this case (no additional protocols related to interactions with backups). Third, if the QEP is reshapable and *P* is partitionable, the right choice between Backup, Overcollection, and Hybrid is driven by the expected success rate and by privacy considerations. Using Backup or Hybrid strategies, the query deadline must be calibrated to accommodate the number of backups defined at each QEP level (i.e., activate them one after the other in case of fault presumption), a factor which disappears with Overcollection alone for which the query deadline depends only on the average network latency and the number of levels in the QEP. This explains the query deadline requirements, which range from large for the Backup strategy to very small for the Overcollection strategy.

Regarding privacy, the three methods do not expose data in the same way. Indeed, as explained in the previous section, TEEs guarantee that data at rest is not exposed in backups until they are activated. Hence, the same personal data is potentially exposed in as many backups as required by the satisfaction of the Resiliency property and this number increases with the presumed fault rate of the OppNet. With Overcollection alone, in contrast, the same data is never exposed twice. However, data from a larger population of individuals must be involved in the computation due to overcollection. The Hybrid strategy combines both types of exposure, with the advantage of being able to adjust the proportion of each one by calibrating the number of backups and the number of overcollected partitions. This decision made by the Recipient regarding how to expose personal data may impact the approval of the Regulator and the consent of the participants.

The taxonomy of solutions presented in Table 2 is still preliminary and more subtle guidelines can be envisioned. In particular, the Ground validity should be more deeply investigated with the goal of identifying finer classes of algorithms for which better validity guarantees can be expected. For instance, the Apriori implementation sketched in subsection 4.2.3 exhibits the salient feature that Validity can be assessed *a posteriori* by the Computing Combiner. Indeed, (1) the reference dataset is the union of the partitions it received from the Data Processor it has heard of, and (2) a frequent itemset is necessarily frequent in at least one of these partitions. The Computing Combiner must simply check that enough information has been received to compute the support of all these candidate frequent itemsets. We expect also to be able to guarantee the convergence for some algorithms when specific conditions are met but let these issues for future work.

6 Quantitative Analysis and Validation

This section presents our quantitative evaluations of the execution strategies presented earlier. Our objective is to validate the relevance of the Edgelet approach, calibrate the system parameters, and verify its effectiveness.

We first compare the three execution strategies, namely Backup (Bak), Overcollection (Ovr), and Hybrid (Hyb), providing insights to properly configure the resiliency parameters. Then, we implement a non-iterative Edgelet execution with the aim of calibrating the query deadline and achieving the targeted success rate. Finally, we test the iterative methods Apriori, K-means, and Stochastic Gradient Descent to evaluate the quality of their results against a centralized execution.

6.1 Comparison of Execution Strategies

In this first subsection, we want to study the respective behavior of the execution strategies presented in Section 4. To enable a fair comparison between the three resiliency mechanisms, we assume that the query deadline is correctly calibrated so that each message sent in the OppNet has enough time to reach its destination (see subsection 6.2 for deadline adjustment). Consequently, the probability of fault presumption p_f is reduced to the probability of device failure. Our goal is to observe the consequences of the resiliency methods, particularly in terms of personal data exposure and but also in terms of network overload. Table 3 below shows the notations needed to understand the formulas given in the following.

Notation	Description
n	Number of initial partitions (horizontal partitioning)
m	Number of additional partitions (in case of overcollection)
$\|C\|$	Number of Computers in each partition (vertical partitioning)
$\|SPF\|$	Number of Single Point of Failure in each partition (1 Snapshot Builder + $\|C\|$ $Computers$)
b	Number of backup nodes associated with each SPF
$\|D\|$	Cardinality of the dataset required for the query
$\|CC\|$	Number of Computing Combiners (1 + b_{cc} specific backup nodes)

Table 3: Notations

6.1.1 Overall Analysis

To begin with, we examine the general impact of execution strategies on the Query Execution Plans (QEPs). We want to observe the transformations induced by the resiliency mechanisms (Bak, Ovr, and Hyb), and to do so, we count the number of additional nodes introduced in the QEPs. Since these are redundant nodes, their presence in large numbers will be seen as a disadvantage, as they lead to additional data exposure (in case they are compromised) and network overload.

	Passive Nodes	Active Nodes
Bak	if (|C|=1): $\|SPF\| \times b \times n$ else: $\|C\| \times b \times n$	if (|C|=1): 0 else: $b \times n$
Ovr	0	$\|SPF\| \times m$
Hyb	$\|C\| \times b \times (n + m)$	$\|SPF\| \times m$

Table 4: Formulas for the Additional Nodes

Table 4 shows the formulas used to determine the number of additional nodes for each strategy. We distinguish two types of nodes: those that are active, working in parallel with the primary nodes (active backups and nodes in overcollected partitions); and those that are passive, waiting to be activated (passive backups). For Bak, the number of backups *b* is calibrated using the formula of subsection 4.1.1 . Note that when |C|>1, we consider that the Snapshot Builders' backups must be active to establish a consensus on the dataset (see subsection 4.1.2). For Ovr, the number of additional partitions *m* is adjusted using the formula of subsection 4.2.1 . For Hyb, the pair (*b*, *m*) is chosen using the formula of subsection 4.3.1 in order to optimize the sum of the number of additional nodes (passive and active).

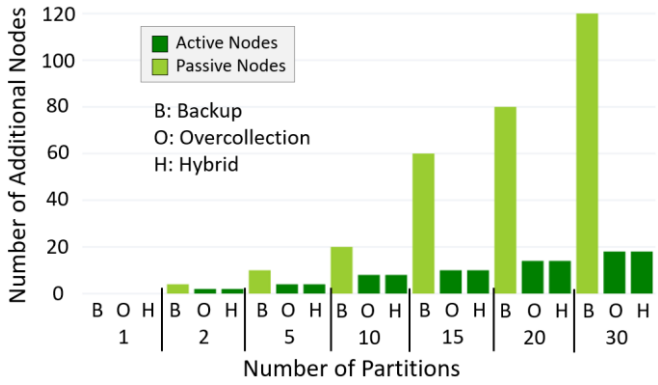

Figure 10: Additional Nodes per Number of Partitions

Figure 10 presents the number of additional nodes for each strategy depending on the number of partitions (horizontal partitioning). We first study a basic setup with a single Computer in each partition (|C|=1) and a device failure probability of 10% (p_f=0.1) for a success probability of 80% (p_s=0.8). We observe that increasing the number of partitions greatly favors strategies exploiting the overcollection principle (Ovr and Hyb). Moreover, Hyb perfectly mimics Ovr by increasing the number of additional partitions *m* while maintaining the number of backups *b* equal to zero. Note that even if the additional nodes are only passive for the Bak strategy, we saw in Section 5 that they necessarily induce extra overhead. Indeed, they generate a lot of communication for data replication, hence the objective to minimize their number.

