LightKone Reference Architecture (LiRA)

White Paper

The LightKone Consortium

Ali Shoker, Paulo Sérgio Almeida, Carlos Baquero, Annette Bieniusa, Roger Pueyo Centelles, Pedro Ákos Costa, Vitor Enes, Carla Ferreira, Pedro Fouto, Felix Freitag, Bradley King, Igor Kopestenski, Giorgos Kostopoulos, João Leitão, Adam Lindberg, Albert van der Linde, Sreeja Nair, Nuno Preguiça, Mennan Selimi, Marc Shapiro, Peer Stritzinger, Ilyas Toumlilt, Peter Van Roy, Dimitrios Vasilas, Georges Younes, Igor Zavalyshyn, and Peter Zeller

This is a draft version that will be updated in the near future.
Current version: V0.91

LiRA is funded by the H2020 Programme of the European Union through the LightKone project.

Abstract

The LightKone Reference Architecture (LiRA) presents a novel edge reference architecture that takes advantage of **decentralized lateral data sharing** and **convergent vertical data semantics** across a myriad of different edge resources. These two principles are key to bridge the existing gaps in current edge-based proposals, namely when considering fog reference architectures, and in particular, by removing the need of centralized lateral data sharing among components that exist in close vicinity and promoting unified data sharing semantics across components in the edge and the core of the system, respectively. LiRA achieves this by moving its focus to application level semantics and exploiting them to enable convergence properties for decentralized data management mechanisms. LiRA is supported by proved and sound techniques like Conflict-free Replicated Data Types (CRDTs) and Transaction Causal Consistency (TCC). These are key to achieve fundamental properties in edge-based solutions such as: autonomy, availability, consistency, and robustness.

LiRA is compatible with, and complementary to, cutting edge/fog standards such as the OpenFog Reference Architecture [5, 12]. To show the practical feasibility of LiRA, this paper presents a reference implementation of LIRA, called *i*-LiRA, composed of a coherent collection of software artifacts and components, that ease the use of the concepts underlying to LiRA in the development of practical applications that take advantage of edge computing. The paper concludes with a discussion on how the artifacts of *i*-LiRA were leveraged to implement four different edge computing case studies across different application domains.

Contents

1. Introduction

1.1 Overview

This paper motivates and presents the LightKone Reference Architecture (LiRA) for edge computing. LiRA was designed to fill gaps in, and complement, existing standard proposals for fog and edge computing architectures, such as the OpenFog Reference Architecture [5, 12] produced by the OpenFog Consortium[1], with emphasis on the data management, both laterally (among devices located within the same fog layer) and vertically (across components operating in distinct fog layers).

LiRA was designed and implemented by the consortium of the LightKone European Project (https://www.lightkone.eu/), which explored the use of state-of-the-art techniques of distributed system design to promote novel architectures for edge and fog computing. We first define LiRA independently of any implementation and subsequently present *i*-LiRA, a concrete implementation of LiRA that provides concrete software artifacts that materialize the key insights of LiRA across the spectrum from cloud to Internet of Things devices. The artifacts of *i*-LiRA were also successfully employed to materialize four different edge-enabled application case studies proposed by the LightKone industrial members. We note that the key insights put forward by LiRA, and materialized by the software artifacts of *i*-LiRA, can easily be integrated and complement existing architectural proposals for edge and fog computing, most notably the OpenFog.

Similar to other proposals, LiRA departs from the fundamental observation that many cloud computing applications operating nowadays, exhibit patterns where (large volumes of) data is generated in the edge of the network (by user and IoT devices) and pushed toward cloud data centers (i.e., the core of the network) for computation and storage/archival. This scenario has become increasingly predominant with the exponential growth of Internet of Things (IoT) applications, whose number of devices and consumed bandwidth are still on the rise today [10]. While the computational power that exists in cloud infrastructures can be scaled to support the increasing pressure generated by IoT and other edge-focused

[1] The OpenFog Consortium has been incorporated into the Industrial Internet Consortium (IIC) as per January 2019: https://www.iiconsortium.org/press-room/01-31-19.htm

applications, the same is not true for the network infrastructures that connects IoT devices to the cloud [25, 26], requiring measures and new technical contributions to, on the one hand alleviate this load, delegating some computations to the edge of the system, and on the other to allow such applications to operate with limited connectivity to the core of the system.

A short-term solution to this problem is to aggregate IoT sensor data close to the (edge) locations where the data is produced – an insight that was made popular by Fog-based architectures and found significant adoption by industry [9, 27]. Unfortunately, this approach does not solve the fundamental problem of having these applications depend on computations (and storage) that is only available at the core of the system in cloud infrastructures. This observation is supported by the exponential growth of IoT that is projected to continue for at least another decade (into the 2030s) with predictions pointing to a grand total of more that one trillion IoT devices being connected and operating by that time [8]. The massive amounts of data that will be generated by these devices will require massive computation and storage capabilities to be continually available, which will not be feasible by only taking advantage of large cloud computing infrastructures.

