A Survey of Distributed Data Aggregation Algorithms

Paulo Jesus, Carlos Baquero and Paulo Sérgio Almeida

Abstract—Distributed data aggregation is an important task, allowing the decentralized determination of meaningful global properties, which can then be used to direct the execution of other applications. The resulting values are derived by the distributed computation of functions like COUNT, SUM and AVERAGE. Some application examples deal with the determination of the network size, total storage capacity, average load, majorities and many others. In the last decade, many different approaches have been proposed, with different trade-offs in terms of accuracy, reliability, message and time complexity. Due to the considerable amount and variety of aggregation algorithms, it can be difficult and time consuming to determine which techniques will be more appropriate to use in specific settings, justifying the existence of a survey to aid in this task. This work reviews the state of the art on distributed data aggregation algorithms, providing three main contributions. First, it formally defines the concept of aggregation, characterizing the different types of aggregation functions. Second, it succinctly describes the main aggregation techniques, organizing them in a taxonomy. Finally, it provides some guidelines toward the selection and use of the most relevant techniques, summarizing their principal characteristics.

Index Terms—distributed algorithms, data aggregation, performance trade-offs, fault-tolerance

1 Introduction

Data aggregation is an essential building block of modern distributed systems, enabling the determination of important system wide properties in a decentralized manner. The knowledge of these global properties can then be used as input by other distributed applications and algorithms. The network size is a common example of such global properties, which is required by many algorithms in the context of Peer-to-Peer (P2P) networks, for instance: in the construction and maintenance of Distributed Hash Tables (DHT) [1], [2]; when setting the fanout of a gossip protocol [3]. This size estimation is also used in many other contexts, for example: to set up a quorum in dynamic settings [4], to compute the mixing time of a random walk [5].

The network size is computed through the COUNT aggregation function. Nevertheless, other meaningful global properties can be computed using different functions, for example: AVERAGE can be applied to determine the average system load which can be used to direct local load balancing decisions; SUM allows the determination of total values such as the total free disk space available in a file-sharing system. In the particular case of Wireless Sensor Networks (WSN) and for the Internet of Things (IOT) [**?**], data gathering is often only practical if data aggregation is performed, due to the strict energy constraints found on such environments. Even when energy is available, there is still need for energy efficient approaches [**?**].

The above examples intend to illustrate some of the main reasons that have motivated the research and development of distributed data aggregation approaches over the past years, but more can be found in the literature. Besides all the existing relevant application examples, aggregation has also been stated as one of basis for scalability in large-scale services [6], reinforcing its importance. Currently, a huge amount of distinct approaches constitute the body of related work on distributed data aggregation algorithms, with all exhibiting different trade-offs in terms of accuracy, time, communication and fault-tolerance. Together, existing techniques have confirmed that obtaining global statistics in a distributed fashion is a difficult problem, especially when considering faults and network dynamism. Moreover, in face of such diversity, it becomes difficult to choose which distributed data aggregation algorithm should be preferred in a given scenario, and which one will best suit the requirements of a specific application. One of the main motivations of this work is precisely to help readers make this choice.

Some surveys have been previously published [7], [8], [9], [10], [11] focusing specifically on aggregation techniques for WSN. Several in-network aggregation techniques for WSN are depicted in [7], which typically operate at the network level, needing to deal with the resource constraints of sensor nodes (limited computational power, storage and energy resources). A review of the existing literature more focused on energy efficiency is presented in [8], and on security in [9], [10]. Another work reviewing the state-of-the-art of information fusion techniques for WSN is also available [11]. In this later work, a broader view of the sensor fusion process is reviewed, from raw data collection, passing through a possible summarization (data aggregation) or compression, until the final resulting decision or action is reached. Data aggregation is considered a subset of

• *P. Jesus, C.Baquero and P.S.Almeida are with HASLab / INESC TEC & Universidade do Minho, Braga, Portugal.*

information fusion, that aims at reducing the handled data volume by establishing appropriate summaries.

In this survey, we intend to address data aggregation algorithms at a higher abstraction level, providing a comprehensive and more generic view of the problem independently from the type of network used. Unlike most of the previous surveys, we do not focus on algorithms for specific networks, like WSN. Instead, we provide a wider view of the existing techniques to tackle specifically the distributed data aggregation problem, as the performance of the algorithms vary according to the underlying network settings. Therefore, more alternatives are presented to the reader compared to previous works, allowing the determination of which data aggregation algorithms might be more suited for any target setting. We define the problem of computing aggregation functions in a distributed fashion, and detail a wide range of distinct solutions. A taxonomy of the existing distributed data aggregation algorithms is proposed, classifying them according to two main perspectives: communication and computation. The first viewpoint refers to the routing protocols and intrinsic network topologies associated with each protocol, which are used to support the aggregation process. The second perspective points out the aggregation functions computed by the algorithms and the main principles from which they are based on. Other perspectives (e.g., algorithm requirements, covered types of aggregation functions) could have been considered, since the mapping between the algorithm attributes is multidimensional. However, we believe that the two chosen perspectives will provide a clear presentation. We also give some important guidelines to help in the decision of which distributed aggregation algorithm should be used, according to the requirements of a target application and environment.

The remaining of this survey is organized as follows. In Section 2, we clarify the concept of aggregation, and define the problem of its distributed computation. A taxonomy of the existing distributed aggregation algorithms from the communication perspective is proposed in Section 3, describing the most relevant approaches and discussing their *pros* and *cons*. Section 4 proposes a taxonomy from the computation perspective. Section 5 summarizes the properties of the most relevant approaches, and gives some guidelines for their practical application. Finally, some concluding remarks and future research directions for the area are drawn in Section 6.

2 Problem Definition

In a nutshell, aggregation can be simply defined as "the ability to summarize information", quoting Robbert Van Renesse [6]. Data aggregation is considered a subset of information fusion, aiming at reducing the handled data volume [11]. Here, we provide a more precise definition, and consider that the process consists in the computation of an *aggregation function* defined by:

Definition 2.1 (Aggregation function): An aggregation function f takes a multiset[1] of elements from a set I and produces an output from a set O:

$$f : \mathbb{N}^I \to O$$

The input being a multiset emphasizes that: 1) the order in which the elements are aggregated is irrelevant; 2) a given value may occur several times. Frequently, for common aggregation functions such as MIN, MAX, and SUM, both I and O are the same set. For others, such as COUNT (which gives the cardinality of the multiset), the result is an integer, regardless of the input set domain.

An aggregation function aims to summarize information. Therefore, the result of an aggregation (in the output set O) typically takes much less space than the multiset to be aggregated (an element from $\mathbb{N}^I$). We will leave unspecified what is acceptable for some function to be considered as summarizing information, and therefore, an aggregation function. It can be said that an element of the output set O is not normally a multiset (we do not have normally $O = \mathbb{N}^I$) and that the identity function is clearly not an aggregation function as it definitely does not summarize information. In most practical cases, the size of the output is at most a logarithm of the input size, and often even of constant size.

2.1 Decomposable functions

For some aggregation functions we may need to perform a single computation involving all elements in the multiset. For many cases, however, one needs to avoid such centralized computation. In order to perform distributed *in-network* aggregation, it is relevant whether the aggregation function can be *decomposed* into several computations involving sub-multisets of the multiset to be aggregated. For distributed aggregation it is useful, therefore, to define the notion of *decomposable aggregation function*, and a subset of these, which we call *self-decomposable aggregation functions*.

Definition 2.2 (Self-decomposable aggregation function): An aggregation function $f : \mathbb{N}^I \to O$ is said to be self-decomposable if, for some (merge) operator $\diamond$ and all non-empty multisets X and Y:

$$f(X \uplus Y) = f(X) \diamond f(Y)$$

In the above, $\uplus$ denotes the standard multiset sum (see, e.g.,[12]). According to the above definition, and given that the aggregation result is the same for all possible partitions of a multiset into sub-multisets, it means that the operator $\diamond$ is commutative and associative. Many traditional functions such as MIN, MAX, SUM and COUNT are self-decomposable:

$$\begin{aligned} \text{SUM}(\{x\}) &= x, \\ \text{SUM}(X \uplus Y) &= \text{SUM}(X) + \text{SUM}(Y). \end{aligned}$$

1. A generalization of set, in which elements are allowed to appear more than once.

$$\begin{aligned}\text{COUNT}(\{x\}) &= 1,\\ \text{COUNT}(X \uplus Y) &= \text{COUNT}(X) + \text{COUNT}(Y).\end{aligned}$$

$$\begin{aligned}\text{MIN}(\{x\}) &= x,\\ \text{MIN}(X \uplus Y) &= \min(\text{MIN}(X), \text{MIN}(Y)).\end{aligned}$$

Definition 2.3 (Decomposable aggregation function): An aggregation function $f : \mathbb{N}^I \to O$ is said to be decomposable if for some function g and a self-decomposable aggregation function h, it can be expressed as:

$$f = g \circ h$$

From this definition, the self-decomposable functions are a subset of the decomposable functions, where $g = \text{ID}$, the identity function. While for self-decomposable functions the *intermediate* results (e.g., for in-network aggregation) are computed in the output set O, for a general decomposable function, we may need a different auxiliary set to hold the intermediate results.

The classic example of a decomposable (but not self-decomposable) function, is AVERAGE, which gives the average of the elements in the multiset:

$$\begin{aligned}\text{AVERAGE}(X) &= g(h(X)), \qquad \text{with}\\ h(\{x\}) &= (x, 1),\\ h(X \uplus Y) &= h(X) + h(Y),\\ g((s,c)) &= s/c.\end{aligned}$$

in which h is a self-decomposable aggregation function that outputs values of an auxiliary set, in this case pairs, and $+$ is the standard pointwise sum of pairs (i.e., $(x_1, y_1) + (x_2, y_2) = (x_1 + x_2, y_1 + y_2)$). Another example is the RANGE function in statistics, which gives the difference between the maximum and the minimum value.

2.2 Duplicate sensitiveness and idempotence

Depending on the aggregation function, it may be relevant whether a given value occurs several times in the multiset. For some aggregation functions, such as MIN and MAX, the presence of duplicate values in the multiset does not influence the result, which only depends on its *support set* (the set obtained by removing all duplicates from the original multiset). E.g.,

$$\text{MIN}(\{1,3,1,2,4,5,4,5\}) = \text{MIN}(\{1,3,2,4,5\}) = 1$$

For others, such as SUM and COUNT, the number of times each element occurs (its multiplicity) is relevant:

$$\begin{gathered}\text{COUNT}(\{1,3,1,2,4,5,4,5\}) = 8\\ \neq\\ \text{COUNT}(\{1,3,2,4,5\}) = 5\end{gathered}$$

Therefore, duplicate sensitiveness is important and must be taken into account for distributed aggregation, in order to compute the correct result.

TABLE 1
Taxonomy of aggregation functions

	Decomposable		Non-decomposable
	Self-decomposable		
Duplicate insensitive	MIN, MAX	RANGE	DISTINCT COUNT
Duplicate sensitive	SUM, COUNT	AVERAGE	MEDIAN, MODE

Definition 2.4 (Duplicate insensitive aggregation function): An aggregation function f is said to be duplicate insensitive if for all multisets M, $f(M) = f(S)$, where S is the support set of M.

Moreover, some duplicate insensitive functions (like MIN and MAX) can be implemented using an idempotent binary operator that can be successively applied (by intermediate nodes) on the elements of the multiset (any number of times without affecting the result). This helps in obtaining fault tolerance and decentralized processing, allowing retransmissions or sending values across multiple paths (without affecting the aggregation result). Unfortunately, the distributed application of an idempotent operator is not always possible, even for some duplicate insensitive aggregation functions, such as DISTINCT COUNT (i.e., cardinality of the support set). In practice, the application of an idempotent operator in a distributed way to compute an aggregation function appears to be only possible if the function is duplicate insensitive and self-decomposable.

2.3 Taxonomy of common aggregation functions

Building on the concepts of decomposability and duplicate sensitiveness, we can obtain a taxonomy of aggregation functions (Table 1). It contains the standard aggregation functions, under the usual statistical definition. In addition to those already mentioned, it contains MEDIAN (the value separating the higher half from the lower half, after ordering the values), and MODE (the value that appears most often).

This table helps to clarify how suited an aggregation function is to a distributed computation. Non-decomposable aggregation functions are harder to compute than decomposable ones, being vulgarly labeled in the literature as “complex”. Commonly, duplicate insensitive aggregation functions are easier to compute than duplicate sensitive. As we will see in the next section, one way to obtain fault-tolerance (at the cost of some accuracy) is to use duplicate insensitive approaches to estimate some aggregation function (e.g., *sketches*, see section 4.3), replacing the use of non-idempotent operations (like SUM) by idempotent ones (like MAX).

This simple classification of common aggregation functions is orthogonal to the taxonomies provided in the next two sections about the distributed algorithms that can be used to compute them.

3 Communication Taxonomy

Three major classes of aggregation algorithms are identified from the communication perspective, according to the characteristics of their communication/routing strategy and network topology (see Table 2): *structured* (usually, hierarchy-based), *unstructured* (usually, gossip-based), and *hybrid* (mixing the previous categories).

The *structured* communication class refers to aggregation algorithms that are dependent from a specific network topology and routing scheme to operate correctly. If the required routing topology is not available, then an additional preprocessing phase is needed in order to create it, before starting the execution of the algorithm. This dependency limits the use of these techniques in dynamic environments. For instance, in mobile networks these algorithms need to be able to continuously adapt their routing structure to follow network changes. Typically, these algorithms are directly affected by problems tied to the used routing structure. For example, in tree-based communication structures a single point of failure (node/link) can compromise the delivery of data from all its subtrees, and consequently impair the applications supported by that structure. In practice, due to their energy efficiency, hierarchical communication structures (e.g., tree routing topology) are the most often used to perform data aggregation, especially in WSN. Alternative routing topologies are also considered, like the ring topology, although very few approaches rely on it.

The *unstructured* communication category covers aggregation algorithms that can operate independently from the network organization and structure, without establishing any predefined topology. In terms of communication, this kind of algorithms is essentially characterized by the used communication pattern: *flooding/broadcast*, *random walk* and *gossip*. The *flooding/broadcast* communication pattern is associated with the dissemination of data from one node to all the network nodes or to a group of nodes – "one to all". A *random walk* consists in sequential message transmissions, from one node to another – "one to one". The *gossip* communication pattern refers to a well-known communication protocol, based on the spreading of a rumor [13] (or an epidemic disease), in which messages are sent successively from one node to a selected number of peers – "one to many". In the recent years, several aggregation algorithms based on gossip communication have been proposed, leveraging properties of simplicity, scalability and robustness. More details about these different communication patterns (see Table 2), applied to data aggregation, will be further described.

The *hybrid* class groups algorithms that mix the use of different routing strategies from the previous categories, with the objective of combining good properties and overcoming weakness, while deriving equivalent or improved aggregations (more details in Section 3.6).

TABLE 2
Taxonomy from a communication perspective

	Routing	**Algorithms**
Structured	*Hierarchy (tree, cluster, multipath)*	TAG [14], DAG [15], I-LEAG [16], Sketches [17], RIA-LC/DC [18], [19], Tributary-Delta [20], Q-Digest [21]
	Ring	(Horowitz and Malkhi, 2003) [22]
Unstructured	*Flooding/Broadcast*	Randomized Reports [23]
	Random walk	Random Tour [24], (Dolev et al., 2002) [25], [26], Sample & Collide [27], [24], Capture-Recapture [28]
	Gossip	Push-Sum Protocol [29], Push-Pull Gossiping [30], DRG [31], Flow Updating [32], [33], Extrema Propagation [34] Equi-Depth[35], Adam2 [36], Hop-Sampling [37], [38], Interval Density [37], [38]
Hybrid	*Hierarchy + Gossip*	(Chitnis et al., 2008) [39]

3.1 Hierarchy-based approaches

Traditionally, existing aggregation algorithms operate on a hierarchy-based communication scheme. Hierarchy-based approaches are often used to perform data aggregations, especially in WSN. This routing strategy consists in the definition of a hierarchical communication structure (e.g., spanning tree), rooted at a single point, commonly designated as *sink*. In general, in a hierarchy-based approach the data is simply disseminated from level to level, upwards in the hierarchy, in response to a query request made by the sink, which computes the final result. Besides the sink, other special nodes can be defined to compute intermediate aggregates, working as aggregation points that forward their results to upper level nodes until the sink is reached.

Aggregation algorithms based on hierarchic communication usually work in two phases, involving the participation of all nodes in each one: *request* phase and *response* phase. The *request* phase corresponds to the spreading of an aggregation request throughout the whole network. Several considerations must be taken into account before starting this phase, depending on which node wants to perform the request and on the existing routing topology. For instance: if the routing structure has not been established yet, it must be created and ought to be rooted at the requesting node; if the required topology is already established, first the node must forward its request to the root, in order to be spread (from the sink) across all the network. During the *response* phase, all the nodes answer the aggregation query by sending the requested data toward the sink. In this phase, nodes can be asked to simply forward the received data or to compute partial intermediate aggregates to be sent.