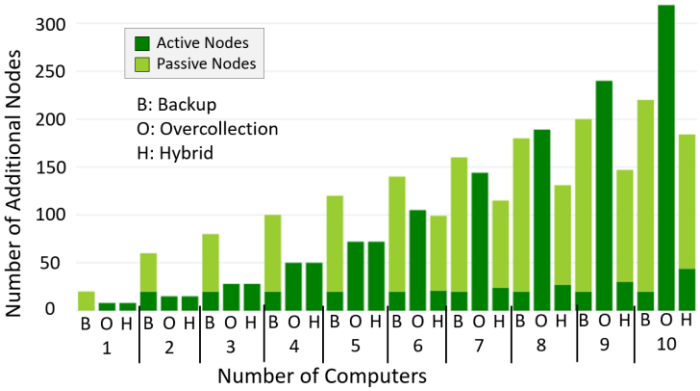

Figure 11: Additional Nodes per Number of Computers (pf =0.1)

Now that we have varied the horizontal partitioning, let us focus on the vertical partitioning, taking the same configuration as in previous chapters, i.e. *n*=10. Figure 11 shows that when the number of Computers per partition is low (|C|<6), Ovr and Hyb are identical and outperform Bak. From |C|≥6, we observe that Hyb begins to distinguish itself by finding a compromise between *b* and *m* that enables it to perform strictly better than the other two strategies. From |C|≥8, we see an inversion between the Bak and Ovr plots, with the number of additional nodes shifting in favor of Bak. As shown in Figure 12, this inversion occurs earlier as the probability of failure increases.

The explosion in the number of additional nodes for Ovr is explained by the fact that, for the strategy to succeed, at least *n* partitions with all SPFs must "survive" failures, whereas for Bak, only one primary or backup Data Processor per SPF (Snapshot Builder and Computers) is needed in each partition to guarantee successful executions.

Based on these results, we can draw the following conclusions: (1) horizontal partitioning greatly favors strategies based on overcollection (Ovr and Hyb), (2) vertical partitioning is detrimental to the Ovr strategy, especially when the probability of failure increases, and (3) the Hyb strategy can outperform Bak and Ovr by leveraging the strengths of both strategies.

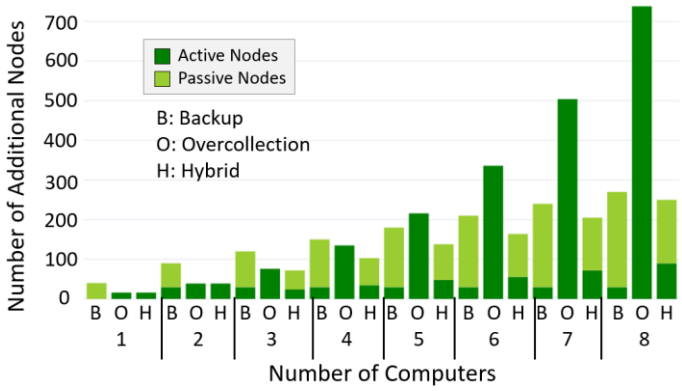

Figure 12: Additional Nodes per Number of Computers (pf=0.2)

6.1.2 Personal Data Exposure

Let us now consider the influence of these resiliency mechanisms on the exposure of personal data, both individually (i.e., how often the same data is exposed) and collectively (i.e., how many additional contributing individuals are included in the QEP). Note that this is potential exposure since the confidentiality property of TEEs is only broken if the devices are physically compromised (see subsection 2.1).

	Individual	Collective
Bak	if (|C|=1): min=0, max=2 × b else: min=b, max=2 × b	0
Ovr	0	m/n
Hyb	min=0, max=b	m/n

Table 5: Formulas for the Potential Additional Exposure

Table 5 gives the formulas used to calculate the potential additional exposure. Individual exposure indicates the number of times the same personal data could be exposed. The minimum is determined by the number of active backups in the QEP, whereas the maximum is determined by the number of passive backups. Note that we count two additional exposures for Snapshots Builders and Computers backups, but that the number of Computers |C| is not considered, as they process data of different individuals (otherwise, increasing vertical partitioning would undermine privacy). Regarding collective exposure, we consider the proportion of additional individuals involved in the QEP, which is determined by the m/n ratio. Note also that, although these two measures aim to quantify exposure, they don't have the same meaning which makes them very difficult to compare. Individual exposure reveals the risk incurred by each individual, while collective exposure indicates the total number of individuals at risk. Depending on the application scenario and the type of queries, it will be the responsibility of both the Recipient and the Regulator to find the right compromise.

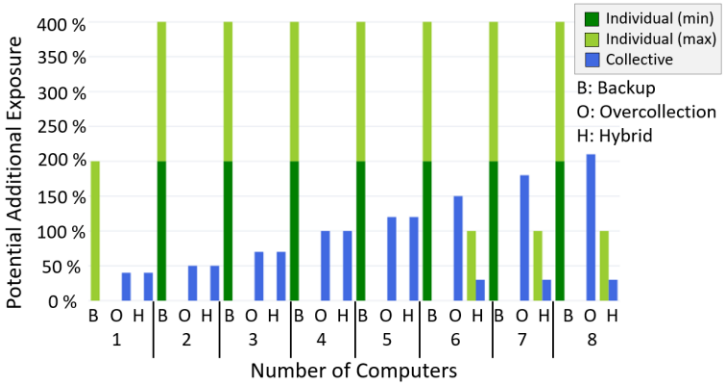

Figure 13: Potential Additional Exposure per Number of Computers (pf=0.1)

Figure 13 shows on the same graph both types of exposure as a function of the number of Computers |C|. Although |C| is not directly considered when calculating personal data exposure, we saw in the previous subsection that it strongly influences the number of backups *b* and the number of additional partitions *m*, which therefore impacts it indirectly. Let us take a look at what happens at |C|=2. For Bak, we observe that the minimal additional individual exposure is at 200%, indicating that the data could be exposed to 2 active backups (corresponding to Snapshot Builders) and that the maximal additional individual exposure is at 400%, meaning that the data may also be exposed to 2 passive backups (those belonging to Computers). For Ovr and Hyb, we see an additional exposure at 50%, meaning that on the n=10 initial partitions, it is necessary to add m=5 other partitions to reach the target success rate of 80%.

Similar to the analysis in the previous subsection, we see that, as the number of Computers |C| increases, the Ovr strategy deteriorates. For example, when |C|=7 and p_f=0.2, we obtain an overcollection rate of over 600% (see Figure 14). For Hyb, we observe a mixture of the two exposures. Indeed, above a certain threshold, the Hybrid method simultaneously integrates passive backups and additional partitions, with the (b, m) pair again chosen to reduce the number of nodes in the QEP. In fact, optimizing these parameters according to personal data exposure is rather complicated and application-dependent, as the two measures (individual and collective) are not comparable. Nevertheless, we can see that this strategy seems to be a good compromise when the other two are no longer acceptable (e.g., |C|=8 on Figure 13: one additional backup and 30% additional collective exposure).