Instead, a paradigm shift is required, to decentralized storage and computations across the multiple computations resource layers that exist between edge devices and cloud data centers. Shifting components of these applications towards the edge will allow to effectively reduce the volume of data that has to transverse networks to reach data centers, while at the same time enabling data to be processed closer to the location where it is generated, enabling better quality of service, namely faster response times, for applications and end-users.

The second insight that is pursued by LiRA is that, despite the fact that data management is a key challenge on edge and fog applications, it is either not addressed or mostly ignored in existing proposal for edge computing architectures. However, data management is a non-trivial problem, and addressing it adequately at the application level imposes a non-negligible burden on application developers, which potentially leads to incorrect solutions. In LiRA, data management is a key aspect, with LiRA providing simple, yet powerful, and pluggable solutions that deal with data management, and in particular availability and consistency aspects of data shared and stored across different layers of the cloud-edge spectrum at the architectural level.

Finally, while the OpenFog Reference Architecture distinguishes between fog and edge computing, by defining *fog computing* as an extension of the traditional cloud-based computing model where a few components of the architecture can reside beyond the boundaries of cloud data centers and *edge computing* as an alternative model that fully excludes the cloud, LiRA presents an integrated view of both architectural models. LiRA perceives the edge computing model as a general architectural pat-

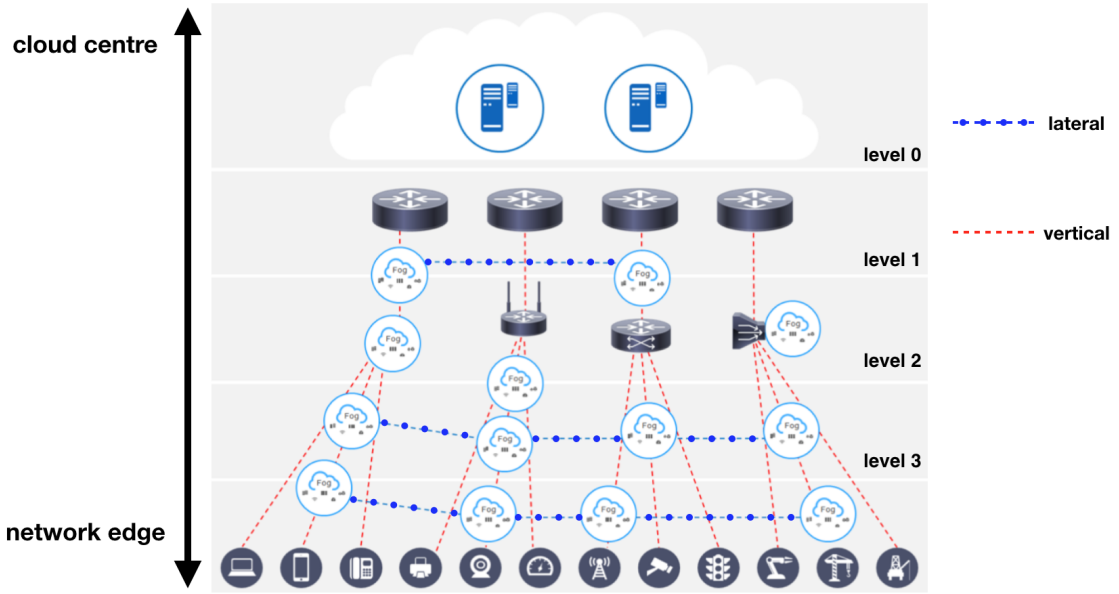

Figure 1.1.1: A typical hierarchical architecture for fog and edge computing in an airport monitoring use case (figure taken from [12]). The diagram shows both lateral and vertical data sharing across different airport sub-systems and terminals at different levels of the fog hierarchy.

tern that can combine and integrate components across a wide range of computational resources that exist between the cloud and the far edge of the system. Due to this, LiRA addresses data management across the full spectrum of the cloud-edge model, being compatible and a potential extension to other proposals [1, 4, 12, 14, 22].

Departing from the architectural view put forward by the OpenFog,[12] and depicted in Figure 1.1.1, existing solutions recognize the existence of storage and computational resources across an hierarchy of different levels from data centers towards the edge. These components, in typical fog architectures, are focused on summarizing collected data from devices located in the lowest hierarchical levels and pushing this data towards the core of the system (i.e., cloud data centers) at the highest hierarchy level. Data is then processed at higher levels and sent back to the network's edge e.g., to update actuators in a sensor-based application.