The aggregation structure of hierarchy-based approaches provides a simple strategy that enables the exact computation of aggregates (if no node or communication failures happen), in an efficient manner in terms of energy consumption (i.e., message load). However, in adverse environments this type of approach exhibits some fragility in terms of robustness, since a single point of failure can jeopardize the obtained result. Furthermore, to correctly operate in dynamic environments, where the network continuously changes, with nodes joining and leaving, extra procedures are required to maintain an updated routing structure.

3.1.1 TAG

The Tiny AGgregation service for ad-hoc sensor networks described by Madden et al. [14] represents a classical tree-based in-network aggregation approach. As referred by the authors, TAG is agnostic to the implementation of the tree-based routing protocol, as far as it satisfies two important requirements. First, it must be able to deliver query requests to all the network nodes. Second, it must provide at least one route from every node (that participates in the aggregation process) to the sink, guaranteeing that no duplicates are generated (at most one copy of every message must arrive). This algorithm requires the previous creation of a tree-based routing topology, and also the continuous maintenance of such routing structure in order to operate over mobile networks.

TAG allows the computation of basic database aggregation functions, such as COUNT, MAXIMUM, MINIMUM, SUM and AVERAGE (providing a declarative SQL-like query language with grouping capabilities). The algorithm performs a classic hierarchy-based aggregation process which consists of two phases: a *distribution phase* (in which the aggregation query is propagated along the tree routing topology, from the root to the leaves) and a *collection phase* (where the values are aggregated from the children to the parents, until the root is reached). The derivation of the aggregation result at the root incurs a minimal time overhead that is proportional to the tree depth. This waiting time is needed to ensure the conclusion of the two execution phases and the participation of all nodes in the aggregation process.

Figure 1 show an example of the execution of TAG to compute the AVERAGE. In the particular case of the AVERAGE the sum and count need to be aggregated at all intermediate nodes in order to compute the global average at the sink[2]. In this example, it is considered

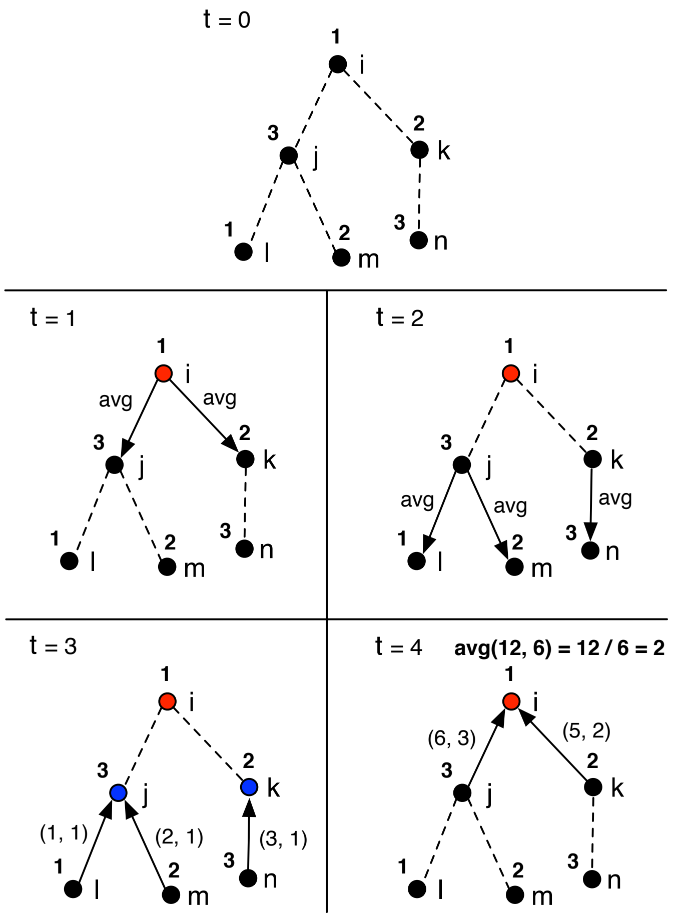


Fig. 1. Execution example of TAG to compute the AVERAGE function.

that a tree routing topology rooted at node i is already available. Then, at time $t = 1$ node i initiate the distribution phase by sending an average request to its children (nodes j and k) which at their turn will also propagate the query to all their children. At time $t = 3$, upon the reception of the average query, the leaves initiate the collection phase, reporting to their parents the value of their local reading for the sum and the value 1 for the count, in a pair $(sum, count)$. E.g., node l sends $(1, 1)$ and node m sends $(2, 1)$ to node j. Then, the intermediate nodes j and k aggregate the values received from their children with their own and send the result to their own parent. E.g., node j aggregates the received values $(1, 1)$ and $(2, 1)$ with its own $(3, 1)$ and sends the result $(6, 3) = (1+2+3, 1+1+1)$ to i. Finally, the sink aggregate the values received from all its children with its own $(6 + 5 + 1, 3 + 2 + 1) = (12, 6)$ and computes the global average result $12/6 = 2$.

A *pipelined aggregate* technique (detailed in [40]) has been proposed to minimize the effect of the waiting time overhead. According to this technique, smaller time intervals (relatively to the overall needed time) are used to repetitively produce periodic (partial) aggregation results. In each time interval, all nodes that have received the aggregation request will transmit a partial result.

2. Recall that AVERAGE is a decomposable aggregation function, but not self-decomposable (see Section 2.1).

The sent value is calculated from the application of the aggregation function to the local reading and the results received from children in the previous interval. Over time, after each successive time interval, the aggregated value will result from the participation of a growing number of nodes, increasing the reliability and accuracy of the result, becoming close to the correct value at each step. The correct aggregation result should be reached after a minimum number of iterations (in an ideal fail-safe environment).

Following the authors concerns about power consumption, additional optimization techniques were proposed to the TAG basic approach, in order to reduce the number of messages sent, taking advantage of the shared communication medium in wireless networks (which enables message snooping and broadcast) and giving decision power to nodes. They proposed a technique called *hypothesis testing*, to use in certain classes of aggregates, where each node can decide to transmit the value resulting from its subtree, only if it will contribute to the final result.

3.1.2 DAG

An aggregation scheme for WSN based on the creation of a Directed Acyclic Graph (DAG) is proposed in [15], with the objective of reducing the effect of message loss in common tree-based approaches, by allowing nodes to use alternative parents. The DAG is created by setting multiple parents (within radio range) to each node, as its next hop toward the sink.

In more detail, request messages are extended with a list of parent nodes (IDs), enabling children to learn the grandparents that are two hops away. In order to avoid duplicated aggregates, only a grandparent is chosen to aggregate intermediate values, preferably a common grandparent of its parents. The most common grandparent among the lists received from parents is chosen as the destination aggregator. Otherwise, one of the grandparents is chosen (e.g., when a node has only one parent node). Response messages are handled according to specific rules to avoid duplicate processing: they can be aggregated, forwarded or discarded. Messages are aggregated if the receiving node corresponds to the destination, forwarded if the destination is a node's parent, and otherwise discarded (destination is not the node or one of its parents). Note that, although the same message can be duplicated and multiple "copies" can reach the same node (a grandparent), they will have the same destination node and only one from the same source will be considered for aggregation after receiving all messages from children.

This method takes advantage of the path redundancy introduced by the use of multiple parents to improve the robustness of the aggregation scheme (tolerance to message loss), when compared to traditional tree-based techniques. Though a better accuracy can be achieved, it comes at the cost of higher energy consumption, as more messages, and with an increased size, are transmitted. Note that, this approach does not fully overcome the message loss problem of tree routing topologies, as some nodes may have a single parent, being dependent from the quality of the created DAG.

3.1.3 Sketches

An alternative multi-path based approach is proposed in [17] to perform in-network aggregation for sensor databases, using small sketches. The defined scheme is able to deal with duplicated data upon multi-path routing and compute duplicate-sensitive aggregates, like COUNT, SUM and AVERAGE. This algorithm is based on the probabilistic counting sketches technique introduced by Flajolet and Martin [41] (FM), used to estimate the number of distinct elements in a data collection. A generalization of this technique is proposed to be applied to duplicate-sensitive aggregation functions (non-idempotent), namely the SUM. Sketches apply a stochastic transformation to these functions, allowing duplicate insensitive aggregation over a transformed data representation, at the cost of less accuracy.

The authors consider the use of multi-path routing to support communication failures (links and nodes), providing several possible paths to reach a destination. Like common tree-based approaches, the algorithm consists of two phases: first, the sink propagates the aggregation request across the whole network; second, the local values are collected and aggregated along a multi-path tree from the children to the root. In this particular case, during the request propagation phase, all nodes compute their distance (level) to the root and store the level of their neighbors, establishing a hierarchical multi-path routing topology (similar to the creation of multiple routing trees). In the second phase, partial aggregates are computed across the routing structure, using the adapted counting sketch scheme, and sent to the upper levels in successive rounds. Each round corresponds to a hierarchy level, in which the received sketches from child nodes are combined with the local one, until the sink is reached. In the last round, the sink merges the sketches of its neighbors and produces the final result, applying an estimation function over the sketch.

Notice that, the use of an auxiliary structure to summarize all data values (FM sketches), and corresponding estimator will introduce an approximation error that will be reflected in the final result. However, according to this aggregation scheme, it is expected that data loss (mitigated with the introduction of multiple alternative paths) will have a higher impact in the result accuracy than the approximation error introduced by the use of sketches.

3.1.4 I-LEAG

This cluster-based aggregation approach, designated as Instance-Local Efficient Aggregation on Graphs (I-LEAG) [16], requires the pre-construction of a different routing structure – *Local Partition Hierarchy*, which can be viewed as a logical tree of local routing partitions. The routing structure is composed by a hierarchy of clusters (partitions), with upper level partitions comprising lower level ones. A single pivot is assigned to each partition, and the root of the tree corresponds to the pivot of the highest level cluster (that includes all the network graph). This algorithm emphasizes local computation to perform aggregation, executed in sequential phases. Each phase is associated to a level of the hierarchy, where the algorithm is executed in parallel by all partitions of the corresponding level (from lower to upper levels).

Basically, the algorithm proceeds as follow: each partition checks for local conflicts (different aggregation outputs between neighbors); detected conflicts are reported to pivots, which compute the new aggregated value and multicast the result to the partition; additionally, every node forwards the received result to all neighbors that do not belong to the partition; received values are used to update the local aggregation value (if received from a node in the current partition) or to update neighbor aggregation output (if received from a neighbor of the upper level partition), enabling the local detection of further conflicts. Conflicts are only detected between neighbors that belong to a different partition in the previous phase, with different aggregation outputs from those partitions. A timer is needed to ensure that all messages sent during some phase reach their destination by the end of the same phase.

Furthermore, two extensions of the algorithm were proposed to continuously compute aggregates over a fixed network where node inputs may change over time: MultI-LEAG and DynI-LEAG [42]. MultI-LEAG mainly corresponds to consecutive executions of I-LEAG, improved to avoid sending messages when no input changes are detected. Inputs are sampled at regular time intervals and the result of the current sampling interval is produced before the next one starts. DynI-LEAG concurrently executes several instances of MultI-LEAG, pipelining its phases (ensuring that every partition level only execute a single MultI-LEAG phase at a time), and more frequently sampling inputs to faster tracking of changes, at the cost of a higher message complexity. Despite the authors' effort to efficiently perform aggregation, these algorithms are restricted to static networks, with fixed size, without considering the occurrence of faults.

3.1.5 *Tributary-Delta*

This approach mixes the use of tree and multi-path routing schemes to perform data aggregation, combining the advantages of both towards provide a better accuracy in the presence of communication failures [20]. Two different routing regions are defined: *tributary* (tree routing, in analogy to the shape formed by rivers flowing into a main stream) and *delta* (multi-path routing, in analogy to the landform of a river flowing into the sea). The idea is to use tributaries in regions with low message loss rates to take advantage of the energy-efficiency and accuracy of a traditional tree-based aggregation scheme, and use deltas in zones where message loss has a higher rate and impact (e.g., close to the sink where messages carry values corresponding to several nodes readings) to benefit from the multi-path redundancy of sketch based schemes.

Two adaptation strategies (TD-Coarse and TD) are proposed to shrink or expand the delta region, according to the network conditions and a minimum percentage of contributing nodes predefined by the user. Prior knowledge of the network size is required, and the number of contributing nodes needs to be counted, in order to estimate the current participation percentage. Conversion functions are also required to convert partial results from the tributary into valid inputs to be used in the delta region, by the multi-path algorithm. Experimental results applying TAG [14] in tributaries and Synopses Diffusion [43] (see Section 4.3) in deltas showed that this hybrid approach performs better when compared to both aggregation algorithms when used separately.

3.1.6 *Other approaches*

Several other hierarchy-based aggregation approaches can be found in the literature, most of them differing on the supporting routing structure, or on the way it is built. Beside alternative variations of the hierarchic routing topology, some optimization techniques to the aggregation process can also be found, especially to reduce the energy-consumption in WSN.

In [44] an aggregation scheme over Distributed Hash Tables (DHT) is proposed. This approach is characterized by its tree construction protocol, which uses a *parental function* to map a unique parent to each node, building an aggregation tree in a bottom-up fashion, unlike traditional approaches. The authors consider the coexistence of multiple trees to increase the robustness of the algorithm against faults, as well as the continuous execution of a tree maintenance protocol to handle the dynamic arrival and departure of nodes. Two operation modes are proposed to perform data aggregation (and data broadcast): *default* and *on-demand* mode. In the *default* mode, the algorithm is executed in background, taking advantage of messages exchanged by the tree maintenance protocol and appending some additional information to these messages. The *on-demand* mode corresponds to the traditional aggregation scheme found on tree-based algorithms.

Zhao et al. [45] proposed an approach to continuously compute aggregates in WSN, for monitoring purposes. They assume that the network continuously computes several aggregates, from which at least one corresponds to the min/max – computed using a simple diffusion scheme. A tree is implicitly constructed during the diffusion process, with the node with the min/max value set as root, and is used for the computation of other aggregates (e.g., average and count). In practice, two different schemes are used: a *digest diffusion* algorithm to compute idempotent aggregates which is used to construct an aggregation tree, and a *tree digest* scheme similar to common hierarchy-based approaches that operates over the tree routing structure created by the previous technique.

Alternative hierarchic routing structures are found in the literature to support aggregation, namely: a Breadth First Search (BFS) tree is used in the Generic Aggregation Protocol (GAP) [46] protocol to continuously compute aggregates for network management purposes; the creation of a Group-Independent Spanning Tree (GIST) based on the geographic distribution of sensors is described in [47], taking into consideration the variation of the group of sensors that may answer an aggregation query. A previous group-aware optimization technique has been proposed: Group-Aware Network Configuration (GaNC) [48]. GaNC influences the routing tree construction by enabling nodes to preferably set parents from the same group (analyzing the GROUP BY clause of the received aggregation queries) and according to a maximum communication range, in order to decrease message size and consequently reduce energy consumption.

Some algorithms [49], [50], [51] based on swarm intelligence techniques, more precisely ant colony optimization, can also be found in the literature to construct optimal aggregation trees, once more to improve the energy efficiency of WSN. Ant colony optimization algorithms are inspired in the foraging behavior of ants, leaving pheromone trails that enable others to find the shortest path to food. In this kind of approach, the aggregation structure is iteratively constructed by artificial ant agents, consisting in the paths (from different sources to the sink) with the higher pheromone values, and where network nodes that belong to more than one path act as aggregation points.

Some studies [52], [53] have shown that deciding which node should act as a data aggregator or forwarder has an important impact on the energy-consumption and lifetime of WSN. A routing algorithm, designated Adaptive Fusion Steiner Tree (AFST), that adaptively decides which nodes should fuse (aggregate) data or simply forward it is described in [52]. AFST evaluates the cost of data fusion and transmission, during the construction of the routing structure in order to minimize energy consumption of data gathering. A further extension to this scheme was proposed to handle node arrival/departure, Online AFST [54], with the objective of minimizing the cost and impact of dynamism in the routing structure. In Low-Energy Adaptive Clustering Hierarchy (LEACH) [53], [55], a cluster-based routing protocol for data gathering in WSN, the random rotation of cluster-heads over time is proposed in order to distribute the energy consumption burden of collecting and fusing cluster's data.