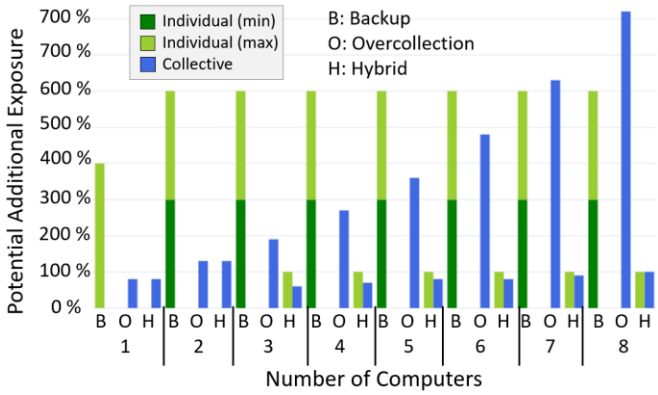

Figure 14: Potential Additional Exposure per Number of Computers (pf=0.2)

6.1.3 Network Overload

To complete this analysis, we propose an evaluation of network overload by counting the number of additional messages generated by each execution strategy. Although our objective is not to optimize network communications (see Section 2), we nevertheless wish to study the impact of strategies on the congestion they may generate. To this end, we are interested in measuring the number of messages containing data sent over the network, ignoring any other type of message (e.g. backup activation messages). As message size depends on the use case and the type of query, we assume a single size for all messages sent (whether data contribution or intermediate result).

	Mandatory Messages	Potential Messages
Bak	if (|C|=1): $(\|D\| + n) \times b$ else: $[\|D\| + (2 + b) \times \|C\| \times n] \times b$	if (|C|=1): $(1 + b + \|CC\|) \times b \times n$ else: $\|C\| \times b \times \|CC\| \times n$
Ovr	$[\|D\|/n + \|C\| \times (1 + \|CC\|)] \times m$	0
Hyb	$[\|D\|/n + \|C\| \times (1 + \|CC\|)] \times m +$ $\|C\| \times b \times (n + m)$	$\|C\| \times b \times \|CC\| \times (n + m)$

Table 6: Formulas for the Additional Messages

Table 6 provides formulas for counting the number of additional messages generated by the three resiliency strategies (Bak, Ovr, Hyb). We observe two categories of messages: mandatory messages, which are necessarily sent when the QEP is executed,

and potential messages, which are sent when passive backups are activated. Note that the number of messages sent by Data Contributors can be much higher than the required cardinality of the dataset $|D|$. Indeed, snapshot construction requires an unpredictable number of contributions in order to satisfy the set of representativeness predicates P. As this unpredictable quantity of messages is context-dependent, we do not include it in our measurements.

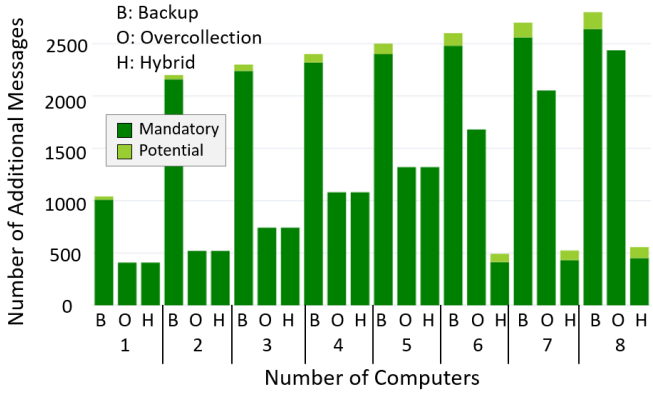

Figure 15: Additional Messages per number of Computers

Figure 15 shows the number of additional messages according to the number of Computers. We have calibrated a horizontal partitioning of n=10, a failure probability of 10%, and a success rate of 80%. For these parameters, the QEP needs only one Computing Combiner, i.e. no active backups associated (b_{cc}=0). The cardinality of the target dataset is set to $|D|$=1000. As expected, the vast majority of additional messages are associated with the contribution phase. Bak is therefore disadvantaged, as every message sent by Data Contributors is replicated on the Snapshot Builder's backups. As for Ovr and Hyb, the strategies are again identical as long as |C|<6. This observation, already made in Figure 11, is explained by the fact that Hyb is still optimized to minimize the additional nodes in the QEP.

Let us now look at Figure 16 to see how Hyb behaves when the pair (b, m) is calibrated to optimize the number of additional messages (mandatory and potential). We can see that Hyb is systematically better than the other strategies and that the gap widens as |C| increases. In fact, similar to Ovr, the Hybrid strategy is naturally designed to minimize the number of messages sent during the contribution phase because Snapshot Builders have no backups. The advantage of Hyb over Ovr is its ability to compensate for the probability of partition failure by adding backups to the Computers. This feature proves to be a significant asset, as reducing the number of additional partitions results in a substantial reduction in the number of additional messages sent. Note that this optimization is not without cost, as it inevitably entails modifications in the exposure of personal data. For instance, with p_f=0.1 and |C|=3, Hyb will count 1 backup per Computer for a 20% overcollection instead of 0 backup for a 70% overcollection (see Figure 13).

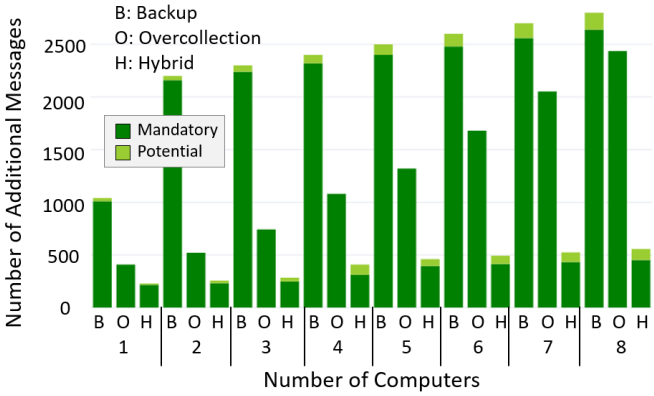

Figure 16: Additional Messages per number of Computers (Hyb optimization)

6.2 Adjustment of the Query Deadline

In this second subsection, we focus on how to calibrate the query deadline such that messages sent in the OppNet have sufficient time to reach their destination. Without loss of generality, we will focus on implementing a non-iterative processing executed

with the Ovr strategy. Indeed, adding backups (for Bak and Hyb) only shifts the deadline according to the procedure described in subsection 4.1.1 . Moreover, we have just seen in the previous subsection that the Ovr strategy is better in most configurations (with Hyb matching it by imitation). So, to study the query deadline, we built an Edgelet computing software on top of the Opportunistic Network Environment (ONE) simulator [41] providing detailed traces of OppNets communications (see Figure 17). We model two representative use cases with messages exchanged using an epidemic routing strategy [68], namely *Mall* and *DomY*.

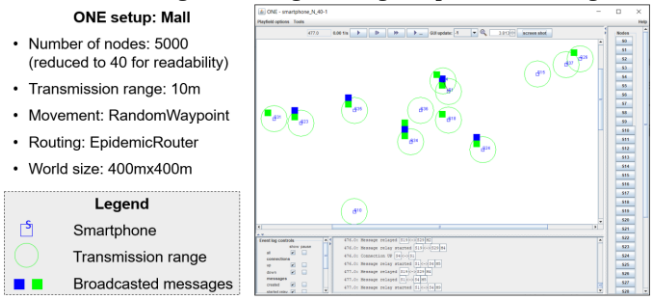

Figure 17 Mall simulation with the ONE

Mall: Edgelet computing within a Mall of 0.16 km2, where 5,000 edgelets (customer smartphones following a RandomWayPoint movement) are opportunistically executing a query exchanging messages using Bluetooth when meeting. The mean OppNet latency obtained with ONE is $\bar{L}$ = 1,936 s for a standard deviation σ = 933 s, resulting in a relative standard deviation of $R\sigma = \frac{\sigma}{\bar{L}} = 0.48$

DomY: The DomYcile project [24], with 8,000 personal home boxes (edgelets) and 800 healthcare workers (with constrained routes in ONE) within the Yvelines district (2,284 km2). We obtained a mean latency $\bar{L}$ = 27,113 s with a relative standard deviation $R\sigma$ = 2.43, (i.e., σ = 65,794 s).