LiRA models this data flow pattern naming it a "vertical" data flow. However, we also observe that there exist the potential to support a complementary data flow among resources located in the same hierarchical level to allow such resources to cooperate among themselves to perform computations over that data, in a way that is independent of other hierarchical level (increasing overall availability) and potentially decreasing response times. LiRA names such data flows as "lateral" data flows. Enforcing unified semantics and guarantees across these different data sharing flows, enables the design and deployment of edge applications, with near-real time potential, that are more general and that offer better

quality of service.

1.2 Data management problem

Fog computing[2] ideally combines the benefits of both the edge network and the cloud data center, which can improve many aspects of applications including autonomy, agility, efficiency, robustness, performance, and so forth [12]. However, all these benefits come at the cost of new data, communication, and computation challenges due to reliability, security, and mobility issues of the heterogeneous devices and networks in the system. Several academic and industrial platforms and reference architectures have been defined to address these challenges, including OpenFog, EdgeX, AzureIoT, Greengrass, ECC, and MEC [4, 11, 12, 14, 15, 22]. The main challenges are summarized by OpenFog as eight *pillars*: security, scalability, openness, autonomy, RAS (reliability, availability and serviceability), agility, hierarchy, and programmability. Data management is a crosscutting concern that touches on most of these pillars. We define the two main problems of data management as follows:

Problem 1 (P1): Data management at the same fog level is centralized. The architecture provides no defined data management directly among lateral edge nodes, i.e., among nodes at the same level in the fog hierarchy. A common pattern to share data in this case is to push it to a "parent" node at a higher level. The parent node plays a centralized data sharing role. The existence of parent nodes reduces the autonomy, robustness, and scalability of fog applications.

Problem 2 (P2): Data management across fog levels is not consistent. The architecture provides no consistent approach to share data across fog levels, which means that applications cannot easily move from one fog level to another without violating correctness. The consequence is that data management solutions have to be reinvented at each fog level. Sometimes, to improve application responsiveness and decrease latency, loose synchronization models are used with relaxed consistency between fog levels. Unless they are defined carefully, such models will introduce a fundamental data inconsistency between fog levels that greatly increases the complexity of building and managing correct applications.

None of the current fog reference architectures, mentioned above, propose explicit solutions to these problems despite their importance for correct and scalable fog applications. Solving P1 would keep the data in close proximity to its source and nearby nodes, thus avoiding extra time delays, communication failures and overheads, and security threats. Solving P2

[2]For brevity, we will use the term edge computing when addressing a single fog level.

would reduce the complexity of building fog applications and improve their autonomy and mobility across different fog levels without compromising correctness. With respect to the eight pillars put forward by the OpenFog, solving P1 and P2 would simplify application development by addressing several requirements and properties, specifically scalability, autonomy, availability, hierarchy, and programmability.

1.3 LiRA solutions for data management

The LightKone Reference Architecture (LiRA) provides simple and powerful solutions to P1 and P2 based on two principles:

- *Convergent data management* maintains data consistency automatically, i.e., without programmer intervention, over all nodes across and within fog layers.
- *Transactional causal consistency* (TCC) extends convergent data management to application-specific functionality by providing convergence for applications that use transactions.

This gives two levels at which LiRA can be used by developers to build fog applications. The first level, convergent data management, is mostly transparent to the developer. It is easy to support by using the appropriate software components that implement convergent data types. The second level is made available to the developer through a database that provides standard transactional API and supports TCC. In *i*-LiRA, AntidoteDB provides these features together with extensions for fog computing.

Both principles are based on scientific innovations developed within LightKone and related projects. We build on existing convergent data types called CRDTs (Conflict-free Replicated Data Types) [23]. These currently enjoy widespread industrial use inside large databases. We extend CRDTs to directly support edge and fog computing at all layers. In addition, we extend CRDTs to support TCC that guarantees the convergent semantics across fog layers by supporting transactions over CRDTs and causal snapshots [2]. This simplifies applications because they can execute operations correctly at any fog layer independent of the programming model (which can be either data- or event-oriented). The convergence provided by CRDTs and their extensions gives the efficiency of eventually consistent storage combined with a consistency model that is almost as easy to program as a strongly consistent storage. The implementation of *i*-LiRA presented in Chapter 3 showcases software artifacts that illustrate both principles and that cover the whole spectrum from cloud to edge. These artifacts are freely available under open-source licenses, for download, study, and use.

Through convergent data management and transactional causal consistency, LiRA provides solutions to both P1 and P2:

Solution 1: Decentralized lateral data sharing. LiRA allows edge nodes to share data laterally without depending on nodes at higher fog levels. This is achieved by adopting loose synchronization among edge nodes on the same level, thus allowing any edge node to execute updates and reads without prior synchronization with others. Although this improves autonomy, availability, and robustness, unless properly designed it can lead to conflicting versions among edge replicas. By using convergent data structures, all conflicts are eliminated. This can be optionally extended by using transactional causal consistency that allows doing read and write operations on many objects without violating the convergent and causal semantics, and without paying the cost of strongly consistent distributed transactions.