Filtering strategies can also be applied to reduce energy consumption in aggregation. For instance, A-GAP [56] is an extension of GAP (previously referred) that uses filters to provide a controllable accuracy in the protocol. Local filters are added at each node in order to control whether or not an update is sent. Updates are discard according to a predefined accuracy objective, resulting in a reduction in terms of communication overhead (number of messages). Filters can dynamically adjust along the execution of the protocol, allowing the control of the trade-off between accuracy and overhead. Another similar approach to reduce message transmissions, under an allowable error value, is proposed in [57] and uses adaptively adjusting filters according to a Potential Gains Adjustment (PGA) strategy. A framework called Temporal coherency-aware in-Network Aggregation (TiNA) that filters reported sensor readings according to their temporal coherency was proposed in [48]. This framework operates on the top of existing in-aggregation schemes (e.g., TAG). In particular, TiNA defines an additional TOLERANCE clause to allow users to specify the desired temporal coherency tolerance of each aggregation query, and filter the reported sensor data (i.e., readings within the range of the specified value are suppressed).

3.2 Ring based approaches

Very few aggregation approaches are supported by a ring communication structure. This particular type of routing topology is typically out-competed by hierarchical approaches. For instance, the effect of failures in a ring can be worst than on hierarchical topologies, as a single point of failure can break the whole communication chain. Furthermore, the time complexity of rings when propagating data across the network is typically much higher, leading to slower data dissemination speed. However, this kind of topology can be explored in alternative ways that can in some sense circumvent the aforementioned limitations.

It is worth referring to an alternative approach described by Horowitz and Malkhi [22], based on the creation of a virtual ring to obtain an estimation of the network size (i.e., COUNT) at each node. This technique relies solely on the departure and arrival of nodes to estimate the network size, without requiring any additional communication. Each node of the network holds a single successor link, forming a virtual ring. It is assumed that each node possesses an accurate estimator. Upon the arrival of a new node, a random successor among the existing nodes, named *contact point*, is assigned to it. During the joining process, the new node gets the *contact point* estimator and increments it (by one). At the end of the joining process, the two nodes (joining node and *contact point*) will yield the new count estimate. Upon the detection of a departure, the inverse process is executed. This method provides a disperse estimate over the whole network, with an expected accuracy that ranges from $n/2$ to n^2, where n represent the correct network size. In the rest of this paper, we will always denote n as the network size, unless explicitly indicated otherwise. Despite the achieved low accuracy and considerable result dispersion, this algorithm has a substantially low communication cost (i.e., communicates only upon arrival/departure, without any further information dissemination; each joining node communicates only with two nodes).

3.3 Flooding/Broadcast based approaches

Flooding/Broadcast based approaches promote the participation of all network nodes in the data aggregation process. The information is propagated from a single node (usually a special one) to the whole network, sending messages to all neighbors – "one to all". This communication pattern normally induces a high network load, during the aggregation process, implying in some cases a certain degree of centralization of data exchanges. Tree-based approaches are traditional usage examples of this communication pattern, but in this case it is supported by a hierarchical routing topology. Additional examples, which are not sustained by any specific structured routing topology are described below.

3.3.1 Randomized Reports

A naive algorithm to perform aggregation consists in broadcasting a request to the whole network (independently from the existing routing topology), collecting the values from all nodes and computing the result at the starting node. This likely leads to network congestion and an expected overload of the source node, due to *feedback implosion*. However, a predefined response probability could be used to mitigate this drawback, such that network nodes only decide to respond according to the defined probability. Such *probabilistic polling* method was proposed in [23] to estimate the network size. The source node broadcasts a query request with a sampling probability p, that is used by all remaining nodes to decide whether to reply or not. All the received responses are counted by the querying node (during a predefined time interval), knowing that it will receive a total number of replies r according to the given probability. At the end, the network size $\widehat{n}$ can be estimated at the source by $\widehat{n} = r/p$.

3.3.2 Other Approaches

A similar approach based on the same principle (sampling probability) is proposed in [58], to approximate the size of a single-hop radio network, considering the occurrence of collisions. In each step, a transmission succeeds if exactly a single station chooses to send a message. Setting a probability p to decide if to send a message at each node, the expression $np(1-p)^{n-1}$ gives the probability p_s of a step being successful. The previous expression is maximized for $p = 1/n$ where $p_s \approx 1/e$. It is expected that approximately t/e steps will be successful, if the experiment is repeated independently t times. Based on the previous probabilistic observation, the algorithm counts the number of successful steps along successive phases, to estimate the network size. Different probability values (decreasing) and number of trials (increasing) are used along each consecutive phase, until the number of successful steps is close to the expected value. Further improvements to this algorithm have been proposed in [59], aiming at making it immune against adversary attacks.

3.4 Random Walk based approaches

Random walk based approaches are usually associated with a data sampling process towards estimating an aggregation value, involving only a partial amount of network nodes. Basically, this communication process consists in the random circulation of a token. A message is sequentially sent from one node to another randomly selected neighbor – "one to one", until a predefined stopping criteria is met (e.g., maximum number of hops, reach a selected node or return to the initial one). performing a random walk across the network. Usually, a small amount of messages are exchanged in this kind of approach, since only a portion of the network is involved in the aggregation process. Due to the partial participation of the network, algorithms using this communication pattern normally rely on probabilistic methods to produce an approximation of the target aggregation function. Probabilistic methods provide result estimates with a known, bounded, error. If the execution conditions and the considered parameters of the algorithm are stable, the estimation error is expected to be maintained (with constant bounds) over time. This kind of aggregation algorithms will always depict an estimation error and do not tightly converge to the correct aggregate value.

3.4.1 Random Tour

The random tour approach [24] is based on the execution of a random walk to estimate a sum of functions of the network nodes, $\Phi = \sum_{i \in \mathcal{N}} \phi(i)$, for a generic function $\phi(i)$ where i denotes a node and $\mathcal{N}$ the set of nodes (e.g., to estimate the network size, count: $\phi(i) = 1$, for all $i \in \mathcal{N}$). The estimate is computed from the accumulation of local statistics into an initial message, all of which are gathered during a random walk, starting at an initiator node and proceeding until the message returns to it. The initiator node i initializes a counter X with the value $\phi(i)/d_i$ (where d_i denotes the degree of node i, i.e. number of adjacent nodes). Upon message reception, each node j increments the message counter X by $\phi(j)/d_j$ (i.e., $X \leftarrow X + \phi(j)/d_j$). In each iteration, the message tagged with the counter is updated and forwarded to a neighbor, chosen uniformly at random, until it returns to the initial node. When the originator receives back the message originally sent in the random walk, it computes the estimate $\widehat{\Phi}$ (of the sum Φ) by $\widehat{\Phi} = d_i X$.

3.4.2 Other approaches

Other approaches based on random walks can be found in the literature, but they are commonly tailored for specific settings and to the computation of specific aggregation functions, like COUNT (to estimate the network or group size).

For instance, to accelerate self-stabilization in a group communication system for ad-hoc networks, a scheme to estimate the group size based on random walks is proposed in [26] (first published in [25]). In this specific case, a mobile agent (called *scouter*) performs a random walk and collects information about live nodes to estimate the system size. The agent carries the set of all visited nodes and a counter associated with each one of them. Whenever the agent moves to a node, all the counters are incremented by one except for the one of the current node, which is set to zero. Large counter values are associated with nodes that have been less recently visited by the *scouter*, becoming more likely to be suspected of nonexistence. Counters are bounded by the scouter's maximum number of moves, which is set according to the expected cover time and a safety function, before considering a corresponding node as not connected. In particular, nodes are sorted in increasing order of their counter value, and they are removed from the *scouter* if the gap between successive nodes (k^{th} and $k-1^{th}$) is greater than the number of moves required to explore k connected elements in a random walk fashion. After having the *scouter* perform a large enough number of moves, the number of nodes in the system can be estimated by simply counting the number of elements kept in the set of visited nodes.

Other relevant approaches are based on the execution of random walks to collect samples, like *Sample & Collide* [27], [24] and *Capture-Recapture* [28], they are described in Section 4.5.

3.5 Gossip-based approaches

Commonly, gossip and epidemic communication are indistinctly referred. However, in a relatively recent review of gossiping in distributed systems [60] a slight distinction between the two is made. In a nutshell, the difference simply relies on the interaction directionality of both protocols. The authors state that gossiping is referred to "the probabilistic exchange of information between two members", and epidemic is referred to "information dissemination where a node randomly chooses another member". Even so, the effect of both protocols in terms of information spread is much alike, and strongly related to epidemics. For this reason, in this work no distinction will be made between gossip and epidemic protocols.

Gossip communication protocols are strongly related to epidemics, where an initial node ("infected") sends a message to a (random) subset of its neighbors ("to be contaminated"), which repeat this propagation process – "one to many". With the right parameters, almost the whole network will end up participating in this propagation scheme. This communication pattern exhibits interesting characteristics despite its simplicity, allowing a robust (fault tolerant) and scalable information dissemination all over the network, in a completely decentralized fashion. Nevertheless, it is important to outline that the robustness of gossip protocols may not be directly attained by any algorithm based on a simple application of this communication pattern. For instance, the algorithm correctness may rely on principles and invariants that may not be guaranteed by a straightforward and incautious use of a gossip communication protocol, as revealed in [61]. In general, gossip communication tends to be as efficient as flooding, in terms of speed and coverage, nonetheless it imposes a lower network traffic load. In the context of aggregation, another difference between gossip and flooding is that gossip is usually applied on a continuous basis, while flooding is issued one time to request an aggregation result. We consider that algorithms using nearest neighbor/local communication (where each node communicates with all of its local neighbors) are included in the gossip-based category.

3.5.1 Push-Sum Protocol

The push-sum protocol [29] is a simple gossip-based protocol to compute aggregation functions, such as SUM or AVERAGE, consisting in an iterative pairwise distribution of values throughout the whole network. In more detail, along discrete times t, each node i maintains and propagates information on a pair of values (s_{ti}, w_{ti}): where s_{ti} represents the sum of the exchanged values, and w_{ti} denotes the weight associated with this sum at the given time t and node i.

In order to compute different aggregation functions, it is enough to assign appropriate initial values to these variables. E.g., considering v_i as the initial input value at node i, AVERAGE: $s_{0i} = v_i$ and $w_{0i} = 1$ for all nodes; SUM: $s_{0i} = v_i$ for all nodes, only one node starts with $w_{0i} = 1$ and the remaining nodes assume $w_{0i} = 0$; COUNT: $s_{0i} = 1$ for all nodes, only one with $w_{0i} = 1$ and the others with $w_{0i} = 0$.

In each iteration, a neighbor is chosen uniformly at random, and half of the actual values are sent to the target node and the other half to the node itself. Upon receive, the local values are updated, adding each received value from a pair to its current correspondent (i.e., pointwise sum of pairs). The estimate of the aggregation function can be computed by all nodes, and is given at each time t by s_{ti}/w_{ti}.

Figure 2 illustrates the execution of this algorithm at nodes i and j to compute the AVERAGE function. Initially ($t = 0$), the initial input values are $v_i = 1$ and $v_j = 3$, the sum of exchanged values are $s_{0i} = v_i = 1$ and $s_{0j} = v_j = 3$, and the weights are set to one at all nodes $w_{0i} = w_{0j} = 1$. In this particular execution, it is considered that node i randomly chooses node j and j picks another arbitrary node. Then, half of the nodes current values, i.e. $(s_{0i}/2, w_{0i}/2)$ and $(s_{0j}/2, w_{0j}/2)$, are sent to the chosen target and themselves. Upon reception of the messages, the state at each node is updated with the sum of the received values, respectively $s_{1i} = 0.5$ and $w_{1i} = 0.5$ for

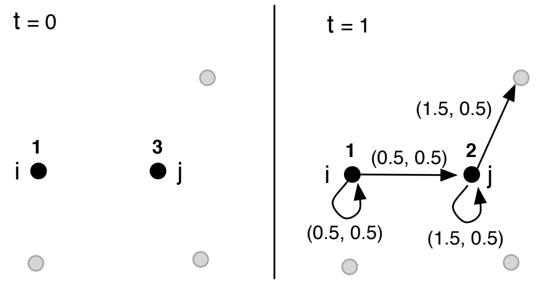


Fig. 2. Execution example of the Push-Sum Protocol.

node i, and $s_{1j} = 0.5+1.5 = 2$ and $w_{1j} = 0.5+0.5 = 1$ for node j. At the end of the depicted example, the estimate of the AVERAGE will be 1 $(0.5/0.5)$ for node i and 2 $(2/1)$ for j.

The accuracy of the produced result will tend to increase progressively along each iteration, converging to the correct value. As referred by the authors, the correctness of this algorithm relies on a fundamental property designated as *mass conservation*, stating that: the global sum of all network values (local value of each node plus the value in messages in transit) must remain constant over time. Considering the crucial importance of this property, the authors assume the existence of a fault detection mechanism that allows nodes to detect when a message did not reach its destination. In this situation, the "mass" is restored by sending the undelivered message to the node itself. This algorithm is further generalized by the authors in the same work – *push-synopses protocol*, in order to combine it with random sampling to compute more "complex" aggregation functions (e.g., quantiles) in a distributed way.

3.5.2 *Other approaches*

In the last years, several gossip-based approaches have been proposed, due to the appealing characteristics of gossip communication: simplicity, scalability and robustness. Several alternative algorithms inspired by the *push-sum protocol* have been proposed, like: *Push-Pull Gossiping* [62], [30] which provides an anti-entropy aggregation technique (see section 4.2), or *Gossip-based Generic Aggregation Protocol (G-GAP)* [63] that extended the *push-synopses protocol* to tolerate non contiguous faults (i.e., only support faults if neighbors do not fail within the same short time period).

Another aggregation algorithm supported by an information dissemination and group membership management protocol, called *newscast protocol*, is proposed in [64]. This approach consists in the dissemination of a cache of items (with a predefined size) maintained by each network node. Periodically, each node randomly selects a peer, considering the network addresses of nodes available on the local cache entries. The cache entries are exchanged between the two nodes and the received information is merged into their local cache. The merge operation discards the oldest items, keeping a predefined number of the freshest ones, also ensuring that there is at most one item from each node in the cache. An estimate of the desired aggregate can be produced by each network node, by applying the correct aggregation function to the local cache of items.

3.6 Hybrid approaches

Hybrid approaches combine the use of different communication techniques to obtain improved results from their synergy. Commonly, the use of a hierarchic topology is mixed with gossip communication. Hierarchic based schemes are efficient and accurate, but highly affected by the occurrence of faults. On the other hand, gossip based algorithms are more resilient to faults, but less efficient in terms of overhead (requiring more message exchanges). In general, this combination enables hybrid approaches to achieve a fair trade-off between performance (in terms of overhead and accuracy) and robustness, when performing aggregation in more realistic environments.

3.6.1 *(Chitnis et al., 2008)*

Chitnis et al. [39] studied the problem of computing aggregates in large-scale sensor networks in the presence of faults, and analyzed the behavior of hierarchy-based (i.e., TAG) and gossip-based (i.e., Push-Sum Protocol) aggregation protocols. In particular, they observe that tree-based aggregation is very efficient for very small failures probabilities, but its performance drops rapidly with increasing failure rates. On the other hand, a gossip protocol is slightly slowed down (almost unaffected), and is better to use with failures (compared to tree-based). Considering these results, the authors proposed an hybrid protocol with the intent of leveraging the strengths of both analyzed mechanisms and minimizing their weakness, in order to achieve a better performance in faulty large-scale sensor networks.

This hybrid approach divides the network nodes in groups, and a gossip-based aggregation is performed within each group. A leader is elected for each group, and an aggregation tree is constructed among leader nodes (multi-hop routing may be required between leaders) to perform a tree-based aggregation with the results from each gossip group. The authors also defined and solved an optimization problem to get the best combination between the two aggregation mechanisms, yielding the optimal size of the groups according to the network size and failure probability. However, in practice it requires the pre-computation of the gossip group size (by solving the referred optimization problem) before starting to use of the protocol with optimal settings. Results from simulations showed that the hybrid aggregation approach usually outperforms the other two (tree-based and gossip-based) [3].

An extension of the previous approach for heterogeneous sensor networks is later discussed in [65]. In this case, it is considered that a few distinguished nodes,

3. Notice that only static network settings (no nodes arriving or leaving) were considered by the authors.

designated as *microservers*, which are more reliable and less prone to failure than the remaining ones, are available in the network. The aggregation technique works mostly like the one previously described for the homogeneous case, but with two differences that take advantage of the reliability of *microservers*. First, microservers are preferably chosen as group leaders. Second, microservers are put on the top of the created aggregation tree that may also be composed by other less reliable motes[4]. The use of microservers in the aggregation tree increases its robustness, and by putting them at the top reduces the need the reconstruct the whole tree when a fault occurs. The obtained evaluation results show that the aggregation process can be enhanced in heterogeneous networks, when taking advantage of more reliable (although more expensive) nodes.