Using the latencies computed by ONE, we execute a non-iterative QEP following the Ovr strategy, considering vertical partitioning on 3 Computers with a device failure probability of 10%. We used 3 values of *m*/*n*, ranging from 0.5 to 1.5. To enable comparisons between Mall and DomY, we used on the x-axis the query deadline divided by $\bar{L}$, called α hereafter, and measured the query success ratio (y-axis on Figure 18). To minimize random variations, we averaged the results of 300 executions. Note that we are only interested in the impact of message transmission along the QEP and not in studying the quality of its processing, the latter being easily assured as long as fewer than *m* partitions are lost (see subsection 4.2.2). Therefore, we consider an execution successful when at least *n* partitions transmit their response to the Recipient before the query deadline.

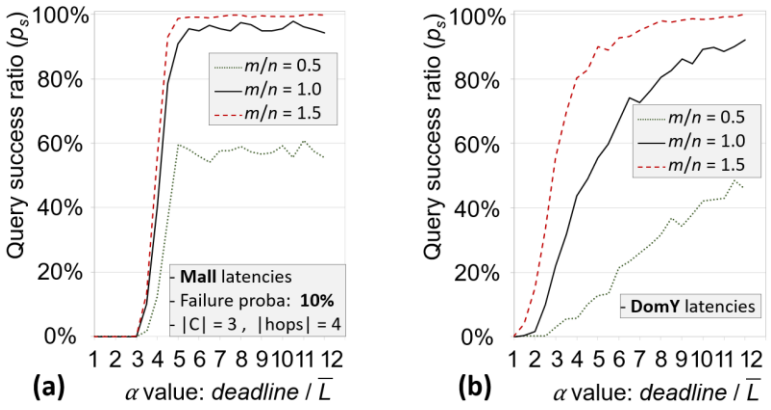

Figure 18: Query Deadlines (for Overcollection)

From these plots, we make 3 observations: (1) in the Mall context (small $R\sigma$), all executions complete successfully with $m/n \geq 1$ and $\alpha = 4$ (vertical increase); while the DomY context requires much larger deadlines due to its larger $R\sigma$ which impacts the fault presumption; (2) having an underestimated *m*/*n* value is risky: it reduces the ratio of successful queries and requires a significantly higher query deadline in contexts with large $R\sigma$ (e.g., DomY); (3) having a larger *m*/*n* value is rather useless for

small $R\sigma$ contexts (e.g., Mall), indeed, a small $R\sigma$ means that latencies are close to the mean, thus a well-calibrated m/n is sufficient to absorb few late messages.

In conclusion, the m/n ratio should not be underestimated and the query deadline should be fixed larger than $\bar{L}\times|\text{hops}|$ where $|\text{hops}|$ is the number of hops in the query plan (in this case, 4: Data Contributor ➛ Snapshot Builder ➛ Computer ➛ Computing Combiner ➛ Recipient). Both m/n and the deadline should be overestimated when the OppNet latencies have a large $R\sigma$. Thus, the query deadline should be fixed (for a 4 hops query) around 2 days for DomY and 2-3 hours for Mall, values that are quite reasonable given our application context.

6.3 Quality of Iterative Computation

In this third subsection, we examine the Edgelet iterative execution of the algorithms presented in subsection 4.2.3 . Our approach is empirical: we aim to demonstrate the relevance of the Iterative Brute-Force method by verifying the quality of the results obtained compared with a centralized execution.

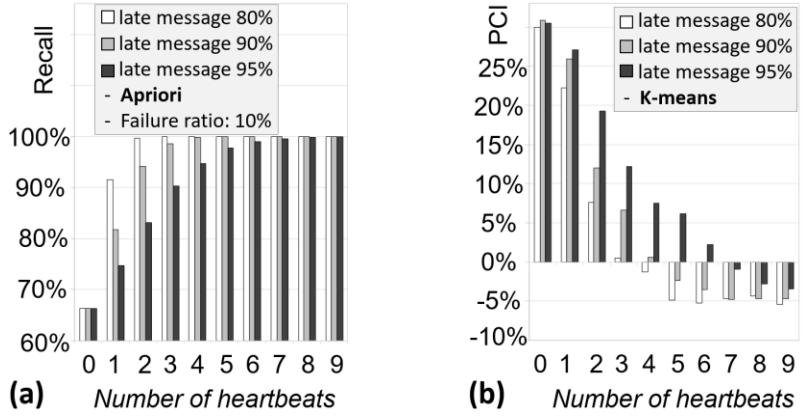

Figure 19: Heartbeat Execution Quality
(Apriori and K-means)

We first simulate the iterative algorithms Apriori and K-means, considering synthetic and real data sets used to evaluate their quality (e.g., [16] for Apriori). In these Edgelet executions, we employed a QEP with a horizontal partitioning, consisting of n=10 partitions and m=10 additional partitions. We systematically measure the quality of Edgelet executions against centralized fault-free executions (y-axis on Figure 19). More precisely, for Apriori, we compare the association rules generated after frequent item mining in Edgelet executions to those obtained in centralized ones and use the precision/recall metrics to assess the comparison. Similarly, for K-means, we compute the Percentage Change Inertia (PCI), i.e., the percentage change between the Edgelet inertia (intra-cluster variance) and the centralized one. The x-axis indicates the number of heartbeats during the query. For assessing iterative methods under extreme conditions, we deliberately reduced the heartbeat duration, resulting in observed proportions of late messages of 80%, 90%, and 95% (as depicted in Figure 19).

We observe that even with no iteration, Apriori reaches 65% recall and K-means reaches 30% degradation of its inertia. Both converge quite quickly towards a recall of 100% or a PCI < 0% (4 or 5 heartbeats for 80 or 90% late messages, 7 or 8 with 95% late messages). Note that with Apriori, precision is always 100% (not shown). Indeed, as we verify that the Computing Combiners always receive n or more sub-QEPs results, we can thus remove potential false positives. With K-means, we observe that as the number of heartbeats increases, the PCI can become negative. The reason for this is that an Edgelet calculation with many heartbeats can be better than a centralized calculation, thanks to the fact that it considers up to m additional partitions.

