Edgelet Computing: Pushing Query Processing and Liability at the Extreme Edge of the Network

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*—We call edgelet computing the current convergence between Opportunistic Network (OppNet) and Trusted Execution Environment (TEE) at the very edge of the network. We believe that this convergence bears the seeds of a novel and important class of applications leveraging fully decentralized and highly secure computations among data scattered on multiple personal devices. This paper introduces the Edgelet computing paradigm, defines properties that guarantee the safety, liveness and security of executions in this unusual context and proposes alternative strategies satisfying these properties. Preliminary performance evaluations and an ongoing real-case study highlights the practicality of the approach. Finally, the paper draws future research challenges for the database and distributed system community.**

Keywords—Edge computing; TEE; OppNet; Resiliency

I. INTRODUCTION

The edge of the Internet is expanding at a much higher pace than its core, with 29.3 billion of connected device by 2023 [1]. At the same time, the reliable, server-based and infrastructure-centric approach of the internet already exhibits its limits in terms of efficiency, privacy and energy consumption. Alternatively, infrastructureless and self-organizing networking protocols have emerged, from the legacy MANET protocol to the Opportunistic Network (OppNet) paradigm, on top of which a new people-centric vision of the Internet (IoP) can be drawn [2]. However, while this paradigm has attracted a lot of attention from the research community, OppNets have never been deployed at large scale, mainly due to the lack of killer applications [3].

A game changer is the generalization of Trusted Execution Environments (TEE) [4] at the extreme edge of the network: Intel SGX [5] is becoming ubiquitous on PC and tablets, ARM's TrustZone [6] on smartphones (Fig. 1.a) and even Trusted Platform Module on smart objects (Fig. 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 – way than with homomorphic cryptography [7], differential privacy [8] or secure multiparty computation protocols [9]. Our approach leverages the security features of TEEs to enable generic computation and scalable execution, processing cleartext data without any tradeoff between privacy and accuracy.

The convergence between OppNets and TEEs, named hereafter *Edgelet computing*, leverages secure personal devices (called edgelets) and holds the promise of concrete killer applications for OppNets by exploiting three non-exclusive dimensions:

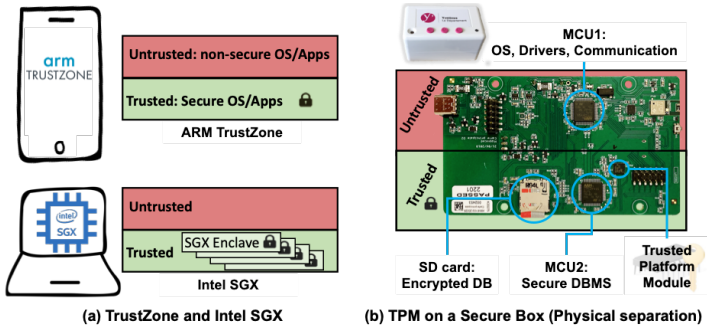

Fig. 1. Examples of Trusted Execution Environments. On the right, the Trusted Platform Module ensures the authenticity of the MCU2 Trusted Computing Base and keeps safely the encryption keys of the database (DB).

– *Data altruism*: This concept introduced in the EU Data Governance Act [10] fosters data subjects to give consent to process their personal data for purposes such as scientific research or public services improvement. Privacy protection is paramount in this context. Notably, a real-case study [11] aiming at querying ephemeral cohorts over 8,000 elderly patients falls in this category and has motivated part of this work. In this study, detailed next, patients hold their medical records in a secure personal box at home (Fig. 1.b) acting as an edgelet using short range communications. Such scenario could be generalized to all forms of Personal Data Management Systems (PDMS) [12] hosted by their holder.

– *Crowd-liable data processing*: Companies and administrations are often interested in analyzing personal data from their customers or citizens, but are reluctant to shoulder the responsibility for treatment due to high legal risk in case of data leakage and to the image deficit incurred by adopting solutions perceived as intrusive. Edgelet computing provides them a mean to convince the targeted users – by any form of reward – to voluntary contribute to a global processing task while letting the computing infrastructure and personal data in their hands. This liability shift from the data controller (in the GDPR sense) to the crowd is made possible by leveraging the security of Edgelets' TEE.

– *Contextual data processing*: Some data processing tasks are relevant only at specific times and/or geographic areas, and of interest only for specific groups of users (e.g., tourists visiting a city, customers in a mall). This motivated the emergence of Mobile Social Networks (MSN) [13]. Edgelet computing provides the required trust among MSN participants to dynamically organize data processing tasks, taking advantage of location-based homophily that exists among people having social relationships [14].

Edgelet computing scenarios require performing complex processing over personal data – from database queries to machine learning – in a highly distributed, failure-prone and infrastructureless context, with strong security guarantees. This paper is a first attempt to tackle the data management and distributed system issues related to this environment and makes the following contributions:

(1) it characterizes the Edgelet computing paradigm by a combination of architectural and computing assumptions, an unusual threat model and three properties that guarantee the liveness, safety and security of executions;

(2) it proposes two alternative execution strategies enforcing liveness in opposite ways and discuss their impact on safety;

(3) it provides quantitative and qualitative evaluations of these strategies and derives design rules for adapting centralized computations to the Edgelet computing paradigm.

Section II presents the Edgelet Computing paradigm and precisely states the problem at hand. Section III introduces the basics (execution plans and task-to-edgelet assignment) to handle distributed computations in this paradigm. Section IV and V detail the two execution strategies mentioned above. Section VI provides a preliminary performance evaluation while Section VII presents a qualitative evaluation and design rules. Section VIII focuses on related works. Finally, we conclude and present important research challenges for the database and distributed system community in Section IX.

II. Edgelet Computing Paradigm

A. Architecture and Computation of Interest

As mentioned above, Edgelet Computing refers to the convergence between OppNets and TEEs at the extreme edge of the network, namely up to personal devices and smart objects sidestepping the classical communication infrastructure for cost or energy constraints, lack of connectivity, security or even freedom of expression concerns. Hence, we consider that the communications are short range (e.g., Bluetooth, Wi-Fi) and asynchronous, i.e., there is no bound on the message transmission delay. Connections among devices then form a non-connected time-varying graph as in traditional OppNets [15]. For simplicity, we consider an epidemic diffusion of the messages, i.e., messages are transferred from device to device following a store-carry-forward strategy. We let optimized routing protocols exploiting moving patterns of users for future work.