3.6.2 Other Approaches

A more elaborated hybrid structure was previously defined by Astrolabe [66]. Astrolabe is a DNS-like distributed management system that supports attribute aggregation. It defines a hierarchy of zones (similar to the DNS domain hierarchy), each one holding a list of attributes called Management Information Base (MIB). This structure can be viewed as a tree, each level composed by non-overlapping zones, where leaf zones are single hosts, each one running an Astrolabe agent, and the root zone includes the whole network. Each zone is uniquely identified by a name hierarchy (similarly to DNS), assigning to each zone a unique string name within the parent zone; the global unique name of each zone is obtained by concatenating the name of all its parent zones from the root with a predefined separator. The zone hierarchy is implicitly defined by the name administratively set to each agent. A gossip protocol is executed between a set of elected agents to maintain the existing zones. The MIB held by each zone is computed by a set of aggregation functions that produce a summary of the attributes from the child zones. An aggregation function is defined by a SQL-like program that have is code embedded in the MIB, being set as a special attribute. Agents keep a local copy of a subset of all MIBs, in particular of zones in the path to the root and siblings, providing replication of the aggregated information with weak consistency (eventual consistency). A gossip protocol is used for agents to exchange data about MIBs from other (sibling) zones and within its zone, and update its state with the most recent data.

Another hierarchical gossiping algorithm was introduced by Gupta et al. [67], being one of the first to use gossip for the distributed computation of aggregation functions. According to the authors, the philosophy of this approach is similar to Astrolabe, but uses a more generic technique to construct the hierarchy, called *Grid Box Hierarchy*. The hierarchy is created by assigning (random or topology aware) unique addresses to all members, generated from a known hash function. The most significant digits of the address are used to divide nodes into different groups (grid boxes) and define the hierarchy. Each level of the hierarchy corresponds to a set of grid boxes, matching a different number of significant digits. The aggregation process is carried out from the bottom to the top of the hierarchy in consecutive gossip phases (for each level of the hierarchy). In each phase: members of the same grid box gossip their data, compute the resulting aggregate after a predefined number of rounds, and then move to the next phase. The protocol terminates when nodes find themselves at the grid box at the top of the hierarchy (last phase). Note that, this approach does not rely on any leader election scheme to set group aggregators, in fact the authors argue for the inadequacy of such mechanism in unreliable networks prone to message loss and node crashes.

Recently, an approach that combines a hierarchy-based technique with random sampling was proposed in [68] to approximate aggregation functions in large WSN. This approach regulates the amount of collected data by a sampling probability, aiming at reducing the energy consumption to compute the aggregate. The sampling probability is calculated from the input accuracy expressed by two parameters ε and δ, i.e. relative error less than ε with probability greater than $1 - \delta$, and the aggregation function, i.e. COUNT, SUM or AVERAGE. This algorithm considers that the sensing nodes are organized in clusters, according to their geographic location, and that cluster heads form a spanning tree rooted at the sink. Basically, the aggregation proceeds as following: first, the sink computes the sampling probability p (according to ε and δ) and transmits it along with the aggregation function to all cluster heads across the spanning tree; then, cluster heads broadcast p to their cluster and each node within it independently decides to respond according to the received probability; samples are collect at each cluster head which computes a partial result; finally, the partial results are aggregated upward the tree (convergecast) until the sink is reached, and where the final approximate result is computed. This algorithm, referred by the authors as Bernoulli Sampling on Clusters (BSC), mixes the application of a common hierarchy based aggregation techniques such as *TAG* (see Section 3.1) between cluster heads, with a flooding/broadcast method like *Randomized Reports* (see Section 3.3) to sample the values at each cluster.

4 Computation Taxonomy

In terms of the computational principles in which the existing aggregation algorithms are based on, the following main categories (see Table 4) were identified: *Hierarchical*, *Averaging*, *Sketches (hash or min-k based)*, *Digests*, *Deterministic*, and *Randomized*. These categories intrinsically support the computation of different kind of aggregation functions, as depicted in Table 3. For instance, *Hierarchical* approaches allow the computation of

4. Alternative designation of a wireless sensor node [14].

TABLE 3
Functions computed by aggregation technique

	Decomposable		Non Decomposable	
	DS	**DI**	**DS**	**DI**
Hierarchical	x	x		
Averaging	x			
Sketches	x			x
Digests	x	x	x	x
Deterministic	COUNT			
Randomized	COUNT			

DS: Duplicate Sensitive; DI: Duplicate Insensitive;

TABLE 4
Taxonomy from the computation perspective

Aggregation	**Basis/Principles**		**Algorithms**
Decomposable Functions	*Hierarchic*		TAG [14], DAG [15], I-LEAG [16], Tributary-Delta [20], (Chitnis et al., 2008) [39]
	Averaging		Push-Sum Protocol [29], Push-Pull Gossiping [30], DRG [31], Flow Updating [32], [33], (Chitnis et al., 2008) [39]
	Sketches		Sketches [17], RIA-LC/DC [18], [19], Extrema Propagation [34], Tributary-Delta [20]
Complex Functions	*Digests*		Q-Digest [21], Equi-Depth[35], Adam2 [36]
Counting	*Deterministic*		(Dolev et al., 2002) [25], [26]
	Randomized	*Sampling*	Random Tour [24], Randomized Reports [23], Sample & Collide [27], [24], Capture-Recapture [28], Hop-Sampling [37], [38], Interval Density [37], [38], (Kutylowski et al., 2002) [58]
		Estimator	(Horowitz and Malkhi, 2003) [22]

any decomposable function. *Averaging* techniques allow the computation of all duplicate sensitive decomposable functions that can be derived from the AVERAGE, by using specific initial input values and combining the results from different instances of the algorithms. *Sketches* techniques also allow the computation of duplicate sensitive decomposable functions that can be derived from the SUM function[5].

Moreover, schemes based on hash sketches are natively able to compute distinct counts (non decomposable duplicate insensitive), and those based on min-k can be easily adapted to compute it (e.g., in *extrema propagation*, see 4.3, using the input value as seed of a pseudo-random generation function, so that duplicate values will generate the same number). *Digests* support the computation of any kind of aggregation function, as this type of approach usually allows the estimation of the whole data distribution (i.e., values and frequencies) from which any function can be obtained. On the other hand, some techniques are restricted to the computation a single type of aggregation function, such as COUNT, which is the case of the *Deterministic*, and *Randomized* approaches.

Besides determining the supported aggregation function, the computational technique on which an aggregation algorithm is based constitutes a key element when defining its behavior and performance, especially in terms of accuracy and reliability. *Hierarchical* approaches are accurate and efficient in terms of message and computational complexity, but not fault tolerant. *Averaging* schemes are more reliable and also relatively accurate (converge over time), although less efficient, requiring more message exchanges. Approaches based on the use of *sketches* are more reliable than hierarchical schemes, adding some redundancy and providing fast multi-path data propagation, however they introduce an approximation error, depending on the number of inputs and size of the used sketch. *Digests* essentially consist in the reduction (compression) of all inputs into a fixed size data structure, using probabilistic methods and losing some information. Consequently, digests provide an approximation of the computed aggregation function, not the exact result. *Randomized* schemes are also based on probabilistic methods to compute the COUNT, mostly by sampling and thus being inaccurate and lightweight in terms of message complexity, as only a portion of the network is asked to participate. On the other hand, *deterministic* counting methods require gathering data from the entire network, which can incur in scalability issues.

In the following sections, the main principles and characteristics of these distinct classes are explained in a comprehensive way, and some important examples are described. A taxonomy of the identified computational principles is displayed in Table 4, associating them to the most relevant distributed aggregation algorithms.

4.1 Hierarchical

Hierarchical approaches take direct advantage of the decomposable property of some aggregation functions. Inputs are divided into separate groups and the computation is distributed and hierarchical. Algorithms from

5. Note that, COUNT is the sum of all elements considering their input value as equal to 1.

this class depend on the initial creation of a hierarchic communication structure (e.g., tree, cluster hierarchy), where nodes can act as *forwarders* or *aggregators*. Forwarders simply transmit the received inputs to an upper level node. Aggregators apply the target aggregation function to all received input (and its own), and forward the result to an upper level node. The correct result is yield at the top of the hierarchy, being the aggregation process carried out from the bottom to the top.

Algorithms from this class allow the computation of any decomposable function, providing the exact result (at a single node) if no faults occur. The global processing and memory resources required are equivalent to the ones used in a direct and centralized application of the aggregation function, but distributed across the network. However, these algorithms are not fault tolerant, e.g. a single point of failure may lead to the loss of all data beneath it.

Most of the algorithms from this category correspond to the ones belonging to the hierarchical communication class, like TAG [14], DAG [15], and I-LEAG [16]. Other algorithms can be found combining a hierarchical computation with another computation principle, namely: Tributary-Delta [20] mix a common hierarchical computation, with the use sketches in regions close to the *sink*; (Chitnis et al., 2008) [39] performs hierarchic aggregation on the top of groups, and averaging is applied inside each one. See sections 3.1 and 3.6 for more details about the aforementioned algorithms.

4.2 Averaging

The *Averaging* class essentially consists in the iterative computation of partial aggregates (averages), continuously averaging and exchanging data among all active nodes that will contribute to the calculation of the final result. This kind of approach tends to be able to reach a high accuracy, with all nodes converging to the correct result along the execution of the algorithm. A typical application of this method can be found in most gossip-based approaches (section 3.5), where all nodes continuously distribute a share of their value (averaged from received values) with some random neighbor, converging over time to the global network average, the correct aggregation result. Algorithms from this category are more reliable than hierarchic approaches, working independently from the supporting network topology and producing the result at all nodes. However, they must respect an important principle, commonly designated as "mass conservation" in order to converge to the correct result. This invariant states that the sum of the aggregated values of all network nodes must remain constant over time [29].

Algorithms based on this technique are able to compute decomposable and duplicate-sensitive functions, which can be derived from the average operation; using different input initializations (e.g., COUNT), or combining functions executed concurrently (e.g., SUM, obtained by multiplying the results from an average and a count). In terms of computational complexity, this method usually involves the computation of simple arithmetic operations (i.e., addition and division), using few computational resources (processor and memory) and thus depicting a fast execution at each node. This kind of algorithm is able to produce (almost) exact results, depending on their execution time. The minimum execution time required by these algorithms, in number of iterations, to achieve a high accuracy, is influenced by the network characteristics (i.e., size, degree of the network graph, and topology) and the communication pattern used to spread the partial averages. The robustness of these types of aggregation algorithms is strongly related with their ability to conserve the global "mass" of the system. The loss of a partial aggregate ("mass") due to a node failure or a message loss introduces an error, resulting in the subtraction of the lost value to the initial global "mass" (leading to the non-contribution of the lost amount to the calculation of the final result, and therefore to the convergence to an incorrect value). In this kind of methodology, it is important to enforce the uphold of the "mass" conservation principle, assuming itself as a main invariant to ensure the algorithms correctness.

This class of algorithms is also known as *Distributed Average Consensus* in the area of Control Theory [69], [70], [71], [72], where the studied algorithms mainly operate like the *Push-Sum Protocol* (see Section 3.5) and *Push-Pull Gossiping*. This field provides interesting theoretical results on the performance of averaging techniques (e.g., optimized weights and convergence rates), but commonly consider impractical assumptions (e.g., assuming instantaneous update of the topology information held at each node).

4.2.1 Push-Pull Gossiping

The push-pull gossiping [62] algorithm performs an averaging process, and it is gossip-based like the *push-sum protocol [29]* (previously described in section 3.5). The main difference in this scheme relies on the execution of an anti-entropy aggregation process. The concept of anti-entropy in epidemic algorithms is related to the regular random selection of another node and the performance of a mutual synchronization of information, exchanging complete databases [73]. In particular, this algorithm executes an epidemic protocol to perform a pairwise exchange of aggregated values between neighbor nodes. Periodically, each node randomly chooses a neighbor to send its current value, and waits for the response with the value of the neighbor. Then, it averages the sent and received value, and calculates the new estimated value. Each time a node receives a value from a neighbor, it sends back its current one and computes the new estimate (average), using the received and sent values as parameters.

Figure 3 illustrates the push-pull process performed by the algorithm. In this particular example, two nodes

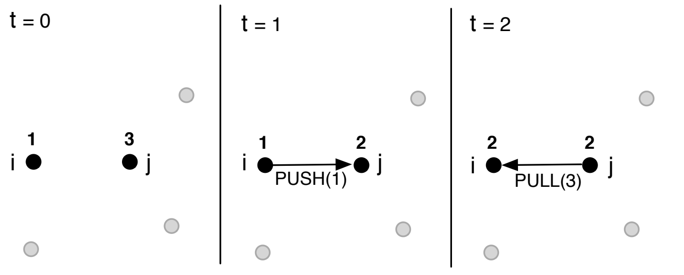


Fig. 3. Execution example of Push-Pull Gossiping.

are considered i and j, respectively with an initial value $v_i = 1$ and $v_j = 1$. Then, it is assumed that node i randomly chooses j to execute the push-pull process, and it sends him a *push* message with its current value 1. Upon reception of the push message, j responds to i with a *pull* message with its current value 3 and it immediately updates its estimate with the average of the received and sent value, i.e. $v_j = (1+3)/2 = 2$. Upon reception of the pull message, node i will also update its estimate by averaging the received and current values, i.e. $v_i = (3+1)/2 = 2$.

In order to be adaptive and handle network changes (nodes joining/leaving), the authors consider the extension of the algorithm with a restart mechanism (executing the protocol during a predefined number of cycles, depending on the desired accuracy, and then restarting a new cycle with fresh initial values). However, they do not address the "mass" conservation problem – impact of message losses or node failures.

A further study of this aggregation algorithm is discussed in [30], proposing a more mature solution that covers some practical issues: splitting the algorithm execution in two distinct threads; use of timeouts to detect possible faults, ignoring data exchanges in those situations; suggesting different versions of the algorithm according to the aggregation function to compute; suggesting the execution of several parallel instances of the algorithm to increase its robustness.

4.2.2 DRG (Distributed Random Grouping)

This approach [31] essentially consists in the continuous random creation of groups across the network, in which aggregates are successively computed (averaged). DRG was designed to take advantage of the broadcast nature of wireless transmission, where all nodes within radio range will be prone to hear a transmission. Thus, this approach is specific for WSN. The algorithm defines three different working modes for each node: *leader*, *member*, and *idle* mode. According to the defined modes and the performed state transitions, the execution of the algorithm can be separated in three main steps. First, each node in idle mode independently decides to become a group leader (according to a predefined probability), and consequently broadcast a Group Call Message (GCM) to all its neighbors, subsequently waiting for members to reply. Second, all nodes in idle mode

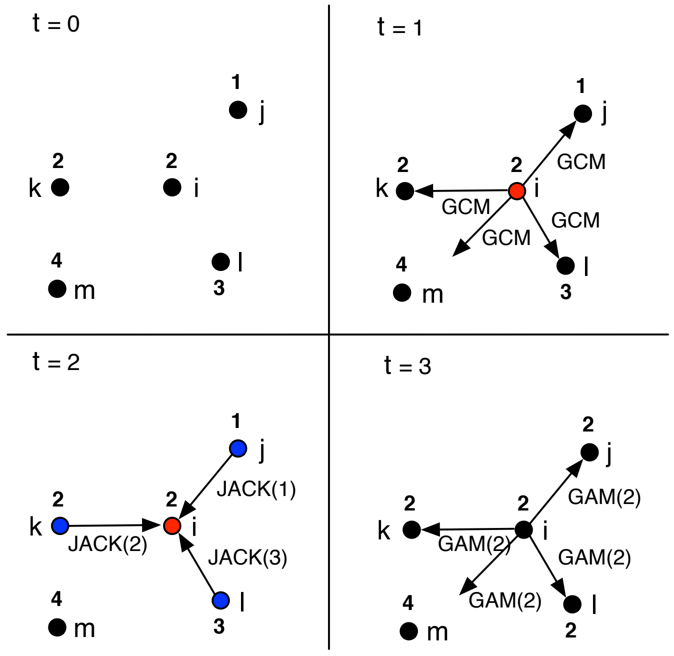


Fig. 4. Execution example of Distributed Random Grouping.

which received a GCM from a leader respond to the first one with a Joining Acknowledgment (JACK) tagged with their aggregate value, becoming members of that group (updating their state mode accordingly). In the third step, after gathering the group members' values from received JACKs, the leader computes (averages) the group aggregate and broadcasts a Group Assignment Message (GAM) with the result, returning to idle mode. Each group member waits until it receives the resulting group aggregate from the leader to update its local value (with the one assigned in the GAM) and returns to idle mode, not responding to any other request until then.

Figure 4 illustrates the execution of the algorithm to compute the AVERAGE. In this particular example, node i decides to become leader and sends a GCM to all its neighbors. In the considered scenario, only nodes j, k and l receive the message and reply to i with a JACK message with their respective current value $v_j = 1$, $v_k = 2$ and $v_l = 3$. After receiving JACK messages from its neighbors, the leader i computes the group average from the received values and its current value $v_i = 2$, i.e. $a = (1+2+3+2)/4 = 2$. Then, the leader updates its value with the resulting average and sends a GAM with the result to its neighbors so that the group members can also update their value with the new average.