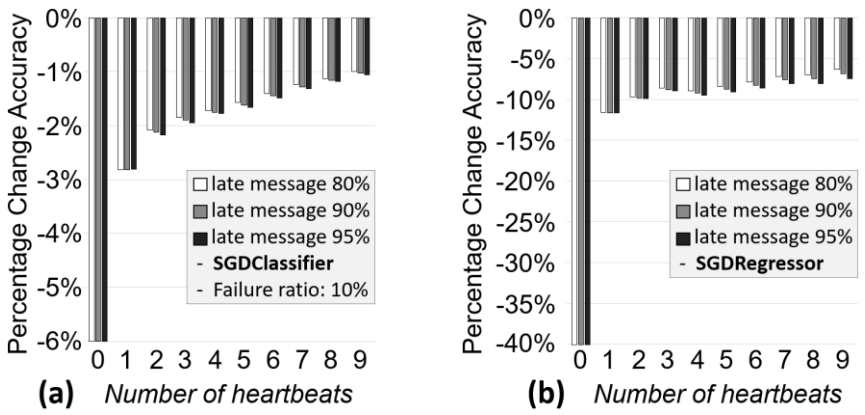

Figure 20: Heartbeat Execution Quality (SGD)

We then simulated the iterative Edgelet execution of Stochastic Gradient Descent (SGD) on a classification problem (Adult Income [52]) and a regression problem (California Housing [60]). As for Apriori and K-means, we consider a QEP with a horizontal partitioning of n=10 for m=10 additional partitions. For classification, the underlying learning model used is a support vector machine, while for regression it is a linear model. For both, we use a constant learning rate and divide the number of maximum local iterations by a factor of 50 compared to centralized executions in order to reduce the amount of work on each edgelet. We then compare the accuracy of the model obtained on test data (mean accuracy for classification and coefficient of determination R^2 for regression).

Figure 20.a and Figure 20.b display the results of heartbeat executions on the two problems (classification and regression). The y-axis represents the Percentage Change Accuracy (PCA), which is the percentage difference between the model accuracy achieved in an Edgelet execution and that obtained in a centralized execution. At first glance, we can see that models built in Edgelet gradually converge towards the quality of centralized models. Even if the initial PCA is variable (-6% for classification and -40% for regression), since it depends on the learning rate and the maximum number of local iterations chosen, it does not prevent the models from progressing. Then, we notice that the proportion of late messages is not a determining factor for convergence speed. In fact, the SGD algorithm's ability to work on small samples (mini-batches) at each iteration makes it a prime candidate for distributed and decentralized execution [43]. As a result, when the data is correctly distributed over the Computers, they compute a local model that naturally converges toward the best parameters.

7 Related Works

A lot of research work has been produced on the different areas encompassed by Edgelet computing: computing architectures, privacy-preserving computation techniques, and fault-tolerant execution strategies. To the best of our knowledge, none of these works considered the conjunction of fully decentralized data-intensive computing on users' devices and the enforcement of validity, resiliency, and security properties. In the following, we overview the closest decentralized computing architectures, then consider user privacy protection, and finally study existing strategies for fault tolerance.

Decentralized computing architectures. Crowd Computing [32], Crowdsourcing [26], and Crowd Sensing [30] are all based on the fact that individuals are increasingly connected and can thus participate in the enrichment of the digital sphere, from the sharing of captured data to the realization of individual micro-tasks. However, these approaches are still driven by centralized servers on which most of the processing is done. At the same time, the Edge paradigm has emerged with the goal of offloading services and computation close to data sources to make the cloud more responsive, scalable, and privacy-friendly [62, 70]. Like Cross-Device Federated Learning [13], these technologies operate in connected infrastructures where processing is performed either in micro data centers (or cloudlets) [28, 50, 58], or directly on edge devices with workload distribution [35, 46]. We differ from these architectures by still offering a reliable and privacy-preserving approach, but by relying exclusively on the edges to organize the processing without the need for any infrastructure. In this direction, an approach quite similar to ours is the crowd computing definition by Murray et al. [48] which considers the combination of mobile devices and OppNets to achieve large-scale distributed computations. However, this proposal does not consider privacy or fault tolerance.

Privacy-preserving computations using cryptography. To guarantee privacy-preserving computations, our approach relies on TEEs present in edgelets, which promotes computational genericity and execution scaling. Numerous cryptography-based solutions have been proposed for almost thirty years. Initial solutions were designed for wireless sensor networks [30, 73] with

the main aim of transmitting collected data to monitoring stations for analysis, but the field has recently taken off again to meet the needs of federated learning (FL). A recent survey [44] discusses 40 works for FL, grouped into four classes: masking, homomorphic encryption, functional encryption, and MPC (or secret shares). The specific scheme proposed may vary depending on the trade-off offered between the computation and the communication cost, or the tolerance to failures. However, whatever cryptographic scheme is used, all existing solutions (e.g., [12, 53], Prio [21], and Drynx [29]) rely on a similar architecture in which one or a few powerful and highly reliable servers collect encrypted user data and then apply costly (albeit optimized) aggregation algorithms based on masking (e.g., [63]), garbled circuits (e.g., [7, 25]) or secret sharing (e.g., [27, 33]). For example, distributed database solutions, such as SMCQL [7], are limited to a few dozen participants; existing distributed computing schemes based on fully homomorphic encryption [14, 31] cannot support any type of operation and scale to a large number of participants at the same time.

Privacy-preserving computations using differential privacy or gossip protocols. Local differential privacy (LDP) has seen significant growth in recent years due to its main advantage over classical differential privacy, namely that it does not require a trusted third party. Existing works address problems such as machine learning [8], federated learning [1, 66, 72], marginal statistics [74], or basic statistics based on range queries [20]. However, LDP accentuates the tension between utility and privacy protection as it requires more noise for the same level of protection as classical differential privacy [3]. This can therefore either affect utility or require a very large number of participants to reduce the impact of noise which limits its applicability in our context. Gossip-based protocols are scalable, fully decentralized, reliable, and have an adjustable accuracy. Unfortunately, classical gossip-based protocols do not protect user privacy. In [8, 56], participants collectively learn a machine learning model while preserving their privacy through differentially private models, impacting accuracy. In [47], participants introduce noise in the first iterations and gradually remove it in subsequent iterations. This approach makes such solutions unreliable in the event of node failures or opportunistic networks. Finally, gossip protocols require the participation of all the nodes of the system, limiting their applicability.

Strategies for fault-tolerance. The standard approach to ensure query termination in distributed systems is replication [4, 69]. Recent database systems, such as AnyLog [49], address the case of the edge-cloud continuum, where applications typically have low latency, sporadic connectivity, and generate large amounts of data, raising new resilience issues. Resilience issues in the context of IoT are also more specific. We refer the reader to [10] for a full classification of resilience issues. Fault-tolerance problems increase significantly when considering asynchronous systems, and the duality between safety and liveness still impacts the most recent works. For instance, [51] proposes a redundancy system called Replicated Dataflow Graph (RDG) and suggests a "routing constraint" mechanism to coordinate data sources and replicas. However, as the authors explain, this does not totally prevent the occurrence of routing inconsistencies, although infrequent in practice. Our backup-based strategy, rooted in these same principles, encounters similar challenges. Conversely, overcollection-based strategies (Overcollection and Hybrid) circumvent the problems by replicating operators rather than data, which has the advantage of ensuring data consistency when building snapshots. Regarding the conjunction with privacy protection techniques, some works [12, 44] have investigated device failures and dropouts in the context of reliable aggregation servers. Unfortunately, these methods cannot be applied in a fully decentralized setting for two reasons. First, scalability is an issue because an edgelet is not a high-end server capable of handling thousands of connections and related cryptography. Second, reliability is a concern in our context since the Data Processor nodes are unreliable as they can fail or drop out, rendering their solutions inoperative. Our proposed TEE-coupled resiliency strategies adapt to this fully decentralized environment while ensuring computational genericity and scalability with no compromise on utility.