We assume that any edgelet is equipped with a TEE which guarantees (1) data confidentiality: data manipulated by a TEE cannot be observed from the outside; and (2) code integrity: an attacker cannot influence the behavior of a program executing within a TEE. Although highly difficult to conduct and requiring physical instrumentation, side-channel attacks compromising data confidentiality cannot however be totally ignored [16].

Finally, we consider computations involving personal data hosted in distributed edgelets, constituting a multitude of disconnected instances of PDMS [12] or Databoxes [17]. Contrary to participatory sensing or sensor networks which focus on stream queries over elementary data, we consider rich data (e.g., healthcare folders, spending habits) and complex processing (e.g., machine learning, data mining) over edgelets data seen as a horizontal partitioning of a shared database. In this context, the computations (called Query hereafter) must cope with the uncertainty inherent to the Edgelet paradigm, making the traditional database closed-world assumption irrelevant. Let E be 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 C (e.g., $C = 2000$). We denote by $\zeta_Q(E)$ the set of all snapshots of E enforcing P and C. The Query Q is thus said *snapshot compliant* if the result of Q considering any snapshot of $\zeta_Q(E)$ is semantically equivalent for the Querier.

B. Edgelet Computing Threat Model

Edgelet computing requires the introduction of a dedicated threat model to capture (1) the shift of responsibility from the data controller to the crowd and (2) the TEE trustworthiness.

Indigent Querier.
Trust: we do not question the good faith of the Querier but rather its ability or willingness to set up and endorse a secure infrastructure. In case of an attack, the Querier may have punctually a Malicious behavior, but recurring inference attacks from its side are precluded and not further considered.
Role: publish the processing for approval, initiate the processing and get the final result without taking part to the computation.
Examples: a public or private agency, the moderator of a Mobile Social Network.

Wolf in sheepfold Participants.
Trust: all participant's devices are equipped with TEE, but as said above, side-channel attacks cannot be totally precluded despite their complexity. A compromised TEE behaves in a "sealed glass proof" mode [18], where code integrity is preserved but data confidentiality is lost. We assume a large majority of honest participants (the lambs) and a few "sealed glass proof" ones (the wolves).
Role: contribute with their data (Data Contributor) and/or their processing power (Data Processor).
Example: a smartphone or home box secured with a TEE.

Trusted Regulatory Agent.
Trust: full.
Role: review and approve (i.e., sign) the code published by the Querier. This role is not devoted to the query execution itself.
Example: a privacy regulatory agency.

Untrusted environment. covers the rest of the infrastructure, device's OS and network notably.

C. Motivating Example

Fig. 2 illustrates Edgelet computing with a real example taken from the DomYcile project [11], where 8,000 elderly patients are equipped with secure boxes (edgelets) storing their medical/social folder. These boxes are not connected to the Internet for subscription cost, security and acceptability reasons and can only be accessed at patient's home by healthcare workers. The Yvelines district (Querier), in charge of health and social cares, wants to query ephemeral cohorts of consenting people to obtain statistical results in the spirit of [19] (Share EU project [20]), without endorsing the responsibility of this processing. The query example given in Fig. 2 illustrates the necessity to compute all statistics on the same snapshot for

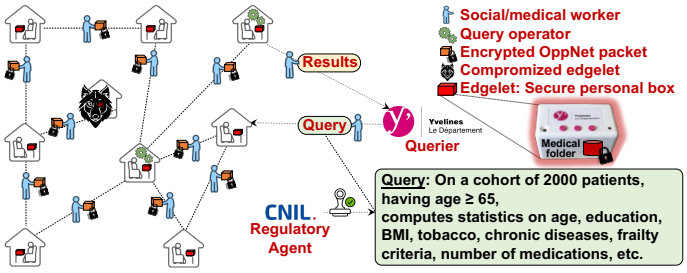

Fig. 2. Motivating example : Edgelet query in the DomYcile project

consistency reasons. The query is approved by the CNIL (French regulatory agency) then broadcast by healthcare workers who implement the OppNet using a store-carry-forward strategy with encrypted messages (untrusted environment) between Edgelets. Secure boxes act as Data Contributor edgelets; some are randomly selected to act as Data Processors, this randomness limiting the action of compromised edgelets.

D. Problem Statement

Edgelet computing opens exciting opportunities to organize secure decentralized computations but combines unusual hypotheses. First, reliable failure detectors cannot exist in OppNets due to the unpredictability of message delays, making difficult to predict the time to build a snapshot from random contributors and to execute a query. The system liveness must then be guaranteed based on fault presumptions only and on probability of success for queries associated to a deadline. Second, the snapshot consistency must be preserved all along the query processing despite presumed faults and message loss between Data Processors to guarantee the system safety, i.e., a consistent result. Third, the liability shift to the crowd must be endorsed in a context where a few "sealed glass proof" devices may endanger the security of the whole system with no way to detect them. We introduce below three properties that must be met together to tackle this problem.

Resiliency (liveness 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 complete before the deadline with a probability greater than p_s, otherwise Q is aborted.

Validity (safety property). The result of an edgelet execution of a query Q must be identical to a centralized execution of Q over at least one snapshot of $\zeta_Q(E)$, a sufficient condition to guarantee the equivalence of the results from the Querier perspective.

Crowd liability (security property). (1) The liability of data processing must be equally distributed among the edgelets by a random process auditable by any participant. (2) Any edgelet involved as a Data Processor is liable only for the processing and data assigned to it. **Note**. Condition (1) precludes targeted attacks by a compromised edgelet which could try to self-assign some operators. Condition (2) guarantees that a data leakage must be circumscribed to the data processed by a compromised edgelet, with no possibility of inducing data leakage in honest edgelets, i.e., only data from compromised edgelets can leak through the network. This guarantee is the strongest possible in the Edgelet context where the TEE security is leveraged to perform processing on clear-text data, allowing any type of processing.

These properties are particularly challenging to tackle together given their mutual impact. We present in Sections IV and V two execution strategies answering this challenge in two opposite ways, both sharing basic principles presented in the next Section.