The repeated execution of this scheme – creation of distributed random groups to perform in-group aggregation – allows the eventual convergence of the estimate produced at all nodes to the correct aggregation result, as long as the groups overlap over time. The performance of DRG is influenced by the predefined probability of a node becoming a leader, which determines its capacity to create groups (quantity and size of groups). Note that, in order to account for the occurrence of faults and avoid consequent deadlock situations that could arise in this algorithm, it is necessary to consider the definition of some timeouts (for the leaders to wait for JACKs, and the members to wait for a GAM). Intuitively, one will notice that the values set for those timeouts will highly influence the performance of the algorithm, although this detail is not addressed by the authors. An analysis of DRG on WSN with randomly changing graphs (modeling network dynamism) is provided in [74], assuming that the graph only changes at the beginning of each iteration of the algorithm. Unrealistic assumption in practice, and a potential source of mass loss.

4.2.3 *Flow Updating*

Flow Updating [32] is a recent aggregation technique that is inspired in the concept of network flow (from graph theory). Unlike common averaging approaches, that start with the initial input value and iteratively change it by exchanging "mass" along the execution of the algorithm, this approach keeps the initial input unchanged, exchanging and updating flows associated to neighbor nodes. The key idea is to explore the concept of flow, and instead of storing the current average at each node in a variable compute it from the input value and the contribution of the flows along the edges to the neighbors. In a sense, flows represent the value that must be transferred between two adjacent nodes for them to produce the same estimate, and are skew symmetric (i.e., the flow value from i to j is the opposite from j to i: $f_{ij} = -f_{ji}$). For example: considering a network with two directly connected nodes i and j with initial input values $v_i = 1$ and $v_j = 3$, for them to produce the same average $a = \frac{1+3}{2} = 2$, the flows at node i and j must be respectively set to $f_{ij} = v_i - a = 1 - 2 = -1$ and $f_{ji} = v_j - a = 3 - 2 = 1 = -f_{ij}$.

In more detail, each node i stores a flow value f_{ij} to each neighbor j, besides its input value v_i which is unchanged by the algorithm. Periodically, each node computes a new estimate e'_i by averaging the ones received from neighbors e_j with its own e_i (obtained by subtracting all flows to its input value). The flows f_{ij} to each neighbor j are then locally updated in order to produce the new estimate result, adding the difference between the new estimate and the one previously received to the respective flow value. Afterwards, the node sends in a message the flows f_{ij} to each neighbor j, as well as the new estimate. Upon reception of a message, node j updates its flow f_{ji} with the symmetric value addressed to him $-f_{ij}$, and keeps the received estimate to further compute the next average. The iterative execution of this process across the whole network allows the estimate of all nodes to converge to the global average of the input values.

This approach distinguishes itself from the existing averaging algorithms by its fault-tolerant capabilities. It solves the mass conservation problem observed on other averaging approaches when subject to message loss, that affect their correctness leading them to converge to a wrong value. Other approaches require additional mechanism to detect and restore the lost mass, which is often not feasible in practice. In contrast, Flow Updating is by design able to support message loss, only delaying convergence without affecting the convergence to the correct value, without requiring any additional mechanism. This is achieved by keeping the input values unchanged and performing idempotent flow updates which guarantee their skew symmetric properties. Moreover, it has been shown that this approach is resilient to node crash and able to support churn (without requiring protocol restarts), self-adapting to network changes [33]. Recently, a variation of this technique called Mass Distribution with Flow Updating (MDFU) was proposed [75], that provides a convergence proof and characterization of the convergence time under stochastic message loss.

4.2.4 *Other Approaches*

A well-known averaging approach, the Push-Sum (push-synopses) Protocol [29] has already been described in Section 3.5. In the last years, other approaches inspired in the Push-Sum Protocol have been proposed, intending to be more efficient in terms of performance and robustness. Kashyap et al.[76] reduced the number of messages needed (communication overhead) to compute an aggregation function at the cost of an increase in the number of rounds. Gossip-based Generic Aggregation Protocol (G-GAP) [63] extended the *push-synopses protocol* [29] to support discontinuous failures (no adjacent node can fail within a period of 2 rounds) by restoring the mass loss resulting from failures (temporarily storing at each node previous data contributions). Recently, *LimoSense* [77] introduced effective strategies to avoid mass loss. However, the mechanism that balances mass growth can lead to divisions by zero under some message sequences.

Dimakis et al. [78], [79] proposed an algorithm to improve the convergence time in random geometric networks. This scheme is similar to push-pull gossiping [62], differing in the peer selection methods. Instead of selecting a one-hop node as target of the averaging step, peers are selected according to their geographical location. In particular, a location is randomly chosen and the node closer to that local is selected. A greedy geographic routing process is used to reach the node at the target location, assuming that nodes known their own geographic location.

Two averaging algorithms for asynchronous and dynamic networks were proposed in [80]. The core of the proposed schemes is based on a pairwise update, similarly to the push-pull gossiping (although not referred by the authors), addressing practical concerns that arise in asynchronous settings. In the first proposed algorithm nodes implement a blocking scheme to avoid the interference of other nodes in the update step and guarantee mass conservation. Additionally, a deadlock avoidance mechanism is considered, by imposing a sender-receiver relation on each link based on nodes Unique Identifiers (UID). An extension to the first algorithm is proposed to cope with churn. The blocking mechanism (maintaining the directed relationship between nodes) is removed, and an additional variable is used to account for changes in each neighbor. When a node leaves the network, all its neighbors subtract the value associated with it from their state.

4.3 Sketches

The main principle in this kind of aggregation algorithm is the use of an auxiliary data structure with a fixed size, holding a *sketch* of all network values. The input values of each node are used to create sketches that are aggregated across the network, using specific operations to update and merge them. Sketches are order and duplicate insensitive, enabling them to be aggregated through multiple paths, being independent from the routing topology. This kind of technique is based on the application of a probabilistic method, generally allowing the estimation of the sum of the values held in the sketch.

Sketching techniques can be based on different methods, with different accuracy bounds and computational complexities. Algorithms from this class are mostly based on the application (with some improvements) of two main ideas: *hash sketches* [41], [81], [82], [83] and *k-mins sketches* [84].

Hash sketches allow the probabilistic counting of the number of distinct elements in a multiset (cardinality of the support set). This type of sketch essentially consists in a map of bits, initially set to zero, where each item is mapped into a position in the binary valued map (generally involving a uniform hashing function) setting that bit to one. The distinct count is estimated by checking the position of the most significant bit that is set to one (leftmost), or counting the number of bits that are set to zero in the sketch. The first hash sketching technique was proposed by Flajolet and Martin [41], and is commonly designated as FM sketches (uniformly hashes items into an integer, and maps only the less significant bit 1 of its bitmap representation to the sketch). In this first study, the authors also proposed the Probabilistic Counting with Stochastic Averaging (PCSA) algorithm to reduce the variance of the produced estimate, using multiple sketches and averaging their estimate (distributing the hash of an element to only one of the sketches). Another approach, Linear Counting [81] uses a hash function to directly map each element into a position of the sketch (setting that bit to 1), and use the count of the number of zeros to produce an estimate. A further improvement to PCSA, designated LogLog, was described in [82], reducing required memory resources (an optimized version super-LogLog is also proposed, improving accuracy and optimizing memory usage applying a truncation and restriction rule). HyperLogLog [83] recently improved LogLog, consuming less memory to achieve a matching accuracy.

The k-mins sketches method was first introduced to determine the size of the transitive closure in directed graphs [84]. It consists of assigning k independent random ranks to each item according to a distribution that depends on its weight, and keeping in a vector of the minimum ranks in the set. The obtained k-vector of the minimum ranks is used by an estimator to produce an approximate result. In other words, it can be said that k-mins sketches reduce the estimation of the sum to the determination of minimums of a collection of random numbers (generated using the sum operands as input parameter of the random distribution from which they are drawn). An improved alternative to k-mins sketches, designated bottom-k sketches, was recently proposed in [85].

The computational cost of sketching is dependent on the complexity of the operations involved in the creation and update of the sketches (e.g., hash functions, random number generation, minimum/maximum determination), and the resources used by the estimator to produce a result. Algorithms based on sketches are not accurate, being based on probabilistic methods and introducing an error factor in the computed aggregation function. There is a trade-off between the accuracy and the size of the sketches. The greater the sketch size the tighter are the accuracy bounds of the produced estimate, although requiring additional memory resources and a larger processing time. These kind of aggregation algorithms tend to be fast, although conditioned by the dissemination protocol used to propagate the sketches, being able to produce an approximate result after a number of iterations close to the minimum theoretical bound (the network diameter).

4.3.1 *RIA-LC/DC*

Fan and Chen [18] proposed a multi-path routing aggregation approach for WSN based on the use of Linear Counting (LC) sketches [81], which they later named Robust In-network Aggregation using LC-sketches (RIA-LC) [19]. The algorithm proceeds in two phases, similar to those in multipath hierarchy-based approaches (see Section 3.1). In the first phase, the aggregation request (query) is spread from the sink throughout the whole network, creating a multipath routing hierarchy. In the second phase, starting at the lower level of the hierarchy, nodes respond to the aggregation request by creating a LC-sketch corresponding to its current local readings and sending it to the nodes at the upper level. All received sketches are combined with the local one (using the OR operation), and the result is sent to the next level until the top of the hierarchy is reached, where the sink computes the aggregate estimate from the resulting LC-sketch.

Equation 1 is used to estimate the number of distinct items represented in a LC-sketch, where m is the size of the allocated bit vector, and z is the count of the number of bits with value equal to zero. In order to allow the computation of the SUM, each node creates a sketch by mapping a number of distinct items corresponding to its input value. For example, assuming that each node has a unique ID, if the node i has an input equal to 3, it maps the items $(ID_i, 1)$, $(ID_i, 2)$, and $(ID_i, 3)$ into the LC-sketch. In more detail, in this case the use of a hash function from the original LC-sketch design (to map duplicated items to the same bit) is replaced by a uniform random generator (since there are no duplicate items), randomly setting to 1 a number of bits equal to the input value.

$$\hat{n} = -m \ln (z/m) \tag{1}$$

The authors show by theoretical comparison and experimental evaluation that their approach outperforms, in terms of space and time requirements, approaches based on FM sketches [41], namely Sketches [17] (see 3.1). They also claim a higher accuracy and lower variance when compared with existing sketch schemes. Moreover, they tackle some practical issues, like message size constraints, avoiding the use of hash functions, and enabling the specification of an approximation error.

Afterwards, the authors improved RIA-LC by considering the use of sketches with variable sizes instead of fixed size sketches, referring to the new technique as Robust In-network Aggregation using Dynamic Counting sketches (RIA-DC) [19]. The authors observed that the large preallocated sketches used in RIA-LC were wasting space, since at the beginning of the computation must of the bits are set to zero. In RIA-DC the initial size of sketches is variable and depends on the local sensor reading. Along the aggregation process the size of the sketches is adjusted (gradually increasing toward the sink), in order to satisfy a given accuracy constrain. RIA-DC decreases message overhead and energy consumption compared to RIA-LC, while keeping similar accuracy properties.

Recently, another variant of the described schemes was proposed by the same authors, referred as *Scalable Counting* (SC) [86], that reduces the space requirements (i.e., size of the transmitted sketches) to obtain the same accuracy guarantees.

4.3.2 Extrema Propagation

This approach reduces the computation of an aggregation function, more precisely the sum of positive real numbers, to the determination of the minimum (or maximum) of a collection of random numbers [34], [87]. Initially, a vector x_i of k random number is created at each network node i. Random numbers are generated according to a known random distribution (e.g., Exponential or Gaussian), using the node initial value v_i as the input parameter for the random generation function (e.g., as the rate of an exponential distribution). Then, the execution of the aggregation algorithm simply consists in the computation of the pointwise minimum (or alternatively maximum) between all exchanged vectors. This technique supports the use of any information spreading algorithm as a subroutine to propagate the vectors, since the calculation of minimums is order and duplicate insensitive. In particular, the authors consider that at each round all nodes send their resulting vector to all their neighbors.

At each node, the obtained vector is used as a sample to produce an approximation of the aggregation result, applying a maximum likelihood estimator derived from the extreme value theory (branch of statistics dealing with the extreme deviation from the median of a probabilistic distribution). For example, considering the generation at each node of k random numbers with an exponential distribution of rate v_i and the use of the minimum function to aggregate the vectors. Equation 2 gives the estimator for the SUM of all v_i from the sample of minimums $x_i[1], ...x_i[k]$ in the vector x_i, with variance $\text{SUM}^2/(k-2)$:

$$\widehat{\text{SUM}} = \frac{k-1}{\sum_{j=1}^{k} x_i[j]} \tag{2}$$

This algorithm is focused on obtaining a fast estimate, rater than an accurate one. Although, the accuracy of this aggregation algorithm can be improved by using vectors of larger size, adjusting k to the desired relative accuracy (e.g., $k = 387$ for a maximum relative error of 10%). A further extension to the protocol to allow the determination of the network diameter has been proposed in [88]. In addition to the use of minimal or maximal values, other estimators can also be designed from the average or range of suitable distributions [89].

4.3.3 Other Approaches

A representative approach based on FM sketches has already been described in section 3.1 – Sketches [17]. In this multi-path approach, a generalization of PCSA is used to distinguish the same aggregates received from multiple paths, and subsequently manage to compute duplicate-sensitive aggregation functions. Other similar approaches, based on hash sketches, can be found in the literature: like *Synopsis Diffusion* [43] and Wildfire [90]. These approaches apply essentially the same aggregation process, operating in two phases (request/response) and only differing on small details.

Synopsis Diffusion [43] is an aggregation approach for WSN close to the one proposed by Sketches [17]. In a sense, this work presents a more generic framework relying on the use of duplicate insensitive summaries (i.e., hash sketches), which they called Order- and Duplicate-Insensitive (ODI) synopses. They generically define the synopses functions (i.e., generation, fusion and evaluation) required to compute aggregation functions, and provide examples of ODI synopses to compute more "complex" aggregates (i.e., not decomposable aggregation functions). For instance, besides the scheme based on FM sketches, they propose other data structures (and respective functions) to uniformly sample sensor readings and compute other samplings based aggregation functions. The authors also tackled additional practical concerns. Namely, they explored the possibility to implicitly acknowledge ODI synopses to infer messages losses, and suggested simple heuristics to modify the established routing topology (assigning nodes to another hierarchic level), in order to reduce loss rate.

Wildfire [90] is based on the use of FM sketches to estimate SUM, but it is targeted at dynamic networks. Despite the fact of operating in two phases like previous hash sketch approaches, and unlike them, it does not establish any specific routing structure to aggregate sketches. After receiving the query, nodes start combining the received sketches with their current one, and then send the result if it differs from the previous one.

A distributed implementation of some basic hash sketches schemes has been proposed in [91], [92]. Distributed Hash Sketches (DHS) is supported by a DHT, taking advantage of the load balancing properties, and scalability of such structure. More specifically, the authors describe how to build DHS based on PCSA [41] and supper-LogLog [82].

Mosk-Aoyama and Shah [93], [94] proposed an algorithm, called COMP, to compute the sum of values from individual function (referred as separable functions). This algorithm is very similar to *Extrema Propagation* but less generic, as it is restricted to the properties of exponential random variables distribution. Furthermore, COMP uses a biased estimator, being less accurate for small sketch sizes than Extrema Propagation that uses unbiased ones.

4.4 Digests

This category includes algorithms that allow the computation of more complex aggregation functions, like quantiles (e.g., median) and frequency distributions (e.g., mode), in addition to common aggregation functions (e.g., count, average and sum). Basically, algorithms from this class produce a *digest* that summarizes the system data distribution (e.g., histogram). The resulting *digest* is then used to approximate the desired aggregation functions. We refer to a *digest* as a data structure with a bounded size, that holds an approximation of the statistical distribution of input values in the whole network. This data structure commonly corresponds to a set of values or ranges with an associated counter.

Digests provide a fair approximation of the data distribution, not holding an exact representation of all the system values for efficiency and scalability reasons. The accuracy of the result yield from a digest depends on its quality (i.e., used data representation) and size. Digests allow the computation of a wider range of aggregation functions, but usually require more resources and are less accurate than the other more specialized approaches.