8 Conclusion

In this paper, we explored the Edgelet computing paradigm, a new framework for executing powerful and privacy-preserving distributed queries on personal devices at the extreme edge of the network. Built on the recent convergence between Trusted Execution Environments and Opportunistic Networks, this paradigm opens disruptive ways to handle personal data processing, pushing computations directly onto crowd devices rather than relying on centralized servers. Besides introducing this new paradigm, this paper makes the following contributions. First, to cope with the specificities of the Edgelet context, it introduces a new responsibility model tackling the shift of responsibility from cloud providers to crowd members and a new query model adapted to the open-world assumption. Second, by leveraging the properties of TEEs, it provides several security mechanisms to preserve both execution integrity and data confidentiality under a threat model dedicated to the Edgelet computing context. Third, it proposes three resiliency strategies to make the executions resilient to personal device failures and OppNets-induced message losses: the first one is a thorough adaptation of backup-based strategies, the second one capitalizes on the open world

assumption to improve the efficiency of a subclass of queries and the last one is a hybridization aiming at combining the generality and the efficiency of the two preceding strategies. It then provides guidelines for determining the most appropriate strategy when designing a computation under Edgelet computing. Finally, we present qualitative and quantitative evaluations of the proposed methods and strategies to validate them and study their limits.

To the best of our knowledge, this work represents the first attempt to combine TEEs and Opportunistic Networks to introduce fully decentralized privacy-preserving computations. Our initial findings [37], alongside our demonstrations illustrating the concrete use case of DomYcile [39] and the versatility of the framework [38] (with various weakly connected TEE devices), give confidence in the relevance and feasibility of Edgelet Computing. However, several new research challenges deserve to be studied by the scientific community to enhance the generality of the approach, among others:

Optimization of resource consumption. The first challenge is to explore the multiple optimization tradeoffs that exist between data confidentiality, query success rate, and (local and global) resource consumption. As explained in subsection 2.1 , we chose an epidemic diffusion of message in the Opportunistic Network. This simplistic routing protocol maximizes the message delivery rate and minimizes message latency, at the cost of significant network congestion and overhead. A significant improvement would be to incorporate individuals' social patterns into routing (e.g., BUBBLE Rap [36]) and facilitate query processing by assigning operators to the most connected devices (e.g., doctors, teachers). Such optimizations would lead to revisiting the enforcement of the security property, in particular the random assignment protocol to allow a (limited and controlled) degree of bias in favor of socially well-connected crowd members.

Management of long-lasting snapshots. A second challenge is to integrate long-lasting snapshots (i.e., persistent datasets) to support processes routinely used in data analysis. Such processes start with an initial set of exploration queries to capture data frequency distributions before running precise database queries, data mining, or machine learning algorithms. Long-lasting snapshots could resort to specific indexing schemes to re-access sets of participating edgelets or materialized snapshot partitions kept (encrypted) on sets of edgelets. Obviously, persistent data management would have various impacts on the properties of security, resiliency, and validity. How to prevent data at rest from being the target of malicious attacks? How to preserve the consistency of successive queries in a fully decentralized environment?

Enhancing the validity of iterative algorithms. A third challenge is to study classes of iterative algorithms compatible with Edgelet computing for which it is possible to prove the strict validity of the results. For example, as mentioned in subsection 5.2 , we observe that the Apriori algorithm executed in Edgelet can produce exactly the same results as a centralized execution. To go further and deepen our first experimental evaluations, it would be essential to theoretically prove the convergence of the algorithms. The research work of Lian et al. [43] on the convergence of the Asynchronous Decentralized Parallel Stochastic Gradient Descent particularly illustrates the interest of this type of proof. Thus, in the same direction, a theoretical and systematic analysis of iterative processing in Edgelet would be a valuable asset for deploying new applications.

9 Acknowledgement

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

10 References

[1] Agarwal, N. et al. 2018. cpSGD: Communication-efficient and differentially-private distributed SGD. *NeurIPS 2018, Montréal, Canada*, 7575–7586.

[2] Agrawal, D. et al. 2009. Database Management as a Service: Challenges and Opportunities. *ICDE 2009, Shanghai, China*, 1709–1716.

[3] Alvim, M.S. et al. 2018. Local Differential Privacy on Metric Spaces: Optimizing the Trade-Off with Utility. *CSF 2018, Oxford, United Kingdom*, 262–267.

[4] Alwan, H. and Agarwal, A. 2009. A Survey on Fault Tolerant Routing Techniques in Wireless Sensor Networks. *2009 Third International Conference on Sensor Technologies and Applications, Athens, Greece*, 366–371.

[5] Anciaux, N. et al. 2019. Personal Data Management Systems: The security and functionality standpoint. *Information Systems*. 80, 13–35.

[6] Androutsellis-Theotokis, S. and Spinellis, D. 2004. A survey of peer-to-peer content distribution technologies. *ACM Comp. Surveys*. 36, 4, 335–371.

[7] Bater, J. et al. 2017. SMCQL: Secure Query Processing for Private Data Networks. *Proc. VLDB Endow.* 10, 6, 673–684.

[8] Bellet, A. et al. 2018. Personalized and Private Peer-to-Peer Machine Learning. *AISTATS 2018, Spain*, 473–481.

[9] Benchi, A. et al. 2015. Solving Consensus in Opportunistic Networks. *ICDCN 2015, Goa, India*, 1–10.