III. Execution Plan and Data Processor Assignment

Before discussing how computations (i.e., query executions) can be made resilient and how the validity of their result can be assessed, we need first to introduce which form takes these computations and how the tasks involved in these computations are assigned to the participating edgelets complying with the Crowd liability property.

A. General Form of Query Plans

We consider that a computation is expressed by a Query Execution Plan (QEP), that 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. Operators are assigned randomly to Data Processors (see Section III.B) taking in charge the execution of the operator's code. To simplify the presentation, we consider that any message is sent atomically through the OppNet (i.e., the payload is either totally received by the recipient or not at all) and that each edgelet executes a single operator.

The simplest form of a QEP is a tree with *Data Contributors* at the leaves (one per edgelet contributing to the query with its data) sending the requested data to a *Snapshot Builder* operator, the role of which is to collect a representative snapshot of cardinality C, satisfying P. The Snapshot Builder then sends the representative snapshot to a *Computer* operator which in turn computes the final result and finally delivers it to the Querier.

The Computer can be decomposed into sub-operators assigned to different edgelets for privacy or performance concerns. This can help minimizing 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 information that become sensitive when combined (e.g., a quasi-identifier) by vertically partitioning the dataset. This can finally help minimizing the Data Processor workload (e.g., when energy consumption matters) or exploit the inherent Edgelet computing parallelism. A *Computing Combiner* operator must then be added in the QEP to combine the outputs of all sub-operators.

In this section, we impose no restriction on the operators to be computed nor on the form of the QEP: we consider them as given by the Querier. Fig. 3 presents the QEP for the query example of Fig. 2, using horizontal partitioning (Data Contributors are assigned to Snapshot Builders, e.g., by hashing their public key) and vertical partitioning (each Computer manages a single statistic).

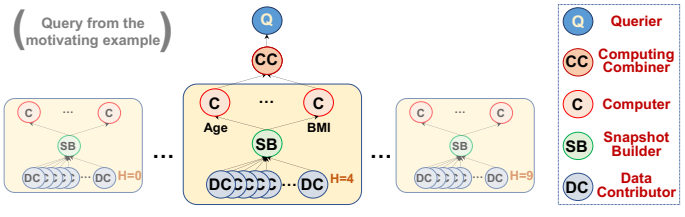

Fig. 3. Vertically & horizontally partitionned QEP

B. Data Processor Assignment

The assignment of tasks to Data Processors must be done with care to avoid violating the Crowd liability property. Condition (1) of this property imposes a random assignment of Edgelets to QEP Operators. This is to guarantee that a compromised Data Processor cannot freely choose to execute e.g., the operator manipulating the data of a targeted contributor or the operator manipulating a maximum amount of sensitive data. This process must also be made auditable by any participant to establish trust among the crowd.

The assignment process we propose works as follows. It assumes that the set of edgelets in E are known by the Querier (e.g., they register to join a community) and it organizes the ID of their edgelets in a hash ring (as in a Chord DHT). It computes a hash of the query (signed by the Trusted Regulatory Agent and publicly known) as a seed for the random process and assigns the first operator to the edgelet having the ID immediately greater than this hash. The first hash is rehashed to assign the second operator and so forth until all operators have been assigned. Note that this is consistent with the Indigent Querier threat model, in that any participant can reproduce this chain of hash and can thus detect a fraudulent assignment for the operator intended for them, assuming they know at least the ID of their predecessor in the ring.

Let's now examine the impact of Condition (2) of the Crowd liability property. It expresses two things. First, it states the absence of data leakage if no Data Processor is compromised. It is trivially guaranteed on the one hand by the TEE confidentiality property and on the other hand by the encryption of all messages in transit, the latter being simplified by the assignment protocol presented above (i.e., any edgelet can encrypt its output according to the targeted recipients since they are statically defined in the QEP). Second, this condition states that a data leakage must be restricted to the data handled by compromised Data Processors. This is achieved by the fact that (1) no cryptographic material (e.g., keys) is ever shared among edgelets and (2) TEE code integrity still holds even in sealed-glass proof mode, thus reducing the potential leakage to the data processed by a compromised edgelet.

IV. Backup-Based Execution Strategy

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

A. Enforcing Resiliency

As already stated, no reliable failure detector exists in our context and every Data Processor (SB, C and CC edgelets in Fig. 3) is a potential Single Point of Failure (SPF). In this approach, we simply try to recover from failures, whatever the Data Processor presumed faulty, the benefit of which being to make the handling of resiliency independent of the form of the QEP. Therefore, we use timeouts to presume faults and secure the execution of all SPFs by means of backups, as usual [21].

We distinguish *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

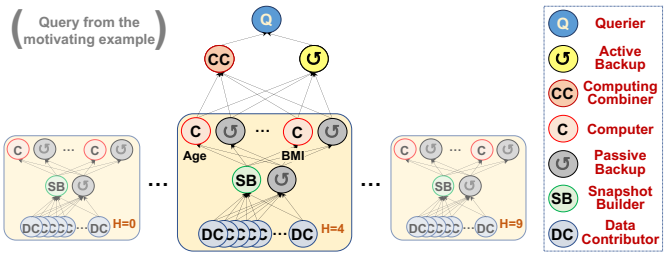

Fig. 4. Active and Passive Backups

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 higher data exposure (see Fig.4). Consequently, all SPFs are passively replicated, except the Computing Combiner which must be actively replicated; otherwise, the Querier would be forced to take part in the processing, at least to activate the Computing Combiner Backup; this would have an impact on data exposure and would hurt the Indigent Querier assumption.

The number b of backups per primary is determined by the inequality $(1 - {p_f}^{b+1})^{|SPF|} \geq p_s$, with p_f the probability of fault presumption, p_s the expected probability of success, and $|SPF|$ the number of SPFs in each horizontal partition of the QEP. Given a query deadline, the timeouts are chosen so that any Data Processor has the ability to activate all of its children's backups if needed. This simple recursive calculation is not detailed here for brevity.

B. Impact on Validity

Satisfying the Validity property requires that (i) the operators involved in the QEP are genuine and correctly executed in the right order and (ii) the dataflow between these operators is consistent wrt. at least one of the $\zeta_Q(E)$ snapshots.