Solution 2: Convergent vertical data semantics. LiRA allows fog nodes across different fog levels or on edge networks to follow relaxed consistency to improve responsiveness. This can be optionally extended by using transactional causal consistency to facilitate the compatibility of applications across different fog layers. Applications can observe the state (as a snapshot) and execute operations correctly at any fog level independently of the programming model.

1.4 LiRA white paper contributions

This white paper makes the following main contributions:

- It introduces the fundamentals and principles of LiRA, the LightKone Reference Architecture, over the entire fog spectrum from the cloud center to the network edge.
- It presents *i*-LiRA: an implementation of LiRA demonstrating how it can be materialized in practice through providing open-source stand-alone artifacts and components that can be integrated into other systems.
- It shows the application of LiRA to four real-world industrial use-cases showing its feasibility to diverse classes of fog and edge applications.

2. LiRA Reference Architecture

2.1 Introduction

There is an increasing interest in the edge and fog computing paradigms aiming to reduce the dependency on cloud data centers. While these paradigms bring several benefits on security, efficiency, and cost (among others), they incur new challenges nicely summarized in the eight *Pillars* of the OpenFog Reference Architecture (OpenFog RA) [12]: security, scalability, openness, autonomy, RAS (reliability, availability and serviceability), agility, hierarchy, and programmability. Several academic and industrial platforms and reference architectures (RAs) like OpenFog, EdgeX, AzureIoT, Greengrass, ECC, MEC [4, 11, 12, 14, 15, 22] have been proposed to address these challenges. However, none of these RAs, and in particular OpenFog RA[1], sufficiently tackled the data management issues in the fog and edge applications.

State of the art fog solutions left two main data management problems: (P1) **centralized lateral data sharing** at the same fog level and (P2) **inconsistent vertical data management** of fog applications across the different fog levels. P1 impedes the full autonomy and robustness of edge applications and increases latency. P2 significantly increases the complexity and cost of developing fog applications that maintain correct semantics while moving across fog levels.

The LightKone Reference Architecture (LiRA)[2] bridges the above gap by introducing two solutions. The first solves P1 through introducing **decentralized lateral data sharing (Solution 1)** through convergent data replication among edge nodes at the same fog level. The second solution addresses P2 through introducing **convergent vertical data semantics (Solution 2)** across the fog levels by specifying a unified data semantics and using data convergence techniques as well.

Both solutions build on the use of relaxed data consistency that is key to reduce the response time of applications. This is possible since a relaxed consistency model allows for concurrent writes without prior

[1] OpenFog is the most comprehensive RA that is a joint effort of a large consortium of leading industry and academia (OpenFog Consortium and the IIC). It has been recently adopted as an IEEE standard [5].

[2] LiRA is an outcome of the LightKone EU project (www.lightkone.eu).

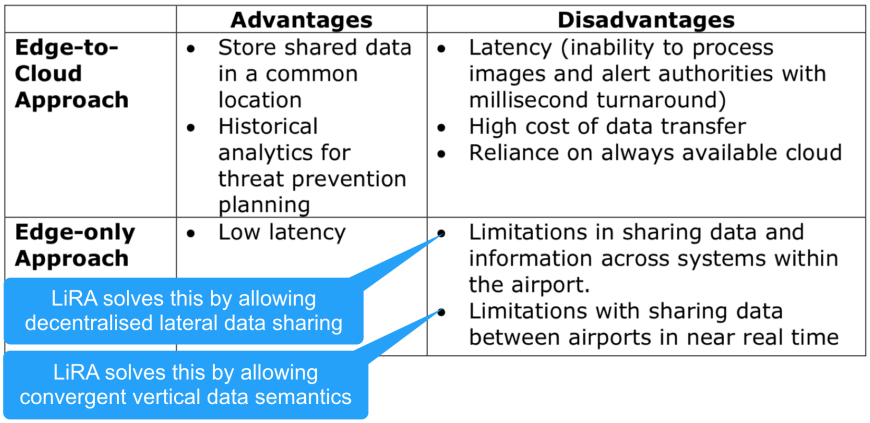

Figure 2.1.1: The limitations and advantages of using the Edge versus Cloud in the Airport Visual Security use case (source OpenFog [12]).

synchronization. However, this often leads to conflicting data versions on different replicas which requires an automatic conflict resolution technique. In LiRA, convergence is guaranteed through using (1) Conflict-free Replicated DataTypes [24] and potentially (2) Transactional Causal Consistency (TCC) [2] for fog applications that require transactions over CRDTs and causal snapshots. LiRA uses these techniques to improve four main properties: Autonomy, Availability, Consistency, and Robustness, while acknowledging the importance of other *Pillars* of fog computing applications [12].

In this section, we introduce LiRA's solutions and principles starting by justifying the need for LiRA. In the following sections, we demonstrate the feasibility of LiRA through presenting a reference implementation, i.e., *i*-LiRA, and four different fog use case discussions.