4.4.1 Q-Digest

An aggregation scheme that allows the approximation of complex aggregation function in WSN is proposed in [21]. This approach is based on the construction and dissemination of q-digests (quantile digests) along a hierarchical routing topology (without routing loops and duplicated messages). A q-digest consists of a set of buckets, hierarchically organized, and their corresponding count (frequency of the values contained by the bucket). Buckets are defined by a range of values $[a, b]$ and can have different sizes, depending on the distribution of values they represent. Each node maintains a q-digest of the data available to it (from its children). Q-digests are built in a bottom-up fashion, by merging received digests from child nodes, and further compressing the resulting q-digest according to a specific compression factor (less frequent values are grouped in large buckets). Aggregation functions are computed by manipulating (e.g., sorting q-digest nodes) and traversing the q-digest structure according to specific criteria that depends on the function to be computed.

The authors provide an experimental evaluation, where they showed that q-digest allows the approximation of quantile queries using fixed message sizes, saving bandwidth and power when compared to a naive scheme that collects all the data. The naive scheme obtains an exact result, but with increasing message size along the routing hierarchy. Obviously, there is a trade-off between the obtained accuracy and the message size used. The authors suggested a way to compute the confidence factor associated with a q-digest (i.e., the error associated to a query), but the effect of faults was not considered in their study.

4.4.2 Equi-Depth

A gossip-based approach to estimate the network distribution of values is described in [35]. This scheme is based on the execution of a gossip protocol and the application of specific merge functions to the exchanged data, to restrict storage and communication costs. In more detail, each node keeps a list of k values (digest), initially set with its input value. At each round, nodes get the list of values from a randomly chosen neighbor and merge it with its own, applying a specific procedure. The results from the execution of several rounds produce an approximation of the network distribution of values (i.e., histogram). Four merging techniques were considered and analyzed by the authors: *swap*, *concise counting*, *equi-width histograms*, and *equi-depth histograms*.

Swap simply consists in randomly picking k values from the two lists (half from each one of them) and discarding the rest. Although simple, by discarding half of the available data in each merge important information is likely to be lost.

Concise counting associates a tuple, value and count, to each list entry. The merge process consists in sorting the tuples (by value), and individually merging the tuples with the closest values, in order to keep a fixed list size. Tuples are merged by randomly choosing one of the values and adding their counts.

The equi-width technique breaks the range of possible values into bins of equal size, associating a counter to each one. Initially, nodes consider the range from 0 to the current input value, as the extremes are not known. Bins are dynamically resized when new extremes are found: all bins are mapped into larger ones, based on their middle value and the range of the new bin, adding their counter to the new mapped bin. This technique requires only the storage of the extreme values and counts, since all bins have an equal width, reducing the volume of data that needs to be stored and exchanged when compared to other techniques (e.g., concise counting). However, equi-width can provide very inaccurate results for severely skewed distributions.

In equi-depth, bins are divided not to be of the same width but to contain approximately the same count. Initially, fixed size bins are set, each represented by a pair $\langle$value, counter$\rangle$, dividing the range from 0 to the input value. Whenever data is exchanged, all pairs (received and owned) are ordered, and consecutive bins that yield the smallest combined bins (in terms of count) are merged, repeating the process until the desired number of bins is obtained. Bin merging consists in adding the counters and using the arithmetic weighted mean as value. This method intends to minimize the counting disparity across bins.

In order to deal with changes of the nodes input values over time, the authors consider the execution of the protocol in phases, restarting it. The authors experimentally evaluated their protocol comparing the previous merging techniques. The obtained results showed that equi-depth outperformed the other approaches, providing a consistent trade-off between accuracy and storage requirements for all tested distributions. The authors also evaluated the effect of duplicates, from the execution of the gossip protocol. They argue from the obtained results that although duplicates bias the estimated result, it is more advantageous (simpler and efficient) to assume their presence than trying to remove them. The occurrence of faults and change of the input values were not evaluated.

4.4.3 Adam2

Adam2 is a gossip-based algorithm to estimate the statistical distribution of values across a decentralized system [36]. More precisely, this scheme approximates the Cumulative Distribution Functions (CDF) of an attribute, which can then be used to derive other aggregates. In this case, a "digest" is composed by a set H_i of k pairs of values (x_k, f_k), where x_k represents an interpolation point and f_k is the fraction of nodes with value less or equal than x_k. At a high abstraction level, it can be said that the algorithm simply executes several instances of an averaging protocol (i.e., Push-Pull Gossiping [30]) to estimate the fraction of nodes in each pair of the CDF.

In more detail, each node can decide to start an instance of Adam2 according to a predefined probability $\frac{1}{\hat{n}_i R}$, where $\hat{n}_i$ is the current network size estimate at node i and R is an input parameter that regulates the aggregation instances frequency (i.e., on average one every R rounds). Each instance is uniquely identified by its starting node. Initially, the starting node i initializes the interpolation set H_i in the following way: fractions f_k are set to 1 if the node attribute reading v_i is less or equal than the corresponding interpolation value x_k, otherwise it is set to 0. Nodes store a set of interpolation points H_i for each running algorithm instance (initiated by a node i). Upon learning about a new instance, a node j initializes H_i setting $f_k = 1$ if $a_j \leq x_k$ and $f_k = 0$ otherwise, and starts participating in the protocol. A push-pull like aggregation is then performed, where nodes randomly choose a neighbor to exchange their set H_i, which are subsequently merged by averaging the fractions at each interpolation point. Over time, the fractions will eventually converge at each node to the correct result associated to each pair. After a predefined number of rounds (*time-to-live*), the CDF is approximated by interpolating the points of the resulting set H_i.

Note that, Adam2 concurrently estimates (by averaging) other aggregation functions besides CDF, namely COUNT to determine the network size, and MIN/MAX to find the extreme attribute values. The result from these aggregation functions are later used as input values of the next instances of the algorithm to tune and optimize its execution (i.e., calculate the instance starting probability, and set new interpolation points).

Like in Push-Pull Gossiping [62], [30], Adam2 handles dynamism (i.e., attribute changes and churn) by continuously starting new instances of the algorithm – restart mechanism. The authors evaluated the algorithm by simulation, comparing it with previous techniques to compute complex aggregates (e.g., *Equi-Depth*). The obtained results showed than Adam2 outperforms the compared approaches, exhibiting better accuracy.

4.4.4 Other Approaches

One of the first algorithms to compute complex aggregation functions in WSN was introduced by Greenwald and Khanna [95]. Their approach is similar to the one previously described for Q-Digest: nodes compute quantile summaries (digest) that are merged in a bottom-up fashion along a tree topology, until the root is reached.

Another gossip-based scheme to estimate the distribution of input readings, and able to detect outliers, was introduced in [96], [97]. In a nutshell, this approach operates like the push-sum protocol [29] (described in Section 3.5), but manipulates a set of clusters (digests) instead of a single value, applying a specific clustering procedure.

Recently, a novel algorithm named *Spectra* [98] was introduced to estimate the distribution function of a value (more precisely its CDF). Basically, this approach works similarly to Adam2 but replaces the Push-Pull Gossiping [30], the technique at is core, by the Flow Updating [33] scheme. Spectra provides an improved performance and robustness when compared to Adam2, inheriting the characteristic of the flow updating mechanism, supporting high-levels of message loss and self-adapting to network change (without requiring restarts).

In general, existing aggregation approaches can be extended to compute more complex aggregation functions, for instance combining them with an additional sampling technique. However, this additional functionality is not part of the essence of their core algorithm, bearing different characteristics (e.g., accuracy) and concerns. Some examples can be found in [29] where push-sum is extended with a push-random protocol to obtain random samples, and in [99] which introduces algorithms to estimate several spatially-decaying aggregation functions.

4.5 Counting (Deterministic/Randomized)

This category refers to a restricted set of distributed algorithms, designed to compute a specific aggregation function: COUNT[6]. Recall that COUNT allows the determination of important properties in the design of some distributed applications. For instance, in this context it finds a common practical application in the determination of the size of the system (or group), or to count the number of votes in an election process.

According to the computational principles, from which all the approaches of this class are founded, two sub-classes of counting algorithms have been identified: *deterministic* and *randomized*. *Deterministic* algorithms correspond to those that precisely count the number of target elements. The *randomized* sub-class refers to algorithms that rely on the use of some randomized principle and probabilistic method to produce an estimate. This type of algorithm is usually based on the execution of some *sampling* technique to provide a probabilistic approximation to the size of the sample population. Nonetheless, a few algorithms are found that do not collect samples for size estimation, applying instead a probabilistic *estimator* over some observed events or known system properties.

Algorithms based on sampling are strongly influenced by the probabilistic method used to obtain the result, inheriting its properties. The accuracy of the algorithm corresponds to the one provided by the used probabilistic method, being bounded by the error factor associated with it. Several probabilistic methods have been applied to samples to yield a counting estimation, namely: *birthday problem* [100] – paradox that refers to the probability of two elements sampled out of a population not being repeated, inspired from the probability of two people out of a group not having a matching birthday (exemplifying results: from a random group of 23 people the probability of two of them been born on the same day of the year is about 50%, and is greater than 99% for a group of 57 persons); *capture-recapture* [101] – probabilistic method based on the repeated capture of samples from a closed population (population that maintains a fixed size during the sampling process), where the number of repeated elements between samples are accounted to provide an estimate of the population size; *fundamental probabilistic methods* – application of Bernoulli based sampling methods [68], and other basic probabilistic concepts on some sampled statistical information, like the distances between nodes (number of hops) or the number of messages successfully sent/received, in order to estimate de size of the network.

In all cases, typically sampling is performed at a single node, and it can take several rounds to collect a single sample. Moreover, an estimation error is always present, even if no faults occur. For example, in *Sample & Collide* [27], [24] the estimation error can reach 20%, and a sampling step takes $\bar{d}T$ (where $\bar{d}$ is the average connection degree and T is a predefined timer that must be sufficiently large to provide a good sample quality), needing to be repeated until l sample collisions are observed.

As previously referred, in some cases a size estimate can be obtained by directly applying an estimator on some available system knowledge (observed events or other known properties), without any previous sampling. For instance, in the approach proposed by Horowitz and Malkhi [22] (see section 3.2) an estimator function is used at each node to estimate the network size, based on the observation of two events (nodes joining or leaving the network), incrementing/decrementing the estimator. Other approaches like the one proposed in [102], [103] provide a size estimate based on knowledge of the routing structure, in this particular case counting the number of high degree nodes. Note that, this kind of techniques does not provide accurate result, in most cases yielding a rough approximation to the correct value.

4.5.1 Deterministic

Dolev et al. [25], [26] proposed a deterministic counting approach to estimate the size of a group. This scheme is based on the circulation of a token across the network, where the information from visited nodes is stored at each passage (updating their counters). The token performs a random walk (see section 3.4) in order to count the number of nodes. Notice that, a random walk can be seen as a probabilistic sampling process, however in this case the idea is to continuously reach all available nodes (not sample a portion of them). Moreover, here, the size is estimated by a simple deterministic count of the number of valid elements stored in the token (not by the application of a probabilistic estimator).

In the case of large network, the data that might be stored in the token can correspond to a proportionally large amount of information, which needs to be transmitted from node to node, and consequently may originate a performance bottleneck. A single visit to all network nodes is required to produce an exact estimation, however it is hard to determine if all nodes

6. Although other classes of algorithms (e.g., averaging) can also be used for COUNT, this category refers to approaches confined to the computation of this aggregation function.

have been visited (since the network size is unknown). Moreover, the loss of the token due to a single failure will compromise the whole process, which depends on its existence.

4.5.2 Sample & Collide

This approach [27], [24] addresses the problem of counting the number of peers in a P2P overlay network, inspired by a birthday problem technique (first proposed by Bawa et al. on a technical report [23]). The application of this probabilistic method requires the collection of uniform random samples. To this end, the authors proposed a peer sampling algorithm based on the execution of a continuous time random walk, in order to obtain unbiased samples (asymptotically uniform). The sampling routine proceeds in the following way: an initiator node i sets a timer with a predefined value T, which is sent in a sampling message to a randomly selected neighbor; upon receiving a sampling message, the target node (or the initiator after setting the timer) picks a random number U uniformly distributed within the interval $[0, 1]$, and decrements the timer by $\log(1/U)/d_i$ (i.e., $T \leftarrow T - \log(1/U)/d_i$, where d_i is the degree of node i); if the resulting value is less or equal than zero ($T \leq 0$) then the node is sampled, its identification is returned to the initiator and the process stops; otherwise the sampling message is sent to one of its neighbors, chosen uniformly at random. The quality of the obtained samples (approximation to a uniform random sampling) depends on the value T initially set to the timer.

The described sampling step (to sample one peer) must be repeated until one of the nodes is repeatedly sampled a predefined number of times l (i.e., l sample collisions are observed). After concluding this sampling process, the network size n is estimated using a Maximum Likelihood (ML) method. The ML estimate can be computed by solving Equation 3, where C_l corresponds to the total number of samples until one is repeated l times, using a standard bisection search. Alternatively, the result can be approximated within $\sqrt{n}$ of the ML-estimator by Equation 4 (asymptotically unbiased estimator), which is computationally more efficient.

$$\sum_{i=0}^{C_l-l-1} \frac{i}{n-1} - l = 0 \tag{3}$$

$$\widehat{n} = C_l^2/2l \tag{4}$$

The accuracy of the produced result is determined by the parameter l, and its fidelity depends on the ability of the sampling method at providing uniformly distributed random samples (T must be sufficiently large).

4.5.3 Capture-Recapture

Mane et al. [28] proposed an approach based on the capture-recapture statistical method to estimate the size of closed P2P networks (i.e., networks of fixed size, with no peers joining or leaving during the process). This method requires two or more independent random samples from the analyzed population, and further counting of the number of repeated individuals that appear in each sample. The authors use random walks to obtain independent random samples. Considering a two-sample strategy, two random walks are performed from a source node, one in each sampling phase (capture and recapture). In more detail, each random walk proceeds in the following way: the source node sends a message to a randomly selected neighbor, which at his turn forwards the message to another randomly chosen neighbor; the process is repeated until a predefined maximum number of hops is reached (parameter: *time-to-live*) or the message gets back to a node that has already participated in the current random walk. During this process, the information about the traversed path (i.e., the UIDs of all participating nodes) is kept in the forwarded message. When one of the random walks stopping criteria is met, the message is sent back to the source node with the list of the "captured" nodes, following the reverse traversed path (stored in the message). The information received at the source node from the sampling steps is used to compute the estimate $\widehat{n}$ of the network size, applying Equation 5 (where n_1 is the number of nodes caught in the first sample, n_2 is the number of nodes caught in the second sample, and n_{12} represent the number of recaptured nodes, i.e. caught in both samples).

$$\widehat{n} = \frac{((n_1+1) \times (n_2+1))}{(n_{12}+1)} \tag{5}$$

4.5.4 Hop-Sampling

One of the approaches proposed by Kostoulas et al. [37], [38] to estimate the size of dynamic groups is based on sampling the receipt times (hop counts) of some nodes from an initiator. Receipt times are obtained across the group from a gossip propagation started by a single node, the initiator, that will further sample the resulting hop counts of some nodes to produce an estimate of the group size. In more detail, the protocol proceeds as following: the initiator starts the process by sending an *initiating* message (to itself); upon receiving the initiating message nodes start participating in the protocol, periodically forwarding it to a number (*gossipTo*) of other targets, until a predefined number of rounds (*gossipFor*) is exceeded, or a maximum quantity of messages (*gossipUntil*) have been received; gossip targets are chosen uniformly at random from the available membership, excluding nodes in a locally maintained list (*fromList*) from which a message has already been received; exchanged messages carry the distance to the initiator node, which is measured in number of hops; each node keeps the received minimum number of hops (*MyHopCount*), and sends the current value incremented by one.

After concluding the described gossip process, waiting for a predefined number of rounds (*gossipResult*), the initiator samples the number of hops (*MyHopCount*) from some nodes that are selected uniformly at random.

The average of the sampled hop counts is then used to estimate the logarithm of the size of the group ($log(n)$). In alternatively to the previous sampling process, where nodes wait for the initiator sample request, nodes can decide themselves to send their hop count value back to the initiator node, according to a predefined probability to allow only a reduced fraction of nodes to respond.

4.5.5 Interval Density

A second approach to estimate the size of a dynamic group has been proposed in [37], [38]. This algorithm measures the density of the process identifier space, determining the number of unique identifiers within a subinterval of this space. The initiator node passively collects information about existing identifiers, snooping the information of complementary protocols running on the network. The node identifiers are randomized by applying a hash function to each one, and mapped to a point in the real interval $[0, 1]$. The initiator estimates the group size by determining the number of sampled identifiers X lying in a subinterval I of $[0, 1]$, returning X/I. Notice that, this kind of approach assumes a uniformly random distribution of the identifiers, or use strategies to reduce the existing correlation between them, in order to avoid biased estimations.

4.5.6 Other Approaches

Some counting approaches that are based on a centralized probabilistic polling to collect samples were previously described in this work (in section 3.3), namely: *randomized reports* that illustrate the basic idea of probabilistic polling, and another approach [58] that samples the number of message successfully sent in a single-hop wireless network (further improved in [59]).