We leverage the code integrity property provided by the TEE to enforce condition (i) as follows. Each edgelet runs a piece of code called *Core* hereafter, which is part of the TEE Trusted Computing Base, that is a code base guaranteed genuine at boot time. In turn, the Core attests the genuineness of the QEP operator's code assigned to the edgelet and guarantees the integrity of the intermediate results in cascade [22] in the spirit of remote attestations [5]. Moreover, the QEP is built statically (as said above), thus all communications are predetermined and can easily be cryptographically secured. Hence, any change in the operators ordering would make the messages indecipherable and the execution would fail.

In a perfect world, condition (ii) directly stems from condition (i). With failure or message loss however, a Snapshot Builder and its backup(s) may build different snapshots, each belonging to $\zeta_Q(E)$. Due to this, 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 of $\zeta_Q(E)$, called hereafter the *reference snapshot*.

1. Without 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 backup otherwise.

2. Similarly, with horizontal partitioning, the reference snapshot is simply the union of each partition's snapshot (either backup or primary).
3. With vertical partitioning however, the Snapshot Builder feeds several Computers, some of them potentially considering the primary snapshot and some others the backup ones in case of fault presumption. This leads to an inconsistency, i.e., a result built over a snapshot that does not correspond to any snapshot belonging to $\zeta_Q(E)$.

Fig. 4 illustrates this concern, where each Computer evaluates a different statistic but must consider a 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 cite only two of them for conciseness: (1) Synchronize the snapshot between the primary Snapshot Builder and its backup(s) thanks to a consensus protocol. [15] proposes an effective consensus protocol for OppNets that matches the Edgelet context, at the price of a distributed consensus; (2) Rearrange the QEP so that the parallel branches are serialized, one after the other, at the price of losing the QEP parallelism.

V. Overcollection-Based Execution Strategy

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

A. Enforcing Overcollection-based 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 SPF that executes a distributive operator. Instead of executing this operator on a single edgelet, we distribute (using hashing) its execution over n edgelets, each processing a partition of the initial dataset (in Figs. 3, 4 and 5, n=10). 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. Hence, the probability of failure of a SPF over-collecting m more partitions is $\sum_{i=m+1}^{n+m}\binom{n+m}{i}{p_f}^i(1-p_f)^{(n+m-i)}$. Note that we could also have increased the number of collected data per partition, keeping n unchanged, but preliminary experiments showed that this solution is less effective in terms of data exposure. The challenge, detailed next, is to apply this intuition to a complete QEP.

B. Impact of Overcollection on Validity

If all QEPs can satisfy the Validity property when executed in a backup mode (in some cases by adding a consensus among backups), this is no longer true when executed in an 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 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 an independent processing over a partition of the collected dataset with a root process assembling the final result (see Fig. 5). This solution applies only if the commutativity rules between operators allow to push all distributive operators down to the sub-QEPs and to push

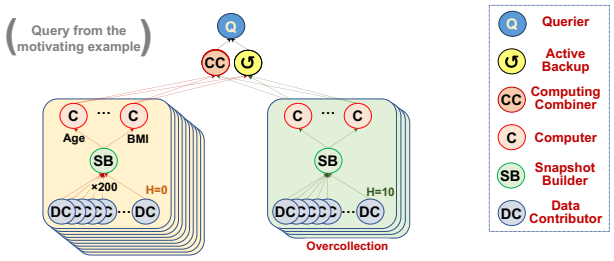

Fig. 5. Overcollection for the QEP of Fig. 3

all non-distributive operators up to the root. A QEP satisfying this condition is said *reshapable*. We consider this brute-force solution next and let more subtle designs for future work. Under this assumption, point (2) can be easily tackled. The reference snapshot D is simply the union of all partitions that contributed to the QEP computation up to its root. Assuming that each of the n+m partitions locally satisfies predicate P and have a cardinality C/n, the execution validity is trivially preserved as long as less than m partitions are lost.

C. Relaxing Validity

Most data intensive queries of interest in our context are distributive by nature (as confirmed by various MapReduce or Spark implementations). 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 (ML) 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 messages 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* and sketched in Fig. 6.

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 knowledges and sends the final result to the Querier.

The main question is when to stop the processing. Fixing a number of iterations a priori (with a minimal number of received messages) has little sense in the OppNet context where message delays, then edgelet progression, are unpredictable. Expecting a local convergence is also hazardous 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