2.2 Why another reference architecture?

Current fog and edge reference architectures [4, 11, 12, 14, 15, 22] address the fog hardware and software stack through the entire fog continuum—from the cloud center to the far edge. These architectures give little emphasis to studying the fog application data semantics and management. This impedes fog applications from taking full advantage of the *decentralization* in the fog or edge model. In particular, it impedes autonomy, robustness, and increases latency. This gap is explained in the two problems P1 and P2 discussed below, and briefly conveyed in the disadvantages of the "Edge-only Approach", as highlighted by the OpenFog RA [12] in Figure 2.1.1.

For convenience, we make use of the airport surveillance "Airport Visual Security" use case in Figure 2.2.1. This architecture arguably represents a typical fog architecture. Each fog level can be considered an edge layer depending on the user perspective. In this very use case, the lower

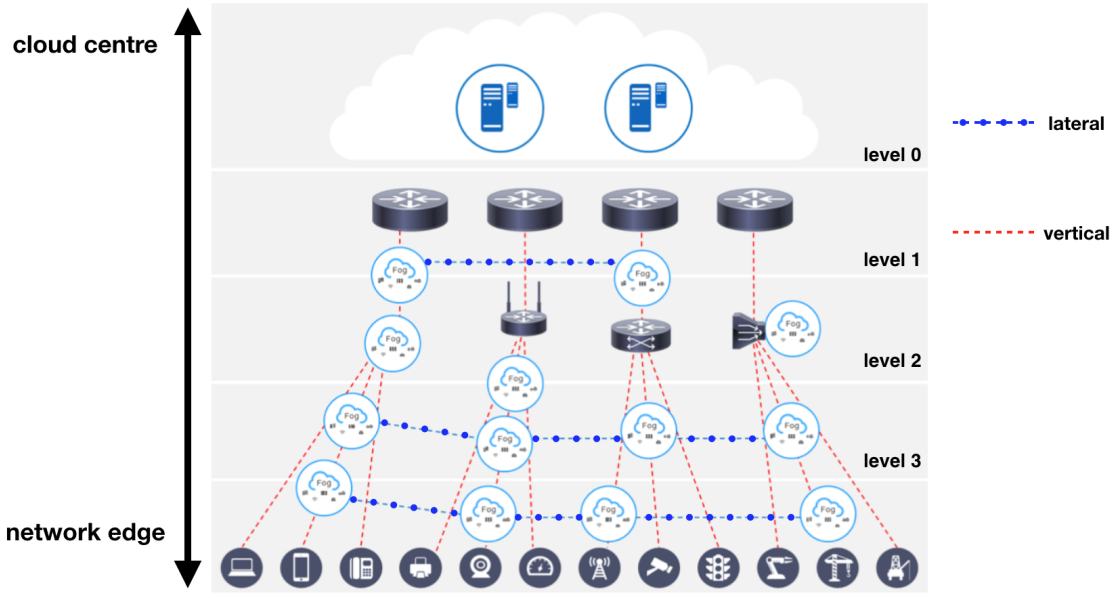

Figure 2.2.1: A typical fog architecture in a airport monitoring use case (source OpenFog [12]). The diagram demonstrates the need for lateral and vertical data sharing across different airport sub-systems and terminals at different fog levels.

layer represents the *Far edge* layer, through which edge nodes in the surveillance sub-systems are scattered over different services or airport terminals.

Problem 1 (P1): Data management at the same fog level is centralized. A common paradigm in fog computing is to have the data retained as close as possible to its source. The aim is to improve availability, data privacy, and reduce the bandwidth overhead. However, the challenge arises since edge applications, at the same fog level, often require a boarder view of the system state. This suggests some sort of lateral data sharing, e.g., replication across different edge nodes within the same fog level. The common practice in current RAs is to push the data to a "parent" node at a lower fog level. In this fashion, the latter node plays a *centralized* data sharing role, i.e., a broker to higher level nodes. Despite being intuitive, this solution is not effective once the lower level nodes are unavailable or the network is slow. Therefore, it does not take full advantage of the "decentralization" properties since data will need to be marshaled to other levels in the fog hierarchy. This induces extra delays and dependencies which results in less autonomy.

Let us exemplify using the airport surveillance "Airport Visual Security" use case (depicted in Figure 2.2.1). Applications can only access the data of the local subsystem; whereas, they can observe the data of other subsystems (or terminals) through the lower level fog nodes (e.g., servers, gateways, etc.). This induces extra delays and potential resilience issues as a result of centralization. A possible alternative is to adopt an "Edge-

only Approach" [12] to get rid of this centralization. Nevertheless, this is considered a limitation as clearly highlighted in the OpenFog RA [12] (see Figure 2.1.1). Therefore, a decentralized lateral data sharing technique is lacking. Such a technique is a non-trivial task due to the known tradeoff of Consistency, Availability, and network Partition-tolerance of the CAP theorem [16].