[10] Berger, C. et al. 2022. A Survey on Resilience in the IoT: Taxonomy, Classification, and Discussion of Resilience Mechanisms. *ACM Comput. Surveys.* 54, 7, 1–39.
[11] BOINC: *https://boinc.berkeley.edu*.
[12] Bonawitz, K. et al. 2017. Practical Secure Aggregation for Privacy-Preserving Machine Learning. *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, *Dallas, Texas, USA*, 1175–1191.
[13] Bonawitz, K.A. et al. 2019. Towards Federated Learning at Scale: System Design. *MLSys 2019, Stanford, CA, USA*.
[14] Boneh, D. et al. 2013. Private Database Queries Using Somewhat Homomorphic Encryption. *ACNS 2013, Banff, AB, Canada*, 102–118.
[15] Bottou, L. 2010. Large-Scale Machine Learning with Stochastic Gradient Descent. *19th International Conference on Computational Statistics, COMPSTAT 2010, Paris, France*, 177–186.
[16] Brijs, T. 2003. Retail market basket data set. *Workshop on Frequent Itemset Mining Implementations (FIMI'03)*.
[17] Cisco Annual Internet Report (2018–2023) White Paper: 2020. *https://tinyurl.com/cisco-internet-report*.
[18] Conti, M. et al. 2010. From opportunistic networks to opportunistic computing. *IEEE Communications Magazine*. 48, 9, 126–139.
[19] Conti, M. et al. 2017. The Internet of People (IoP): A new wave in pervasive mobile computing. *Pervasive Mob. Comput.* 41, 1–27.
[20] Cormode, G. et al. 2019. Answering Range Queries Under Local Differential Privacy. *Proc. VLDB Endow.* 12, 10, 1126–1138.
[21] Corrigan-Gibbs, H. and Boneh, D. 2017. Prio: Private, Robust, and Scalable Computation of Aggregate Statistics. *NSDI 2017, Boston, MA, USA*, 259–282.
[22] Costan, V. and Devadas, S. 2016. Intel SGX Explained. *IACR Cryptol. ePrint Arch.*, 86.
[23] Dhillon, I.S. and Modha, D.S. 1999. A Data-Clustering Algorithm on Distributed Memory Multiprocessors. *SIGKDD, San Diego, CA, USA*, 245–260.
[24] DomYcile Project: https://tinyurl.com/domycile, Salon E-Tonomy: https://tinyurl.com/e-tonomy.
[25] Dong, Y. et al. 2021. FLOD: Oblivious Defender for Private Byzantine-Robust Federated Learning with Dishonest-Majority. *ESORICS 2021, Darmstadt, Germany*, 497–518.
[26] Estellés-Arolas, E. and González-Ladrón-de-Guevara, F. 2012. Towards an integrated crowdsourcing definition. *Journal of Information Science*. 38, 2, 189–200.
[27] Fereidooni, H. et al. 2021. SAFELearn: Secure Aggregation for private FEderated Learning. *SP Workshops 2021, San Francisco, CA, USA*, 56–62.
[28] Ferrer, A.J. et al. 2019. Towards the Decentralised Cloud: Survey on Approaches and Challenges for Mobile, Ad hoc, and Edge Computing. *ACM Comput. Surveys.* 51, 6, 1–36.
[29] Froelicher, D. et al. 2020. Drynx: Decentralized, Secure, Verifiable System for Statistical Queries and Machine Learning on Distributed Datasets. *IEEE Trans. Inf. Forensics Secur.* 15, 3035–3050.
[30] Ganti, R.K. et al. 2011. Mobile crowdsensing: current state and future challenges. *IEEE Commun. Mag*. 49, 11, 32–39.
[31] González-Manzano, L. et al. 2016. PAgIoT – Privacy-preserving Aggregation protocol for Internet of Things. *Journal of Network and Computer Applications*. 71, 59–71.
[32] Guo, B. et al. 2015. Mobile Crowd Sensing and Computing: The Review of an Emerging Human-Powered Sensing Paradigm. *ACM Comp. Surveys*. 48, 1, 1–31.
[33] Gupta, P. et al. 2019. Obscure: information-theoretic oblivious and verifiable aggregation queries. *Proc. VLDB Endow*. 12, 9, 1030–1043.
[34] Herr, M. et al. 2017. Frailty, polypharmacy, and potentially inappropriate medications in old people: findings in a representative sample of the French population. *European Journal of Clinical Pharmacology*. 73, 9, 1165–1172.
[35] Huba, D. et al. 2022. PAPAYA: Practical, Private, and Scalable Federated Learning. *MLSys 2022, Santa Clara, CA, USA*.
[36] Hui, P. et al. 2011. BUBBLE Rap: Social-Based Forwarding in Delay-Tolerant Networks. *IEEE Trans. Mob. Comput.* 10, 11, 1576–1589.
[37] Javet, L. et al. 2022. Edgelet Computing: Pushing Query Processing and Liability at the Extreme Edge of the Network. *CCGrid 2022, Taormina, Italy*, 160–169.
[38] Javet, L. et al. 2023. Pushing Edge Computing one Step Further: Resilient and Privacy-Preserving Processing on Personal Devices. *EDBT 2023, Ioannina, Greece*, 835–838.
[39] Javet, L. et al. 2023. Secure Computations in Opportunistic Networks: An Edgelet Demonstration with a Medical Use-Case. *PerCom Workshops 2023, Atlanta, GA, USA*, 331–333.
[40] Kasiviswanathan, S.P. et al. 2011. What Can We Learn Privately? *SIAM J. Comput.* 40, 3, 793–826.
[41] Keränen, A. et al. 2009. The ONE Simulator for DTN Protocol Evaluation. *SIMUTools 2009, Rome, Italy*.
[42] Ladjel, R. et al. 2019. Trustworthy Distributed Computations on Personal Data Using Trusted Execution Environments. *TrustCom 2019, Rotorua, New Zealand*, 381–388.
[43] Lian, X. et al. 2018. Asynchronous Decentralized Parallel Stochastic Gradient Descent. *ICML 2018, Stockholm, Sweden*, 3049–3058.
[44] Mansouri, M. et al. 2023. SoK: Secure Aggregation Based on Cryptographic Schemes for Federated Learning. *Proc. on Privacy Enhancing Technologies*. 1, 140–157.
[45] Ménétrey, J. et al. 2022. An Exploratory Study of Attestation Mechanisms for Trusted Execution Environments. arXiv.
[46] Mirval, J. et al. 2023. Federated Learning on Personal Data Management Systems: Decentralized and Reliable Secure Aggregation Protocols. *SSDBM 2023, Los Angeles, CA, USA*, 1-12.
[47] Mo, Y. and Murray, R.M. 2017. Privacy Preserving Average Consensus. *IEEE Trans. Autom. Control.* 62, 2, 753–765.
[48] Murray, D.G. et al. 2010. The case for crowd computing. *ACM SIGCOMM 2010, New Delhi India*, 39–44.
[49] Nawab, F. and Shadmon, M. 2024. The Tipping Point ofEdge-Cloud Data Management. *CIDR 2024, Chaminade, CA, USA*.
[50] Nguyen, J. et al. 2022. Federated Learning with Buffered Asynchronous Aggregation. *AISTATS 2022, Virtual Event*, 3581–3607.