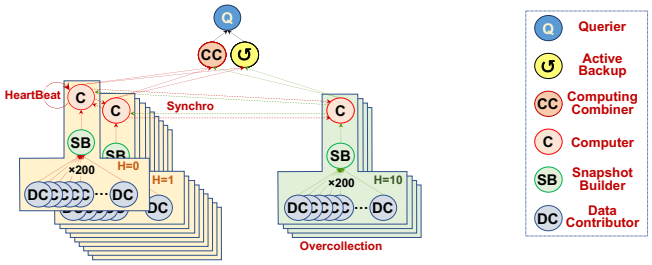

Fig. 6. Overcollected APriori/K-means QEPs

cadenced by a clock, 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, 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 to Querier
```

We illustrate this method on two classical algorithms, namely Apriori (data mining representative) and k-means (ML representative), and show its effectiveness in terms of proximity of results wrt. centralized executions in Section VI.

Apriori [23]: mine 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 [24]: form k clusters minimizing the intra-cluster variance.
- *knowledge*: current centroids (initially, k initial 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 newly created clusters, updating its *knowledge*.
- *Synchronization*: computes, on a cluster basis, the barycenter of all centroids received from other sub-QEPs it has heard of, and integrates the result in *knowledge*.
- *Computing Combiner*: computes, on a cluster basis, the barycenter of all centroids received.

VI. Preliminary Evaluations

The goal of these evaluations is threefold: to validate the relevance of the approach, to calibrate the system and to verify its effectiveness. We first compare the backup-based strategy (**Bak**) and the Overcollection strategy (**Ovr**), providing insights to properly configure the parameters. Then, we evaluate Ovr on a non-iterative query to calibrate the query delay with respect to the probability of success (p_s). Finally, we test the iterative methods Apriori and k-means and evaluate the quality of their results against a centralized execution.

To this end, we built an Edgelet computing simulator on top of the Opportunistic Network Environment (ONE) simulator [25] providing detailed traces of OppNets communications (see Fig. 7). We model two representative use cases with messages exchanged using an epidemic routing strategy [26]:

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

Mall: Edgelet computing within a Mall of 0.16 km^2, where 5,000 edgelets (customer smartphones following a RandomWayPoint movement) are opportunistically executing a query exchanging messages using Bluetooth when meeting. We obtained $\bar{L}$ = 1,936 s and $R\sigma$ = 0.48 (σ = 933 s).

Our simulator can process real data on simulated edgelets, using the latencies computed by ONE. The experiments are done on both use cases. To minimize random variations, we averaged the results of 300 executions. We consider the QEPs of figures 4 (Exp. 1), 5 (Exp. 1 & 2), and 6 (Exp. 3), all being horizontally partitioned on n = 10 partitions (plus potentially over-collected ones) as in the figures. Other experiment parameters are indicated on the graphs.

Experiment 1: Bak vs Ovr. Fig. 8.a and 8.b compare the replication degree (i.e., b for Bak) and the Overcollection degree (i.e., m/n for Ovr), for each SPF (SB or C in Fig. 3 and 4). These values (y-axis) are indicative of the potential exposure in case of compromised Data Processors. We vary the degree of vertical partitioning (v on x-axis), i.e., the number of Computers in each partition, to avoid concomitant exposure of, e.g., a quasi-identifier, and consider large values to check the behavior of Bak and Ovr in extreme cases (partitioning in 2 or 3 subsets is generally sufficient). We calibrate b and m using the formulas of Section IV.A and Section V, and, to simplify the analysis, we assume an infinite deadline, i.e., there is only effective failures. Thus, b and m/n values are identical for DomY or Mall contexts.

Observations: (1) Bak shows steps since b is an integer while Ovr is smoothly increasing (m/n); (2) when v is large (e.g., $v > 6$) and there are many failures (15%), m/n increases considerably for Ovr while b stays reasonable for Bak. Indeed, with Ovr, at least n partitions with all SPFs must "survive" failures while Bak needs at least one survival primary or backup Data Processor per SPF in each partition.

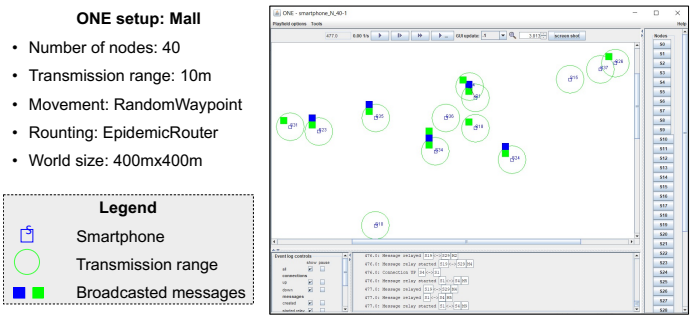

Fig. 7. Mall simulation with the ONE (reduced to 40 nodes for readability)

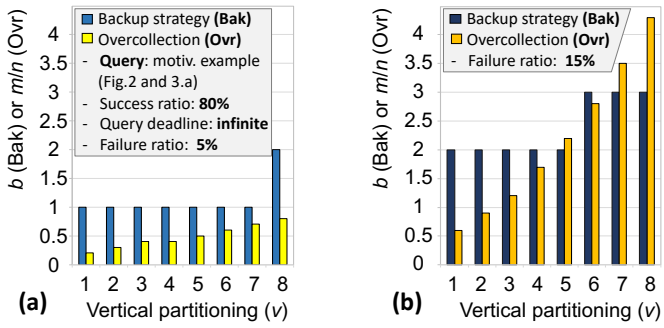

Fig. 8. Backup vs Overcollection strategies

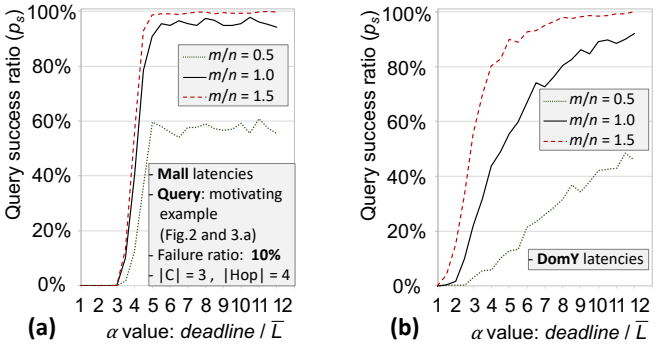

Fig. 9. Query deadlines (for Overcollection)

Conclusion: Except for very high values of v (which correspond to unusual scenarios) combined with high failure rate, Ovr should be preferred to Bak since it avoids costly consensus or QEP rearrangement (see Section IV.B). Note that considering infinite deadlines favors Bak since, with real deadlines, increasing b leads to decrease the timeouts for Bak, thus increasing p_f.

Experiment 2: Query deadline (non-iterative queries). We execute the QEP of Fig. 5 with Ovr, considering vertical partitioning on 3 Computers ($v = 3$). We used 3 values of m/n, ranging from 0.5 to 1.5 wrt. the first experiment. To enable comparisons between DomY and Mall, we used on x-axis the query deadline divided by $\bar{L}$, called α hereafter, and measure the query success ratio (y-axis on Fig. 9.a and 9.b).

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.

Conclusion: m/n should not be underestimated and the query deadline should be fixed larger than $\bar{L} \times |hops|$ where $|hops|$ is the number of hops in the query plan (in this case, 4: Contributor➛Snapshot Builder➛Computers➛ Computing Combiners➛Querier). 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.

Experiment 3: Iterative computations quality. We simulate the iterative algorithms Apriori and k-means, considering synthetic and real data sets used to evaluate their quality (e.g., [27] for Apriori). Thanks to the simulator, we systematically measure the quality of Edgelet executions against centralized fault-free executions (y-axis on Fig. 10.a and 10.b). 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. To evaluate iterative methods in extreme conditions, we reduced artificially the HeartBeat duration such that the observed proportion of late messages (i.e., messages arrived after the HeartBeat) is 80%, 90% and 95% (see Fig. 10.a and 10.b). As a consequence, DomY and Mall contexts produce approximately the same results (DomY results are shown).