Problem 2 (P2): Data management across fog levels is not consistent. Solving the lateral data sharing problem is insufficient for scenarios where nodes do not share the same edge network or fog level. The reason is that the traveling cost of data to other fog levels is paid anyways. Consequently, data has to be marshaled across fog levels or edge networks via an intermediary fog node. Again, this centralization incurs "limitations with sharing data between airports in near real time" as pointed out by OpenFog [12] in the airport surveillance use case (in Figure 2.1.1). In this example, while it is efficient to have lateral data sharing within an airport terminal (as in Figure 2.2.1), this advantage is lost once data travels to other terminals or services via "level 3" fog nodes.

To improve responsiveness, one can resort to a relaxed consistency model across fog levels. This is possible because relaxed consistency allows for concurrent operations and stale reads. This may result in conflicting versions and incompatible data semantics that impede applications from moving from one fog level to another, or from one edge network to another. Therefore, there is a need for a consistent data semantics and a technique to enforce them across fog levels. Current fog RAs lack [4, 11, 12, 14, 15, 22] such techniques. This naturally results in complex fog applications and increasing burden on developers.

To solve this problem, current fog RAs delegate the data management of fog applications to a lower applicational layer, e.g., a database or cache system. In this case, there are three options. The first is using a conservative (strong consistency) model that ensures safety guarantees on the expense of data availability at the application layer. Another optimistic solution is to adopt an eventual consistency model in which application semantics may be violated. In this case, the developer has to deal with the nightmare of resolving conflicts. A third tradeoff solution in current RAs is to use timestamped databases (e.g., Cassandra) that ensure *last-writer-wins* as suggested in OpenFog RA [12]. This solves the problem only partially being restrictive to the diverse fog application semantics.

Therefore, there is a need to specify a unified semantics for fog applications and a generic technique to ensure convergence despite changing underlying networks. This allows building applications that are responsive, correct, and compatible with different fog levels and edge networks.

2.3 LiRA Principles and Solutions

LiRA is a fog reference architecture that solves the two problems P1 and P2 through two solutions: *(Solution 1) Decentralized lateral data sharing* between edge nodes at the same fog level and *(Solution 2) Convergent vertical data semantics* across fog levels or edge networks. Both solutions are based on two LiRA principles and a gamma of cutting edge techniques that we discuss in the following. Furthermore, these principles result in a new classification for **LiRA edge layers** that we also present afterwards.

2.3.1 LiRA Principles

LiRA follows the following two main principles to solve problems P1 and P2:

- *Convergent data management* maintains data consistency automatically, i.e., without programmer intervention, over all nodes across and within fog layers.
- *Transactional causal consistency* (TCC) extends convergent data management to application-specific functionality by providing convergence for applications that use transactions.

The problems P1 and P2 in current fog RAs share a common problem: centralization. This is due to relying on a lower level fog node that plays the role of intermediary to higher level edge nodes (at the same fog level). This impedes the full autonomy of nodes and causes high latency and robustness issues due to single point of failures. This strongly suggests decentralization as a viable alternative. However, decentralization can induce a high synchronization overhead that impedes the autonomy of fog nodes and increases the response time of fog applications. This is referred to the known tradeoff of the CAP theorem [16]. The CAP suggests to choose either Availability or (strong) Consistency—given the fact that Fog networks are susceptible to network Partitions.

LiRA trades strong consistency for availability. It adopts a *relaxed data consistency* model: allow an edge node to execute *read* and *update* operations without prior synchronization. Synchronization is however done in the background. Relaxed consistency is vital for fog applications that must operate despite the unreachability of other edge nodes, or fog nodes at other fog levels. However, this means that nodes are operating concurrently, which brings two main challenges on both reads and updates.

Stale reads. The first challenge is that fog applications are required to tolerate *reading stale data*. New data versions can then be observed as soon as remote updates from other nodes *eventually* arrive and get merged. Throughout our research, considering diverse fog and edge use cases (see Chapter 3.2), we concluded that fog applications often tolerate

reading stale data as long as: (1) the system *eventually converges* and (2) causal consistency is preserved.

Conflict resolution. Executing concurrent updates on the same (replicated) object can lead to divergence among its replicas in different edge nodes. This requires a conflict resolution technique that eventually leads to convergence. In LiRA, convergence is maintained in an automatic fashion through the use of *Conflict-free Replicated DataTypes* [23] (CRDTs) that are widely adopted in the geo-replicated data industry[3]. In a nutshell, CRDTs are replicated data structures, e.g., Counters Sets, Maps, Graphs, etc. CRDTs are mathematically proven to converge when commutative, or designed to be so, operations are eventually propagated and applied to all system replicas.