[51] O'Keeffe, D. et al. 2018. Frontier: Resilient Edge Processing for the Internet of Things. *Proc. VLDB Endow.* 11, 10, 1178–1191.
[52] OpenML Adult dataset: *http://www.openml.org/d/1590*.
[53] Phong, L.T. et al. 2018. Privacy-Preserving Deep Learning via Additively Homomorphic Encryption. *IEEE Trans. Inf. Forensics Secur.* 13, 5, 1333–1345.
[54] Pinto, S. and Santos, N. 2019. Demystifying Arm TrustZone: A Comprehensive Survey. *ACM Comp. Surveys*. 51, 6, 1–36.
[55] Proposal for a Proposal for a Regulation of the European Parliament and of the Council on European data governance: *https://tinyurl.com/data-governance-act*.
[56] Sabater, C. et al. 2022. An accurate, scalable and verifiable protocol for federated differentially private averaging. *Mach. Learn.* 111, 11, 4249–4293.
[57] Sabt, M. et al. 2015. Trusted Execution Environment: What It is, and What It is Not. *TrustCom 2015*, 57–64.
[58] Satyanarayanan, M. et al. 2009. The Case for VM-Based Cloudlets in Mobile Computing. *IEEE Pervasive Comput.* 8, 4, 14–23.
[59] Savasere, A. et al. 1995. An Efficient Algorithm for Mining Association Rules in Large Databases. *PVLDB'95, Zurich, Switzerland*, 432–444.
[60] Scikit-learn California dataset: *https://tinyurl.com/california-housing-dataset*.
[61] Shared Responsibility Model - Amazon Web Services: *https://aws.amazon.com/compliance/shared-responsibility-model/*.
[62] Shi, W. et al. 2016. Edge Computing: Vision and Challenges. *IEEE Internet of Things Journal*. 3, 5, 637–646.
[63] So, J. et al. 2022. LightSecAgg: a Lightweight and Versatile Design for Secure Aggregation in Federated Learning. *Proceedings of Machine Learning and Systems 2022, MLSys 2022, Santa Clara, CA, USA*.
[64] The Survey of Health, Ageing and Retirement in Europe (SHARE): *http://www.share-project.org/*.
[65] Tramer, F. et al. 2017. Sealed-Glass Proofs: Using Transparent Enclaves to Prove and Sell Knowledge. *EuroS&P 2017*, 19–34.
[66] Triastcyn, A. and Faltings, B. 2019. Federated Learning with Bayesian Differential Privacy. *IEEE BigData 2019, Los Angeles, CA, USA*, 2587–2596.
[67] Trifunovic, S. et al. 2017. A Decade of Research in Opportunistic Networks: Challenges, Relevance, and Future Directions. *IEEE Commun. Mag.* 55, 1 (2017), 168–173.
[68] Vahdat, A. and Becker, D. 2000. Epidemic Routing for Partially-Connected Ad Hoc Networks.
[69] Wiesmann, M. et al. 2000. Understanding replication in databases and distributed systems. *Proceedings 20th IEEE International Conference on Distributed Computing Systems, Taipei, Taiwan*, 464–474.
[70] Xu, D. et al. 2021. Edge Intelligence: Empowering Intelligence to the Edge of Network. *Proceedings of the IEEE*. 109, 11, 1778–1837.
[71] Xu, Y. et al. 2023. Towards machine-learning-driven effective mashup recommendations from big data in mobile networks and the Internet-of-Things. *Digit. Commun. Networks*. 9, 1, 138–145.
[72] Yang, G. et al. 2021. Federated Learning with Personalized Local Differential Privacy. *ICCCS 2021, Chengdu, China*, 484–489.
[73] Yick, J. et al. 2008. Wireless sensor network survey. *Computer Networks*. 52, 12, 2292–2330.
[74] Zhang, Z. et al. 2018. CALM: Consistent Adaptive Local Marginal for Marginal Release under Local Differential Privacy. *CCS 2018, Toronto, ON, Canada*, 212–229.

Appendix 1: Query Deadline Calibration

The backup-based strategy presented in subsection 4.1 requires calibrating the query deadline according to the query plan to be executed, to the expected probability of success p_s for this query, and to the maximum transmission delay δ for a sent message. The intermediate timeouts for each query plan node can be obtained using a simple recursive calculation based on the following principles:

- For each data processor, a timeout Δ is associated with each of its predecessor nodes, whether primary or backup. When this timeout Δ expires, a fault of the corresponding predecessor node is presumed and an activation message is sent to the appropriate backup node (the first backup of a primary if the primary is presumed faulty, the second backup if the first backup is presumed faulty, etc.). After the activation message has been sent, only the first message received from either the presumed faulty primary or any of its activated backups will be considered and processed.
- Primary and active backup nodes follow a push message pattern for all their successors (whether primary or backup nodes), meaning that messages are sent to successor nodes on their own initiative. Conversely, passive backup nodes follow a pull message pattern, i.e., messages are sent after a successor node's activation message is received.

Level 0: Snapshot Builder and its backups. Since the Data Contributors are never recalled, there is no timeout defined at the Snapshot Builders level and their backups.

Level 1: Computer and its backups. The Computer nodes (primary and backups) should receive their input data from the Snapshot Builder primary node after a maximum time delay of ~2δ. Indeed, the Snapshot Builders and their backups are supposed to receive the messages from the Data Contributors after a delay δ. Assuming that the data processing time of a node is negligible compared to the message latency, an additional maximum delay of δ is needed for the Snapshot Builder to transmit

its result to the Computers. A first intermediate timeout at the Computer is hence fixed at $\Delta_1=2\delta$, at which the first backup of the Snapshot Builder will be activated. Recursively, as we expect the result of this first backup to be received after a maximum additional 2δ delay, we can define the second intermediate timeout (activating the second backup) at $\Delta_2=\Delta_1+2\delta$, and so on for the next backups.

We can generalize this to obtain the timeout Δ_b for any passive backup node in the tree. Indeed, this timeout is determined by adding 2δ (activation and response delays) to the Maximum Execution Time of a given Backup node (*METB*). At tree level l+1, *METB* is equal to the sum of:

- The consecutive activation time of all direct predecessor backup nodes and their corresponding response time: $b \times 2 \times \delta$, with b the number of backup nodes.
- The recursive runtime of these backup nodes if they are not yet provisioned. This value depends directly on the number of levels l in the sub-QEP (i.e., the tree height): $b \times METB(l)$

Thus, $METB(l+1) = b \times \big(2 \times \delta + METB(l)\big)$

Similarly, the Maximum Execution Time of a Primary node (*METP*) at tree level l+1 is equal to the sum of:

- The emission time of its direct predecessor primary node and its recursive runtime: $\delta + METP(l)$
- The successive activation time of the direct predecessor backup nodes and their corresponding response time: $b \times 2 \times \delta$
- The recursive runtime of these backup nodes if they are not yet provisioned: $b \times METB(l)$

$$METP(l+1) = \delta + METP(l) + b \times \big(2 \times \delta + METB(l)\big)$$

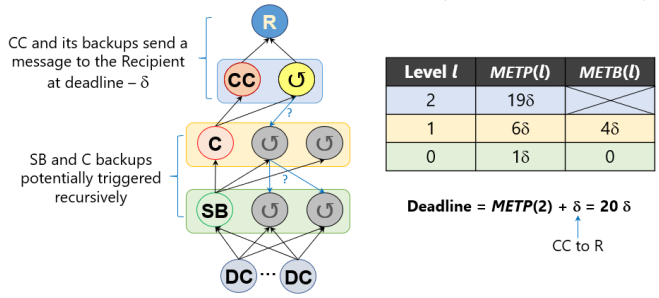

Figure 21: Example of Deadline Calibration

Figure 21 shows an example of the query deadline calibration with the QEP of Figure 6, setting b=2 for the passive backups of the Snapshot Builders and Computers and b_{cc}=1 for the active backups of the Computing Combiner. We can see that at level 0, the SB has a *MEPT* equal to δ, which corresponds to the emission time of the DCs' contributions, while the SB's backups have a *METB* equal to 0. Indeed, the execution time of the backups is only counted from their activation time, since the data replication is done in parallel with the primary node and the emission time is therefore already integrated. From this figure, we can see that the calculation of the deadline is strongly influenced by the height of the tree and the number of backups. Note that the horizontal and vertical partitioning have no impact on the deadline calibration as long as they do not change the height of the QEP and the number of backups per partition.

Conflicts of Interest:
On behalf of all authors, the corresponding author states that there is no conflict of interest.

Data Availability Statement
The datasets generated during and/or analyzed during the current study are available from the corresponding author upon reasonable request.