Observations: (1) Even with no iteration, Apriori reaches 65% recall and k-means reaches 30% degradation of its inertia; (2) 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); (3) with Apriori, precision is always 100% (not shown). We verified that Computing Combiners always receive n or more sub-QEPs results and can then remove potential false positive; (4) with k-means, we observe a negative PCI. Indeed, the Edgelet computation with many heartbeats is better than the centralized one since it may consider up to m additional partitions.

Conclusion: HeartBeat execution shows quite good results on Apriori and k-means despite an (artificially) high ratio of late messages. Other classical iterative algorithms deserve further study.

These evaluation results are only preliminary but they suggest that regular queries can be executed in the Edgelet context while enforcing Resiliency, Validity and Crowd liability with performance compatible with the targeted applications, and that even ML algorithms could be supported with a pretty good accuracy of their results.

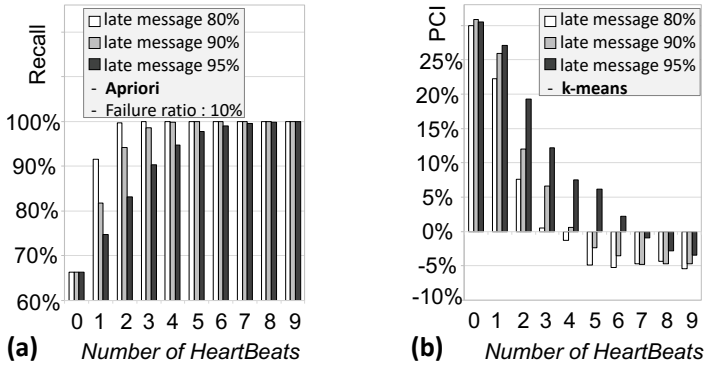

Fig. 10. Heartbeat execution quality

VII. BACKUP VS. OVERCOLLECTION: SOME DESIGN RULES

This section compares qualitatively the two execution strategies proposed in Sections IV and V. Our goal is to guide a potential Querier towards the right execution model when designing a computation dedicated to the Edgelet computing paradigm. This choice depends on the type of computation to be performed, the way to define a representative snapshot for this computation and the expected query deadline. We can draw some design rules from the statements summarized in Table 1.

In this table, *Horizontally/Vertically partitionable* refers to the property of the computation to be distributed among several Data Processors, as explained in Section III.A. 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 constraint on the processing, horizontal partitioning requires that predicate 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 to exploit the Overcollection-based strategy for this QEP. *Ground validity* means that the Validity property defined in Section II cannot be enforced; hence, it is up to the Querier to assess empirically the accuracy of the final result, as we did in Section VI for Apriori and k-means. Finally, The *Crowd liability* column expresses the additional amount of data exposed by each strategy compared to an ideal strategy without resiliency (i.e., without Backups nor Overcollection).

Resiliency	Type of computation	Snapshot definition	Validity	Crowd liability	Success Rate
Backup based resiliency	Horizontally partitionable	P partitionable	By construction	Activated backup exposed	Requires large query deadline
	Vertically partitionable	Any P	Consensus		
	Others	Any P	By construction		
	Iterative				*Unrealistic*
Over collection based resiliency	QEP reshapable	P partitionable	By construction	Over collected data exposed	Supports small query deadline
	Iterative reshapable	P partitionable	Ground validity		
	Others		*Invalid*		

Table 1: A taxonomy of computing strategies

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. Second, if the computation is iterative, the Overcollection strategy turns to be the only solution as well. Indeed, using Backup, 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 in this case. Third, if the QEP is reshapable and P is partitionable, the right choice between Backup and Overcollection is driven by the expected success rate and by privacy considerations. Using Backup, 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 for which the query deadline depends only on $\bar{L}$ and $|hops|$; this is especially true if the fault presumption rate p_f is high, making this criterion dependent of the OppNet quality. Regarding privacy, both methods do not expose data in the same way. Indeed, as explained in Section IV, TEEs guarantee that data at rest is not exposed in backups until the backup is 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. In the Overcollection model, 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. This factor may influence the approval of the Trusted Regulatory Agent and the participants' consent as well and finally depends on the privacy model exposed by the Querier.

The taxonomy of solutions presented in Table 1 is only preliminary and more subtle design rules can be envisioned. Hybrid strategies mixing backups and Overcollection deserve to be considered in the future, with the goal to decrease the extra amount of data exposed by each strategy (e.g., exploit over-collected partitions only if initial once are lost or backup the root of an over-collected QEP statically instead of pushing it in the Computing Combiner directly). Similarly, Ground validity should be more deeply investigated with the goal to identify finer classes of algorithms for which better validity guarantees can be expected. For example, the Apriori implementation sketched in Section V.C 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 be able to guarantee the convergence for some algorithms when specific conditions are met but let these issues for future work.

VIII. RELATED WORKS

A lot of research work has been produced on the different areas encompassed by Edgelet computing: computing architectures, types of communication, resiliency strategies and confidential computing techniques. To the best of our knowledge, none of the aforementioned topics considered the conjunction of fully decentralized architecture on users' devices and opportunistic networks while computing data-intensive algorithms and ensuring safety, liveness and security properties.

Crowd processing and Edge Computing. Crowd Computing [28], Crowdsourcing [29], 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 [31], [32]. Like Cross-Device Federated Learning [33], these technologies operate in connected infrastructures where processing is performed in micro data centers (MDCs) [34] or small clouds (Cloudlets) [35]. 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. An approach quite similar to ours is the system of composable services "Zoo" [36], but the authors focused on a framework for composing ML algorithms on connected edge devices, while we focused on defining a general paradigm in a disconnected environment.