Causality. Even under relaxed consistency, LiRA tries to maintain the strongest guarantees possible in the fog. Therefore, it promotes the *causal consistency* model. This is the strongest consistency model that can be maintained in an Available-Partition-tolerant (AP) system [6]. Causal consistency helps designing fog applications with reasonable *happens-before* [17] semantics: if event A happened before B, then the effect of A must be observed before that of B. Causal consistency is ensured for CRDTs through the support of underlying causal delivery middlewares like *Reliable Causal Broadcast* (RCB) or *anti-entropy* protocols [23, 24].

CRDTs can significantly reduce the burden on application developers, provided the presence of the underlying communication protocol. However, there are cases where multiple objects shall be read and updated at once in a transactional manner. LiRA makes use of *Transactional Causal Consistency (TCC)* [2] that allows applying read and write operations on many objects without violating the causality semantics. This reduces the cost of classical strongly consistent Distributed Transactions without giving up the strongest possible guarantees under relaxed consistency. TCC allows applications to observe the state (a snapshot) that is causally consistent with a previously observed state. This is especially useful to build fog applications compatible with different fog levels or edge networks as explained later.

The use of these principles is key to ensure the availability and correctness of data in edge networks. This also brings more autonomy to edge nodes that can now operate on their own data replicas to serve read and update operations without prior synchronization with other nodes in the edge network. This paves the way to build more robust edge networks as it minimizes the dependency on nodes in other fog levels or cloud data centers. Finally, TCC together with CRDTs significantly reduce the burden on fog application developers. Developers can hence simply use a

[3]Few examples on "Industry use" of CRDTs can be found here: https://en.wikipedia.org/wiki/Conflict-free_replicated_data_type.

datatype (that ensures the required semantics) without delving into the complexity of maintaining convergence under concurrency.

2.3.2 LiRA Solutions

(a) Solution 1: Decentralized lateral data sharing

To solve the centralization problem (P1) in state of the art edge RAs, LiRA enables decentralized lateral data sharing among edge nodes (of the same fog level). Data updates are no longer pushed to a lower level fog node. An object is however fully replicated over the edge nodes. Applications can do updates and reads without prior synchronization, thus boosting autonomy and availability. Nodes propagate updates directly to each other in the background. This keeps the edge network operational despite the unreachability of nodes in other fog levels. In our example, depicted in Figure 2.2.1, the edge (camera) nodes of the surveillance cameras network can share data directly without referring to *level-2* fog nodes (i.e., routers) as intermediaries.

Decentralization is made easy following the above LiRA principles. Using CRDTs, the application developer is only required to choose the CRDT datatype according to the desired semantics of the application. Since CRDTs automatically resolve conflicts, the developer is not required to delve into the concurrency details. Even the underlying communication protocol is transparent to the developer. In the case where multiple CRDT objects are managed together, the developer can resort to using a database that provides TCC, e.g., AntidoteDB [13] in *i*-LiRA. As long as the API of the database is standard, the developer can achieve the expected application semantics even if multiple objects are updated or read at once.

(b) Solution 2: Convergent vertical data semantics

LiRA solves problem (P2) through ensuring data convergence when a relaxed consistency model is used across fog levels or edge networks. This reduces the complexity of building responsive fog applications that can move from one edge network to another without anomalies. Indeed, although Solution 1 solves the centralization problem within an edge network, the data management outside that network must go through lower fog levels. Having a single node or multiple nodes that are strongly synchronized at a lower level is undesired: it will incur response delays and reduce robustness when node and network (partition) failures occur. LiRA requires these very nodes at the lower layer to follow a relaxed consistency model as well. In this fashion, edge networks at the same fog level are always decentralized and higher level nodes may only need to communicate with one lower level node (likely the closest one).