Other probabilistic polling algorithms are also available in the specific context of multicast groups, to estimate their membership size. For example, in [104] some older mechanisms were analyzed and extended, and in [105] an algorithm using an estimator based on the Kalman filter theory was proposed to estimate the size of dynamic multicast groups.

5 Summary and Practical Guidelines

Here, we summarize the properties of the main classes of algorithms, stating their advantages and disadvantages (see Table 5), and give some guidelines about their use in specific settings.

Hierarchy-based approaches (see 3.1 and 4.1) require a specific routing structure (e.g., a spanning tree) to operate, and thus are limited by the ability of such structure to cope with churn and link failures. However, this kind of approach is very cheap in terms of exchanged messages, requiring only $2N - 1$ messages[7] to compute the correct average at the sink, i.e. two messages for each N nodes (except for the sink), one to broadcast the aggregation request to its child nodes and another to send the result to its parent. In terms of time, the aggregation process takes $2h$ rounds (at most $2D$, with D representing the network diameter), where h is the height of the routing hierarchy, i.e. h rounds to spread the aggregation request and another h rounds to aggregate the results from child nodes to parents. This kind of technique is commonly used in energy-constrained environments (i.e., WSN), taking advantage of the reduced messages exchanges. Therefore, we would only recommend the application of such aggregation schemes to fault free scenarios, which is often not the case. This kind of approach can be significantly affected by the occurrence of a single failure, losing all the subtree data and greatly impacting the result produced at the sink. For this reason, in scenarios where faults might occur and without regarding energy efficiency, sketch techniques should be preferred, at least providing some path redundancy to reach the sink (at the cost of a k factor increase in terms of messages, with k representing the number of alternative routing paths).

Sketch based approaches (see 4.3) can be applied independently from the underlying routing topology, and thus their use is adequate in fault prone scenarios where only a fair approximation of the aggregate is required. This kind of technique is fast, the closest to the theoretical minimum, requiring only D rounds for all nodes to obtain the estimation result, and achieving this at a total cost of $\bar{d}ND$ messages (i.e., each N nodes send at most one message to each d neighbors at each round)[8]. This kind of approach is adequate for faulty scenarios, especially if one privileges obtaining a fast estimate rather than a precise one. Nonetheless, if a precise estimation is required in a faulty environment, another type of aggregation approach should be chosen, namely an averaging technique.

Averaging algorithms (see 3.5 and 4.2) work independently from the routing topology, and have the particularity of producing results that converge over time to the correct value, being able to output results at all nodes with high accuracies even in faulty environments. The execution time of this kind of algorithms depends on the target accuracy, converging exponentially with linear rounds (at an approximately constant convergence factor between each round), with all nodes sending from one to d message at each round. This kind of approach is slower and consequently requires more messages (although smaller) than sketches, but can exhibit better properties in terms of fault-tolerance, especially to cope with churn. Nevertheless, one should be very careful when choosing the appropriate averaging approach to use, as in practice many exhibit dependability issues (not converging to the correct value) and very few are effectively able to operate on dynamic networks.

7. This is for scenarios where radio broadcast is used to transmit data, such as in WSN. Otherwise, approximately $\bar{d}(2N - 1)$ messages are required, where $\bar{d}$ is the average degree.

8. In the case of WSN, with radio broadcast the total message cost is reduced to ND.

TABLE 5
Summary of the characteristics of main data aggregation classes

	Advantage	Disadvantage	Requirements
Hierarchical	- accurate (if no faults occur); - optimal message load;	- result at a single node; - not fault-tolerant;	- specific routing structure (e.g., spanning tree);
Sketches	- very fast; - result at all nodes; - fault-tolerant;	- limited accuracy (by the sketches size);	- local knowledge of neighbor IDs, or global UIDs; - random number generator;
Averaging	- accurate (converge over time); - result at all nodes; - fault-tolerant; - support network changes;	- fair message load;	- local knowledge of neighbor IDs;
Randomized	- reduced message load (partial network participation);	- not accurate - result at a single node; - not fault-tolerant	- global UIDs;
Digests	- computation of complex aggregates; - result at all nodes;	- limited accuracy (by the digests size); - resources needed (e.g., larger messages);	- local knowledge of neighbor IDs;

Randomized (aka sampling) aggregation techniques (see 3.4 and 4.5) do not seem to bring any advantages when compared to the other kind of approaches. This kind of approach provides an irregular approximation at a single node, not being accurate and usually restricted to the computation of a single aggregation function: COUNT. Furthermore, the random walk based approaches are usually slow, taking several rounds to obtain a sample, and are unreliable as the random walk token might get lost in a faulty environment.

In terms of dynamism, few of the described approaches are tailored to operate on dynamic settings, commonly relying on a restart mechanism (i.e., periodically resetting the computation) to handle network and value changes. In fact, in such settings no algorithm is able to provide an exact value of the aggregate at a specific time instant, as changes can happen during the computation (snapshot validity [90]). Hierarchy-based approaches are completely dependent on a topology maintenance and recovery scheme to work on dynamic settings. The efficiency of such protocol will directly influence the performance of the aggregation algorithm, consuming additional resources to monitor the network in order to detect changes. Moreover, the topology adaptation process (parent switching) can cause temporary disconnections that can still significantly affect the aggregation process. Approaches that operate independently from the network topology, like sketches and averaging schemes, remove the burden of such maintenance protocol to adapt to churn. Sketch based approaches are "fast" but cannot be applied continuously without resetting the computation, as in the case of nodes departure (even when announced) it is not trivial (if not impossible) to remove items from such structures [?]. From the averaging class, only the few that perform idempotent operations between nodes are able to efficiently and continuously (without restarts) work on dynamic settings, converging to the new average resulting from the global "mass" change due to the arrival/departure of nodes or change of the measured input values.

One should notice that most of the existing approaches only allow the computation of simple aggregation functions, such as AVERAGE, SUM and COUNT, or others that can be derived from their combination (by executing multiple instances of the used algorithm). In many cases, this kind of aggregation functions is enough, but in many other situations the computation of more complex aggregation functions is more useful. A simple example can be found considering some load balancing application that aims to distribute equitably the global load of a system. In this case, the knowledge of the total or average load does not provide enough information to assess the distribution of the system load, i.e. determine if some processing nodes are overloaded or idle. Even the computation of the maximum and minimum is insufficient, although it allows the detection of a gap across the global load distribution, as it does not provide information about the number of processes at each load level. In this situation, an estimation of the (statistical) load distribution is required to provide the desired information and reveal outlier values. Other examples can be found in the context of monitoring applications. For instance, in WSN estimating the distribution of the monitored attribute can be very useful to distinguish isolated sensor anomalies from the occurrence of a relevant event characterized by a certain amount of abnormal values. Few approaches are available to compute more complex aggregates and able to approximate the statistical distribution of some attribute (see 4.4). Existing algorithms from this class (i.e., digests) are more resource consuming and less accurate than other approaches, so that their application should be carefully evaluated despite their additional value.

6 Final Remarks and Future Directions

This survey was organized around three main contributions. First, it provides a formal definition of the target aggregation problem, defining different types of aggregations functions and their main properties. Second, a taxonomy of the existing algorithms is proposed, from two perspectives: communication and computation, and the most relevant algorithms are succinctly described. Finally, a summary of the characteristics of the main approaches is provided, giving some guidelines about their suitability to different scenarios.

Distributed data aggregation has been an active field of research in the last decade, and a huge diverse amount of techniques can be found in the literature. For this reasons, this survey intends to be an important time saving instrument, for those that desire to get a quick and comprehensive overview of the state of the art on distributed data aggregation. Moreover, by carefully highlighting the strengths and limitations of the more pertinent approaches, this study can provide a useful assistance to help readers choose which technique to apply in specific settings.

Currently, there is no ideal general solution to the distributed computation of an aggregation function, all existing techniques have their drawbacks (some more than others as depicted by Table 5). Therefore, more research in this field will be expected in the next few years. In particular, due to the added value of computing complex aggregates, new algorithms might arise to estimate the statistical distribution of values, as the few existing approaches exhibit some limitations in terms of accuracy and resource consumption. Additional research efforts should be made in this field to improve the support to churn, message loss, and continuous estimation of mutable input values.

References

[1] G. Manku, “Routing networks for distributed hash tables,” in *22nd annual symposium on Principles of Distributed Computing (PODC)*, 2003.

[2] I. Stoica, R. Morris, D. Karger, M. Kaashoek, and H. Balakrishnan, “Chord: A scalable peer-to-peer lookup service for internet applications,” in *conference on Applications, technologies, architectures, and protocols for computer communications (SIGCOMM)*, 2001, pp. 149–160.

[3] A. Ganesh, A. Kermarrec, and L. Massoulie, “Peer-to-peer membership management for gossip-based protocols,” *IEEE Transactions on Computers*, vol. 52, no. 2, pp. 139–149, 2003.

[4] I. Abraham and D. Malkhi, “Probabilistic quorums for dynamic systems,” *Distributed Computing*, vol. 18, no. 2, pp. 113–124, 2005.

[5] Z. Bar-Yossef, R. Friedman, and G. Kliot, “RaWMS - Random Walk Based Lightweight Membership Service for Wireless Ad Hoc Networks,” *ACM Transactions on Computer Systems (TOCS)*, vol. 26, no. 2, 2008.

[6] R. van Renesse, “The Importance of Aggregation,” in *Future Directions in Distributed Computing*, A. Schiper, A. A. Shvartsman, H. Weatherspoon, and B. Y. Zhao, Eds. Springer-Verlag, 2003, pp. 87–92.

[7] E. Fasolo, M. Rossi, J. Widmer, and M. Zorzi, “In-network aggregation techniques for wireless sensor networks: a survey,” *IEEE Wireless Communications*, vol. 14, no. 2, pp. 70–87, 2007.

[8] R. Rajagopalan and P. Varshney, “Data-aggregation techniques in sensor networks: a survey,” *IEEE Communications Surveys & Tutorials*, vol. 8, no. 4, pp. 48–63, 2005.

[9] Y. Sang, H. Shen, Y. Inoguchi, Y. Tan, and N. Xiong, “Secure Data Aggregation in Wireless Sensor Networks: A Survey,” in *7th International Conference on Parallel and Distributed Computing, Applications and Technologies (PDCAT)*, 2006, pp. 315–320.

[10] H. Alzaid, E. Foo, and J. Nieto, “Secure data aggregation in wireless sensor network: a survey,” in *6th Australasian conference on Information security (AISC)*, 2008, pp. 93–105.

[11] E. Nakamura, A. Loureiro, and A. Frery, “Information fusion for wireless sensor networks: Methods, models, and classifications,” *ACM Computing Surveys (CSUR)*, vol. 39, no. 3, pp. 1–55, 2007.

[12] A. Syropoulos, “Mathematics of multisets,” in *Workshop on Multiset Processing: Multiset Processing, Mathematical, Computer Science, and Molecular Computing Points of View*, 2001, pp. 347–358.

[13] B. Pittel, “On Spreading a Rumor,” *SIAM Journal on Applied Mathematics*, vol. 47, no. 1, pp. 213–223, 1987.

[14] S. Madden, M. Franklin, J. Hellerstein, and W. Hong, “TAG: a Tiny AGgregation service for ad-hoc sensor networks,” *ACM SIGOPS Operating Systems Review*, vol. 36, no. SI, pp. 131–146, 2002.

[15] S. Motegi, K. Yoshihara, and H. Horiuchi, “DAG based In-Network Aggregation for Sensor Network Monitoring,” in *International Symposium on Applications and the Internet (SAINT)*, 2005, p. 8.

[16] Y. Birk, I. Keidar, L. Liss, A. Schuster, and R. Wolff, “Veracity Radius: Capturing the Locality of Distributed Computations,” in *25th annual ACM symposium on Principles of Distributed Computing (PODC)*, 2006.

[17] J. Considine, F. Li, G. Kollios, and J. Byers, “Approximate aggregation techniques for sensor databases,” in *20th International Conference on Data Engineering (ICDE)*, 2004, pp. 449–460.

[18] Y.-C. Fan and A. Chen, “Efficient and robust sensor data aggregation using linear counting sketches,” in *IEEE International Symposium on Parallel and Distributed Processing (IPDPS)*, 2008, pp. 1–12.

[19] Y.-C. Fan and A. L. Chen, “Efficient and Robust Schemes for Sensor Data Aggregation Based on Linear Counting,” *IEEE Transactions on Parallel and Distributed Systems*, vol. 21, no. 11, pp. 1675–1691, 2010.

[20] A. Manjhi, S. Nath, and P. Gibbons, “Tributaries and Deltas: Efficient and Robust Aggregation in Sensor Network Streams,” in *ACM SIGMOD International Conference on Management Of Data*, 2005, pp. 287–298.

[21] N. Shrivastava, C. Buragohain, D. Agrawal, and S. Suri, “Medians and beyond: new aggregation techniques for sensor networks,” in *2nd international conference on Embedded networked sensor systems (SenSys)*, 2004, pp. 239–249.

[22] K. Horowitz and D. Malkhi, “Estimating network size from local information,” *Information Processing Letters*, vol. 88, no. 5, pp. 237–243, 2003.

[23] M. Bawa, H. Garcia-Molina, A. Gionis, and R. Motwani, “Estimating aggregates on a peer-to-peer network,” Stanford University, Computer Science Department, Tech. Rep., 2003.

[24] L. Massoulié, E. Merrer, A.-M. Kermarrec, and A. Ganesh, “Peer Counting and Sampling in Overlay Networks: Random Walk Methods,” in *25th annual ACM symposium on Principles of Distributed Computing (PODC)*, 2006.

[25] S. Dolev, E. Schiller, and J. Welch, “Random Walk for Self-Stabilizing Group Communication in Ad-Hoc Networks,” in *21st IEEE Symposium on Reliable Distributed Systems*, 2002, pp. 70–79.

[26] ——, “Random Walk for Self-Stabilizing Group Communication in Ad Hoc Networks,” *IEEE Transactions on Mobile Computing*, vol. 5, no. 7, pp. 893–905, 2006.

[27] A. Ganesh, A. Kermarrec, E. Le Merrer, and L. Massoulié, “Peer counting and sampling in overlay networks based on random walks,” *Distributed Computing*, vol. 20, no. 4, pp. 267–278, 2007.

[28] S. Mane, S. Mopuru, K. Mehra, and J. Srivastava, “Network Size Estimation In A Peer-to-Peer Network,” Department of

Computer Science - University of Minnesota, Tech. Rep. TR 05-030, 2005.

[29] D. Kempe, A. Dobra, and J. Gehrke, "Gossip-Based Computation of Aggregate Information," in *44th Annual IEEE Symposium on Foundations of Computer Science*, 2003, pp. 482–491.

[30] M. Jelasity, A. Montresor, and O. Babaoglu, "Gossip-based aggregation in large dynamic networks," *ACM Transactions on Computer Systems (TOCS)*, vol. 23, no. 3, pp. 219–252, 2005.

[31] J.-Y. Chen, G. Pandurangan, and D. Xu, "Robust Computation of Aggregates in Wireless Sensor Networks: Distributed Randomized Algorithms and Analysis," *IEEE Transactions on Parallel and Distributed Systems*, vol. 17, no. 9, pp. 987–1000, 2006.

[32] P. Jesus, C. Baquero, and P. S. Almeida, "Fault-Tolerant Aggregation by Flow Updating," in *9th IFIP International Conference on Distributed Applications and interoperable Systems (DAIS)*, ser. Springer LNCS, vol. 5523, 2009, pp. 73–86.

[33] ——, "Fault-Tolerant Aggregation for Dynamic Networks," in *29th IEEE Symposium on Reliable Distributed Systems*, 2010, pp. 37–43.

[34] C. Baquero, P. Almeida, and R. Menezes, "Fast Estimation of Aggregates in Unstructured Networks," in *5th International Conference on Autonomic and Autonomous Systems (ICAS)*, 2009, pp. 88–93.

[35] M. Haridasan and R. van Renesse, "Gossip-based distribution estimation in peer-to-peer networks," in *International workshop on Peer-To-Peer Systems (IPTPS)*, 2008.

[36] J. Sacha, J. Napper, C. Stratan, and G. Pierre, "Adam2: Reliable Distribution Estimation in Decentralised Environments," in *30th International Conference on Distributed Computing Systems (ICDCS)*. IEEE, 2010, pp. 697–707.

[37] D. Kostoulas, D. Psaltoulis, I. Gupta, K. Birman, and A. Demers, "Decentralized Schemes for Size Estimation in Large and Dynamic Groups," in *4th IEEE International Symposium on Network Computing and Applications*, 2005, pp. 41–48.

[38] D. Kostoulas, D. Psaltoulis, I. Gupta, K. P. Birman, and A. J. Demers, "Active and passive techniques for group size estimation in large-scale and dynamic distributed systems," *Elsevier Journal of Systems and Software*, vol. 80, no. 10, pp. 1639–1658, 2007.