Data processing in WSN. Wireless Sensor Networks were introduced to address monitoring and tracking needs [37]. They are based on weakly connected sensors with low computational capabilities. Their primary purpose is to transmit the collected data to monitoring stations for analysis. The processing performed on these devices is then streamed queries [38], [39], the majority of which is executed at the end of the chain in a centralized manner. Most use-cases consider sensors for the environment or for animal tracking and do not raise privacy issues. Our approach is therefore quite different. We are closer to [40] which proposes a database system that leverages the capabilities of edge devices to run complex algorithms in a distributed manner to alleviate IoT data pressure. This decentralization of processing also allows us to reduce the exposure of personal data, a key concern in our approach.

Fault-tolerance strategies. The standard approach to ensure queries termination in distributed systems is replication [21], [41]. Fault-tolerance issues significantly raise in complexity when asynchronous systems are considered, and the duality between safety and liveness still impacts the most recent works. For instance, [38] proposes a redundancy system called Replicated Dataflow Graph (RDG) and suggests a "routing constraint" mechanism to coordinate data sources and replicas. But as the authors explain, this does not totally prevent the occurrence of routing inconsistencies, although infrequent in practice. Our backup-based solution, based on these same principles, faces the same difficulties. In contrast, the Overcollection approach circumvents the problems by replicating only the operators and not the data, which makes it easier to guarantee the validity property with the advantage of limiting the exposure of individual data.

Privacy-preserving computations. In order to ensure confidentiality of processing, our approach relies on TEEs present in edgelets, which promotes computational genericity and execution scaling. Existing distributed computing schemes based on fully homomorphic encryption [7] or secure Multi-Party Computation (MPC) cannot support any type of operation and scale to a large number of participants at the same time. For example, distributed database solutions, such as SMCQL [9], are limited to a few dozen participants. Only specific operations can scale up for ad hoc constructions only (e.g., secure sum [42]). Similarly, confidential distributed computing using gossip protocols [43] and differential privacy techniques [8] are limited to a small set of operations and produce approximate results. Moreover, "central" differential privacy relies on a trusted server, which contradicts our assumptions. The "local" differential privacy approach addresses this problem, but at the price of reducing further the utility of the computational results.

IX. Conclusions and Future Challenges

In this paper, we introduce the Edgelet Computing paradigm, a new framework for executing complex and privacy-preserving distributed queries on personal devices at the extreme edge of the network. This paradigm builds on the recent convergence between Trusted Execution Environments and Opportunistic Networks and opens disruptive ways to handle personal data management problems. First, the paper precisely characterizes this paradigm through an ad-hoc threat model and properties guaranteeing the soundness of the executions. Second, it presents two alternative computing strategies satisfying these properties under this threat model. Third, it evaluates quantitatively and qualitatively these strategies and draws preliminary design rules to adapt computations to the Edgelet computing paradigm.

While our first experiments give confidence in the relevance and feasibility of the approach, we now aim at validating it through a field deployment of the solution. So far, 4,000 over 8,000 edgelets (i.e., patient's box) have been deployed in the DomYcile project [11] and we are working hard to make decentralized queries operational in the very near future. The ability to query cohorts of disconnected patients in few days and the unusual threat model provided by the Edgelet paradigm has been a strong incentive for the Yvelines district to adopt this technology.

However, this paper is still a preliminary study and several challenges remain to be tackled.

A first challenge is to explore the multiple **optimization tradeoffs** that exist between data exposure, query success rate and (local and global) resource consumption. New design rules should be introduced to accommodate existing algorithms to the Edgelet context in a way that facilitates the calibration between these optimization objectives.

A second challenge is to **optimize the task-to-edgelet assignment**. A large bunch of work has been devoted to the exploitation of social relationships and location-based homophily to improve network routing in OppNets [13], [14]. The same kind of optimization could be envisioned to improve the query processing by assigning operators to the most connected devices (e.g. doctors, teachers) or to devices having the highest probability of meeting. This would lead to revisit the Crowd liability property, notably its randomness assumption to allow a (limited and controlled) degree of bias in favor of edgelets with good social relationships.

A third challenge is to accommodate **long-lasting snapshots** 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, with various impacts on Resiliency, Validity and Crowd liability.

Other challenges are **multi-disciplinary**. Crowd liability has never been considered so far in the legislation protecting personal data. For example, the GDPR considers as liable a single so-called "data controller" entity, while liability is scattered among the crowd in our context. Jurists and computer scientists should remedy this shortfall by translating the Edgelet threat model in appropriate obligations for each party in the computation. It makes also sense in some contexts to track the participants in a processing (e.g., to provide them feedbacks or rewards or alert them from a leakage). This requires designing a provenance mechanism compatible with the new-defined properties. The acceptability of such mechanisms raises also new multidisciplinary challenges with social scientists.

Hence, we strongly believe that the Edgelet computing paradigm raises many exciting and promising challenges for the database and distributed systems community and that it can play a major role in the emergence of a new important class of applications related to the management of sensitive data.

X. REFERENCES

[1] "Cisco Annual Internet Report (2018–2023) White Paper," Cisco, 2020. https://tinyurl.com/cisco-internet-report

[2] M. Conti et al., "The Internet of People (IoP): A new wave in pervasive mobile computing," Pervasive Mob. Comput., vol. 41, pp. 1–27, 2017.

[3] S. Trifunovic et al., "A Decade of Research in Opportunistic Networks: Challenges, Relevance, and Future Directions," IEEE Communications Magazine, vol. 55, no. 1, pp. 168–173, 2017.

[4] M. Sabt et al., "Trusted Execution Environment: What It is, and What It is Not," in IEEE Trustcom/BigDataSE/ISPA, vol. 1, pp. 57–64, 2015.

[5] V. Costan and S. Devadas, "Intel SGX Explained," IACR Cryptol EPrint Arch, p. 86, 2016.

[6] S. Pinto and N. Santos, "Demystifying Arm TrustZone: A Comprehensive Survey," ACM Computing Surveys, vol. 51, no. 6, pp. 1–36, 2019.

[7] D. Boneh et al., "Private Database Queries Using Somewhat Homomorphic Encryption," in ACNS, vol. 7954, pp. 102–118, 2013.

[8] C. Dwork, "Differential Privacy," in Automata, Languages and Programming, 33rd Int. Colloquium, ICALP, vol. 4052, pp. 1–12, 2006.

[9] J. Bater et al., "SMCQL: Secure Query Processing for Private Data Networks," VLDB Endowment, vol. 10, no. 6, pp. 673–684, 2017.

[10] "Proposal for a Regulation of the European Parliament and of the Council on European Data Governance (Data Governance Act)," 2020. https://tinyurl.com/data-governance-act

[11] "DomYcile Project," https://tinyurl.com/domycile, "Salon E-Tonomy," https://tinyurl.com/e-tonomy

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

[13] P. Bellavista et al., "Mobile social networking middleware: A survey," Pervasive and Mobile Computing, vol. 9, no. 4, pp. 437–453, 2013.

[14] K. Pelechrinis and P. Krishnamurthy, "Socio-spatial affiliation networks," Computer Communications, vol. 73, pp. 251–262, 2016.

[15] A. Benchi et al., "Solving Consensus in Opportunistic Networks," in Int. Conf. on Distributed Computing and Networking, pp. 1–10, 2015.

[16] W. Wang et al., "Leaky Cauldron on the Dark Land: Understanding Memory Side-Channel Hazards in SGX," in Computer and Communications Security, pp. 2421–2434, 2017.

[17] R. Mortier et al., "Personal Data Management with the Databox: What's Inside the Box?," in ACM Workshop on Cloud-Assisted Networking, pp. 49–54, 2016.

[18] F. Tramer et al., "Sealed-Glass Proofs: Using Transparent Enclaves to Prove and Sell Knowledge," in EuroS&P, pp. 19–34, 2017.

[19] M. Herr et al., "Frailty, polypharmacy, and potentially inappropriate medications in old people: findings in a representative sample of the French population," European Journal of Clinical Pharmacology, vol. 73, no. 9, pp. 1165–1172, 2017.

[20] "The Survey of Health, Ageing and Retirement in Europe (SHARE): Home," http://www.share-project.org/home0.html, 2004-2020.

[21] M. Wiesmann et al., "Understanding replication in databases and distributed systems," in ICDCS, Taipei, Taiwan, pp. 464–474, 2000.

[22] R. Ladjel et al., "Trustworthy Distributed Computations on Personal Data Using Trusted Execution Environments," in TrustCom, 2019.

[23] A. Savasere et al., "An Efficient Algorithm for Mining Association Rules in Large Databases," in VLDB, Zurich, Switzerland, pp. 432–444, 1995.

[24] I. S. Dhillon and D. S. Modha, "A Data-Clustering Algorithm on Distributed Memory Multiprocessors," in Large-Scale Parallel Data Mining, Workshop on Large-Scale Parallel KDD Systems, SIGKDD, San Diego, CA, USA, vol. 1759, pp. 245–260, 1999.

[25] A. Keränen, J. Ott, and T. Kärkkäinen, "The ONE Simulator for DTN Protocol Evaluation," in SIMUTools'09, New York, NY, USA, 2009.

[26] A. Vahdat and D. Becker, "Epidemic Routing for Partially-Connected Ad Hoc Networks," Technical Report CS-200006, Duke University, 2000.

[27] T. Brijs, "Retail market basket data set," in Workshop on Frequent Itemset Mining Implementations (FIMI'03), 2003.

[28] B. Guo et al., "Mobile Crowd Sensing and Computing: The Review of an Emerging Human-Powered Sensing Paradigm," ACM Computing Surveys, vol. 48, no. 1, pp. 1–31, 2015.

[29] E. Estellés-Arolas and F. González-Ladrón-de-Guevara, "Towards an integrated crowdsourcing definition," Journal of Information Science, vol. 38, no. 2, pp. 189–200, 2012.

[30] R. K. Ganti et al., "Mobile crowdsensing: current state and future challenges," IEEE Commun. Mag., vol. 49, no. 11, pp. 32–39, 2011.

[31] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu, "Edge Computing: Vision and Challenges," IEEE Internet Things J., vol. 3, no. 5, pp. 637–646, 2016.

[32] D. Xu et al., "Edge Intelligence: Empowering Intelligence to the Edge of Network," Proceedings of the IEEE, vol. 109, no. 11, pp. 1778–1837, 2021.

[33] K. A. Bonawitz et al., "Towards Federated Learning at Scale: System Design," in Machine Learning and Systems (MLSys'19), USA, 2019.

[34] A. J. Ferrer, J. M. Marquès, and J. Jorba, "Towards the Decentralised Cloud: Survey on Approaches and Challenges for Mobile, Ad hoc, and Edge Computing," ACM Comput. Surv., vol. 51, no. 6, pp. 1–36, 2019.

[35] M. Satyanarayanan, P. Bahl, R. Cáceres, and N. Davies, "The Case for VM-Based Cloudlets in Mobile Computing," IEEE Pervasive Computing, vol. 8, no. 4, pp. 14–23, 2009.

[36] J. Zhao et al., "Privacy-Preserving Machine Learning Based Data Analytics on Edge Devices," in AAAI/ACM Conference on AI, Ethics, and Society, pp. 341–346, 2019.

[37] J. Yick, B. Mukherjee, and D. Ghosal, "Wireless sensor network survey," Computer Networks, vol. 52, no. 12, pp. 2292–2330, 2008.

[38] D. O'Keeffe, T. Salonidis, and P. R. Pietzuch, "Frontier: Resilient Edge Processing for the Internet of Things," VLDB Endowment, vol. 11, no. 10, pp. 1178–1191, 2018.

[39] S. Zeuch et al., "The NebulaStream Platform for Data and Application Management in the Internet of Things," in CIDR, 2020.

[40] J. Paparrizos et al., "VergeDB: A Database for IoT Analytics on Edge Devices," in CIDR, 2021.

[41] H. Alwan and A. Agarwal, "A Survey on Fault Tolerant Routing Techniques in Wireless Sensor Networks," in IEEE SENSORCOMM, Athens, Greece, pp. 366–371, 2009.

[42] R. Sheikh, B. Kumar, and D. K. Mishra, "A Distributed k-Secure Sum Protocol for Secure Multi-Party Computations," *CoRR*, vol. abs/1003.4071, 2010.

[43] T. Allard et al., "Chiaroscuro: Transparency and Privacy for Massive Personal Time-Series Clustering," in ACM SIGMOD International Conference on Management of Data, Melbourne Victoria Australia, pp. 779–794, 2015.