[39] L. Chitnis, A. Dobra, and S. Ranka, "Aggregation Methods for Large-Scale Sensor Networks," *ACM Transactions on Sensor Networks (TOSN)*, vol. 4, no. 2, pp. 1–36, 2008.

[40] S. Madden, R. Szewczyk, M. Franklin, and D. Culler, "Supporting Aggregate Queries Over Ad-Hoc Wireless Sensor Networks," in *4th IEEE Workshop on Mobile Computing Systems and Applications*, 2002, pp. 49–58.

[41] P. Flajolet and G. Martin, "Probabilistic counting algorithms for data base applications," *Journal of Computer and System Sciences*, vol. 31, no. 2, pp. 182–209, 1985.

[42] Y. Birk, I. Keidar, L. Liss, and A. Schuster, "Efficient Dynamic Aggregation," in *20th International Symposium on DIStributed Computing (DISC)*, 2006, pp. 90–104.

[43] S. Nath, P. Gibbons, S. Seshan, and Z. Anderson, "Synopsis diffusion for robust aggregation in sensor networks," in *2nd international conference on Embedded networked sensor systems (SenSys)*, 2004.

[44] J. Li, K. Sollins, and D. Lim, "Implementing Aggregation and Broadcast over Distributed Hash Tables," *ACM SIGCOMM Computer Communication Review*, vol. 35, no. 1, pp. 81–92, 2005.

[45] J. Zhao, R. Govindan, and D. Estrin, "Computing Aggregates for Monitoring Wireless Sensor Networks," in *1st IEEE International Workshop on Sensor Network Protocols and Applications*, 2003, pp. 139–148.

[46] M. Dam and R. Stadler, "A Generic Protocol for Network State Aggregation," *Radiovetenskap och Kommunikation (RVK)*, 2005.

[47] L. Jia, G. Noubir, R. Rajaraman, and R. Sundaram, "GIST: Group-Independent Spanning Tree for Data Aggregation in Dense Sensor Networks," in *Distributed Computing in Sensor Systems*, ser. Lecture Notes in Computer Science, P. Gibbons, T. Abdelzaher, J. Aspnes, and R. Rao, Eds. Springer Berlin / Heidelberg, 2006, vol. 4026, pp. 282–304.

[48] A. Sharaf, J. Beaver, A. Labrinidis, and K. Chrysanthis, "Balancing energy efficiency and quality of aggregate data in sensor networks," *International Journal on Very Large Data Bases (VLDB)*, vol. 13, no. 4, pp. 384–403, 2004.

[49] R. Misra and C. Mandal, "Ant-aggregation: ant colony algorithm for optimal data aggregation in wireless sensor networks," in *IFIP International Conference on Wireless and Optical Communications Networks*, 2006, p. 5.

[50] W. Liao, Y. Kao, and C. Fan, "Data aggregation in wireless sensor networks using ant colony algorithm," *Journal of Network and Computer Applications*, vol. 31, no. 4, pp. 387–401, 2008.

[51] H. Wang and N. Luo, "An Improved Ant-Based Algorithm for Data Aggregation in Wireless Sensor Networks," in *International Conference on Communications and Mobile Computing (CMC)*, 2010, pp. 239–243.

[52] H. Luo, J. Luo, Y. Liu, and S. Das, "Adaptive Data Fusion for Energy Efficient Routing in Wireless Sensor Networks," *IEEE Transactions on Computers*, vol. 55, no. 10, pp. 1286–1299, 2006.

[53] W. Heinzelman, A. Chandrakasan, and H. Balakrishnan, "Energy-Efficient Communication Protocol for Wireless Microsensor Networks," in *33rd Annual Hawaii International Conference on System Sciences*, 2000, p. 10.

[54] H. Luo, Y. Liu, and S. K. Das, "Distributed Algorithm for En Route Aggregation Decision in Wireless Sensor Networks," *IEEE Transactions on Mobile Computing*, vol. 8, no. 1, pp. 1–13, 2009.

[55] W. Heinzelman, A. Chandrakasan, and H. Balakrishnan, "An application-specific protocol architecture for wireless microsensor networks," *IEEE Transactions on Wireless Communications*, vol. 1, no. 4, pp. 660–670, 2002.

[56] A. Prieto and R. Stadler, "Adaptive Distributed Monitoring with Accuracy Objectives," in *SIGCOMM Workshop on Internet Network Management (INM)*, 2006, pp. 65–70.

[57] A. Deligiannakis, Y. Kotidis, and N. Roussopoulos, "Hierarchical In-Network Data Aggregation with Quality Guarantees," in *International Conference on Extending Database Technology (EDBT)*, 2004, pp. 658–675.

[58] T. Jurdziński, M. Kutylowski, and J. Zatopiański, "Energy-Efficient Size Approximation of Radio Networks with No Collision Detection," in *8th Annual International Conference on Computing and Combinatorics (COCOON)*, 2002, pp. 279–289.

[59] J. Kabarowski, M. Kutyłowski, and W. Rutkowski, "Adversary Immune Size Approximation of Single-Hop Radio Networks," in *3th International Conference on Theory and Applications of Models of Computation (TAMC)*, 2006, pp. 148–158.

[60] A.-M. Kermarrec and M. Steen, "Gossiping in distributed systems," *ACM SIGOPS Operating Systems Review*, vol. 41, no. 5, pp. 2–7, 2007.

[61] P. Jesus, C. Baquero, and P. S. Almeida, "Dependability in Aggregation by Averaging," in *Simpósio de Informática (INForum)*, 2009.

[62] M. Jelasity and A. Montresor, "Epidemic-style proactive aggregation in large overlay networks," in *24th International Conference on Distributed Computing Systems*, 2004, pp. 102–109.

[63] F. Wuhib, M. Dam, R. Stadler, and A. Clemm, "Robust Monitoring of Network-wide Aggregates through Gossiping," in *10th IFIP/IEEE International Symposium on Integrated Network Management*, 2007, pp. 226–235.

[64] M. Jelasity, W. Kowalczyk, and M. van Steen, "An Approach to Massively Distributed Aggregate Computing on Peer-to-Peer Networks," in *12th Euromicro Conference on Parallel, Distributed and Network-Based Processing*, 2004, pp. 200–207.

[65] L. Chitnis, A. Dobra, and S. Ranka, "Fault tolerant aggregation in heterogeneous sensor networks," *Journal of Parallel and Distributed Computing*, vol. 69, no. 2, pp. 210–219, 2009.

[66] R. van Renesse, K. Birman, and W. Vogels, "Astrolabe: A Robust and Scalable Technology for Distributed System Monitoring, Management, and Data Mining," *ACM Transactions on Computer Systems (TOCS)*, vol. 21, no. 2, pp. 164–206, 2003.

[67] I. Gupta, R. V. Renesse, and K. P. Birman, "Scalable fault-tolerant aggregation in large process groups," in *International Conference on Dependable Systems and Networks (DSN)*, 2001, pp. 433–442.

[68] S. Cheng, J. Li, Q. Ren, and L. Yu, "Bernoulli Sampling Based (ε, δ)-Approximate Aggregation in Large-Scale Sensor Networks," in *29th conference on Information communications (INFOCOM)*. IEEE Press, 2010, pp. 1181–1189.

[69] R. Olfati-Saber and R. M. Murray, "Consensus Problems in Networks of Agents With Switching Topology and Time-Delays," *IEEE Transactions on Automatic Control*, vol. 49, no. 9, pp. 1520–1533, 2004.

[70] S. Boyd, A. Ghosh, B. Prabhakar, and D. Shah, "Randomized gossip algorithms," *Information Theory, IEEE Transactions on*, vol. 52, no. 6, pp. 2508–2530, 2006.

[71] L. Xiao, S. Boyd, and S.-J. Kim, "Distributed average consensus with least-mean-square deviation," *Journal of Parallel and Distributed Computing*, vol. 27, no. 1, pp. 22–46, 2007.

[72] S. Patterson, B. Bamieh, and A. El Abbadi, "Convergence Rates of Distributed Average Consensus With Stochastic Link Failures," *IEEE Transactions on Automatic Control*, vol. 55, no. 4, pp. 880–892, 2010.

[73] A. Demers, D. Greene, C. Hauser, W. Irish, J. Larson, S. Shenker, H. Sturgis, D. Swinehart, and D. Terry, "Epidemic algorithms for replicated database maintenance," in *6th annual ACM Symposium on Principles of distributed computing (PODC)*, 1987.

[74] J.-Y. Chen and J. Hu, "Analysis of Distributed Random Grouping for Aggregate Computation on Wireless Sensor Networks with Randomly Changing Graphs," *IEEE Transactions on Parallel and Distributed Systems*, vol. 19, no. 8, pp. 1136–1149, 2008.

[75] P. S. Almeida, C. Baquero, M. Farach-Colton, P. Jesus, and M. A. Mosteiro, "Fault-Tolerant Aggregation: Flow Update Meets Mass Distribution," in *15th International Conference On Principles Of Distributed Systems (OPODIS)*, ser. Springer LNCS, vol. 7109, 2011, pp. 513–527.

[76] S. Kashyap, S. Deb, K. Naidu, R. Rastogi, and A. Srinivasan, "Gossip-Based Aggregate Computation with Low Communication Overhead," in *12th International Telecommunications Network Strategy and Planning Symposium*, 2006, pp. 1–6.

[77] I. Eyal, I. Keidar, and R. Rom, "LiMoSense – Live Monitoring in Dynamic Sensor Networks," in *7th International Symposium on Algorithms for Sensor Systems, Wireless Ad Hoc Networks and Autonomous Mobile Entities (ALGOSENSOR)*, 2011.

[78] A. Dimakis, A. Sarwate, and M. Wainwright, "Geographic Gossip: Efficient Averaging for Sensor Networks," *IEEE Transactions on Signal Processing*, vol. 56, no. 3, pp. 1205–1216, 2008.

[79] ——, "Geographic gossip: efficient aggregation for sensor networks," in *5th International Conference on Information Processing in Sensor Networks (IPSN)*, 2006, pp. 69–76.

[80] M. Mehyar, D. Spanos, J. Pongsajapan, S. Low, and R. Murray, "Asynchronous Distributed Averaging on Communication Networks," *IEEE/ACM Transactions on Networking*, vol. 15, no. 3, pp. 512–520, 2007.

[81] K.-Y. Whang, B. Vander-Zanden, and H. Taylor, "A linear-time probabilistic counting algorithm for database applications," *ACM Transactions on Database Systems (TODS)*, vol. 15, no. 2, pp. 208–229, 1990.

[82] M. Durand and P. Flajolet, "Loglog Counting of Large Cardinalities (Extended Abstract)," in *11th Annual European Symposium on Algorithms (ESA)*, 2003, pp. 605–617.

[83] P. Flajolet, E. Fusy, O. Gandouet, and F. Meunier, "HyperLogLog: the analysis of a near-optimal cardinality estimation algorithm," in *International Conference on Analysis of Algorithms (AofA)*, 2007, pp. 127–146.

[84] E. Cohen, "Size-estimation framework with applications to transitive closure and reachability," *Journal of Computer and System Sciences*, vol. 55, no. 3, pp. 441–453, 1997.

[85] E. Cohen and H. Kaplan, "Summarizing data using bottom-k sketches," in *26th annual ACM symposium on principles of distributed computing (PODC)*, 2007, pp. 225–234.

[86] Y.-C. Fan and A. L. Chen, "Energy Efficient Schemes for Accuracy-Guaranteed Sensor Data Aggregation Using Scalable Counting," *IEEE Transactions on Knowledge and Data Engineering*, vol. 24, no. 8, pp. 1463–1477, 2012.

[87] C. Baquero, P. Almeida, R. Menezes, and P. Jesus, "Extrema propagation: Fast distributed estimation of sums and network sizes," *IEEE Transactions on Parallel and Distributed Systems*, vol. 23, no. 4, pp. 668–675, 2012.

[88] J. C. Cardoso, C. Baquero, and P. S. Almeida, "Probabilistic Estimation of Network Size and Diameter," in *4th Latin-American Symposium on Dependable Computing (LADC)*, 2009, pp. 33–40.

[89] D. Varagnolo, G. Pillonetto, and L. Schenato, "Distributed statistical estimation of the number of nodes in sensor networks," in *Decision and Control (CDC), 2010 49th IEEE Conference on*, ser. IEEE, dec 2010, pp. 149–1503.

[90] M. Bawa, A. Gionis, H. Garcia-Molina, and R. Motwani, "The price of validity in dynamic networks," in *ACM SIGMOD international conference on Management of data*, 2004, pp. 515–526.

[91] N. Ntarmos, P. Triantafillou, and G. Weikum, "Counting at Large: Efficient Cardinality Estimation in Internet-Scale Data Networks," in *22nd International Conference on Data Engineering (ICDE)*, 2006.

[92] ——, "Distributed hash sketches: Scalable, efficient, and accurate cardinality estimation for distributed multisets," *ACM Transactions on Computer Systems (TOCS)*, vol. 27, no. 1, 2009.

[93] D. Mosk-Aoyama and D. Shah, "Fast Distributed Algorithms for Computing Separable Functions," *IEEE Transactions on Information Theory*, vol. 54, no. 7, pp. 2997–3007, 2008.

[94] ——, "Computing Separable Functions via Gossip," in *25th annual ACM symposium on Principles of Distributed Computing (PODC)*, 2006, pp. 113–122.

[95] M. Greenwald and S. Khanna, "Power-conserving computation of order-statistics over sensor networks," in *23rd ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems (PODS)*, 2004, pp. 275–285.

[96] I. Eyal, I. Keidar, and R. Rom, "Distributed Clustering for Robust Aggregation in Large Networks," in *5th Workshop on Hot Topics in System Dependability (HotDep)*, 2009.

[97] ——, "Distributed data classification in sensor networks," in *Proceeding of the 29th ACM SIGACT-SIGOPS symposium on Principles of distributed computing (PODC)*, 2010, pp. 151–160.

[98] M. Borges, P. Jesus, C. Baquero, and P. S. Almeida, "Spectra: Robust Estimation of Distribution Functions in Networks," in *12th IFIP International Conference on Distributed Applications and interoperable Systems (DAIS)*, ser. Springer LNCS, vol. 7272, 2012, pp. 96–103.

[99] E. Cohen and H. Kaplan, "Spatially-decaying aggregation over a network: model and algorithms," in *ACM SIGMOD international conference on management of data*, 2004.

[100] A. DasGupta, "The matching, birthday and the strong birthday problem: a contemporary review," *Journal of Statistical Planning and Inference*, vol. 130, no. 1-2, pp. 377–389, 2005.

[101] C. J. Schwarz and G. A. F. Seber, "Estimating Animal Abundance: Review III," *Statistical Science*, vol. 14, no. 4, pp. 427–456, 1999.

[102] D. Dolev, O. Mokryn, and Y. Shavitt, "On multicast trees: structure and size estimation," in *22nd Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM)*, 2003, pp. 1011–1021.

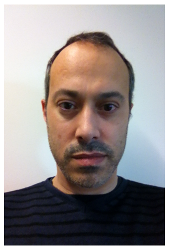
[103] ——, "On multicast trees: structure and size estimation," *IEEE/ACM Transactions on Networking*, vol. 14, no. 3, pp. 557–567, 2006.

[104] T. Friedman and D. Towsley, "Multicast session membership size estimation," in *18th Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM)*, 1999, pp. 965–972.

[105] S. Alouf, E. Altman, and P. Nain, "Optimal on-line estimation of the size of a dynamic multicast group," in *21st Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM)*, 2002, pp. 1109–1118.

Paulo Jesus is currently a Software Developer at Oracle in the MySQL Utilities team. He received the BEng degree in systems and informatics in 2001, and the MSc degree in mobile systems in 2007, both from the University of Minho (Portugal). He obtained his PhD degree in 2012, from the MAP-i doctoral program in computer science by the Universities of Minho, Aveiro and Porto (Portugal). His research interests include distributed algorithms, fault tolerance, and mobile systems.

Carlos Baquero is currently Assistant Professor at the Computer Science Department in Universidade do Minho (Portugal). He obtained his MSc and PhD degrees from Universidade do Minho in 1994 and 2000. His research interests are focused on distributed systems, in particular in causality tracking, peer-to-peer systems and distributed data aggregation. Recent research is focused on highly dynamic distributed systems, both in internet P2P settings and in mobile and sensor networks.



Paulo Sérgio Almeida is currently Assistant Professor at the Computer Science Department in Universidade do Minho (Portugal). He obtained his MSc in Electrical Engineering and Computing from Universidade do Porto in 1994 and his PhD degree in Computer Science from Imperial College London in 1998. His research interests are focused on distributed systems, in particular distributed algorithms (namely aggregation) and logical clocks for causality tracking (with applications to optimistic replication).