In our example, depicted in Figure 2.3.1, there are four fog nodes at

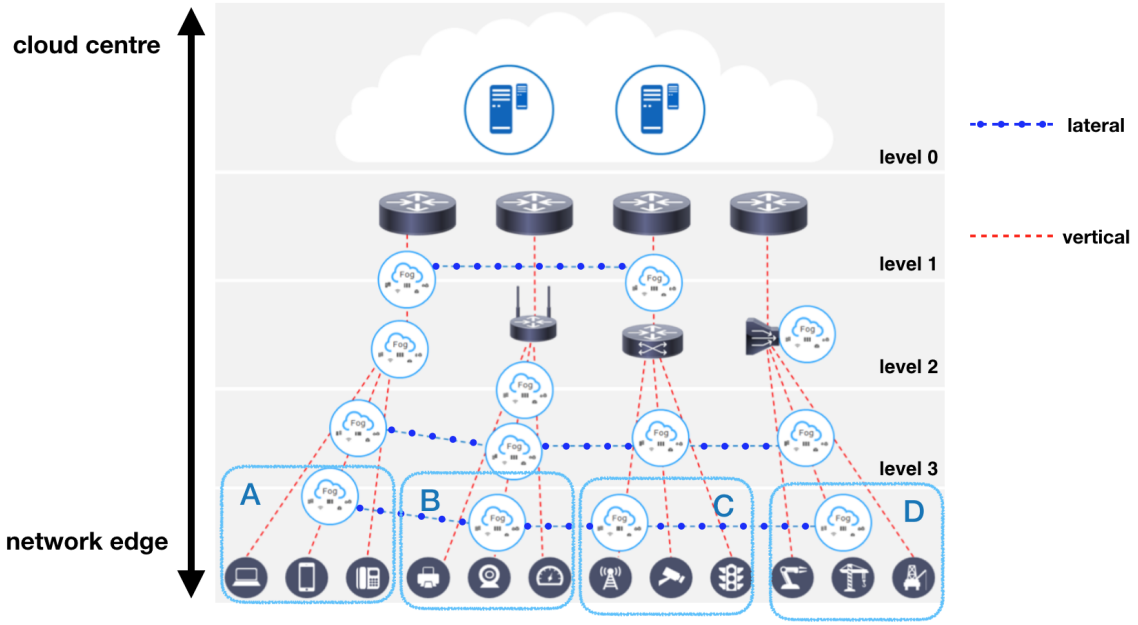

Figure 2.3.1: A typical fog architecture in a airport monitoring use case (source OpenFog [12]). The diagram shows the four Far edge regions where nodes are at a close proximity.

level 3. In LiRA, these nodes belong to a single decentralized edge network following the LiRA principles, explained above. Each one is assumed to be at a closer proximity of three higher level nodes, as shown in regions A, B, C, and D. At the higher level, there are 12 edge networks. Each edge network is decentralized following the LiRA principles. In the inter-edge-network case, one can envision each subsequent three edge networks, e.g., in region A, at this level accessing a single fog node in the lower level (Level 3). This allows edge nodes at the highest level to access data in other edge networks through the intermediary lower node at level 3 without prior synchronization with other level 3 nodes. This gives these nodes higher autonomy and robustness against potential network failures.

While applications accessing nodes within each region (e.g., A, B, C and D in Figure 2.3.1) maintain the correct causal semantics, this is no longer true as an application moves from one region (e.g., A) to another (e.g., B). LiRA suggests common (causal) semantics for applications in all layers being the strongest model possible in AP systems. To facilitate this process, LiRA optionally suggests using a database with a standard API that provides data convergence and TCC. Data convergence is maintained using CRDTs in order to have a correct fog service. On the other hand, TCC allows application semantics to be consistent as they move from one edge network to another. In TCC, applications observe a snapshot of the state that is causally consistent with a previously observed state everywhere. As demonstrated in *i*-LiRA, in Section 3, AntidoteDB [13]

is an example of such a database. This reduces the cost of building applications tailored for each edge network alone. It also reduces the burden on developers who can now delegate the hard consistency work to the database.

2.4 LiRA Edge Layers

Although the above LiRA principles are fog level-agnostic, LiRA defines another classification composed of **Near edge, Mid edge, and Far edge** layers, conveyed in Figure 2.4.1. The identified classification is more relevant from an application semantics perspective—that LiRA emphasizes. (Notice that an edge layer can subsume multiple fog levels.) Although these layers are specified according to their proximity to the cloud center, they share common workflows, communication patterns, computational models, and hardware properties. This aims at making LiRA more feasible to real implementations as it helps using data, communication, and computational abstractions that are convenient to each layer. As we demonstrate in Section 3, this classification simplified the reference implementation *i*-LiRA by building artifacts tailored for the three layers. The important property is that LiRA principles are always maintained within a single layer or across layers.

To explain the LiRA edge layers, we present some of the characteristics that are common for each layer, focusing on: workflows, communication patterns, computational models, and hardware properties. We convey these details in Figure 2.4.2 as well. For convenience, we point to the LiRA use cases as concrete examples demonstrating the soundness of LiRA edge classification. Therefore, the reader may opt to refer to Section 3.2 for more details on use cases' description and specification.

2.4.1 Near edge

This layer is the closest to the cloud center. It considers edge networks having abundant resources in terms of computation, storage, and bandwidth (see Figure 2.4.2). Therefore, Near edge is assumed to handle heavy loads of data either to store or process. LiRA proposes storing data in highly available convergent databases that guarantee LiRA principles. The data is then computed through running well-known batch or stream processing tools, e.g., Hadoop, Spark, etc. This allows the autonomic processing of data being only dependent on the local or close-by database node.

On the other hand, since a Near edge node is close to the lowest level in the fog hierarchy, it plays a role similar to a server or cloud center in the server-client system model. Therefore, although Near edge nodes share data laterally, the common workflow is often to have a Near edge node receive requests from the other edge layers (to store or compute),

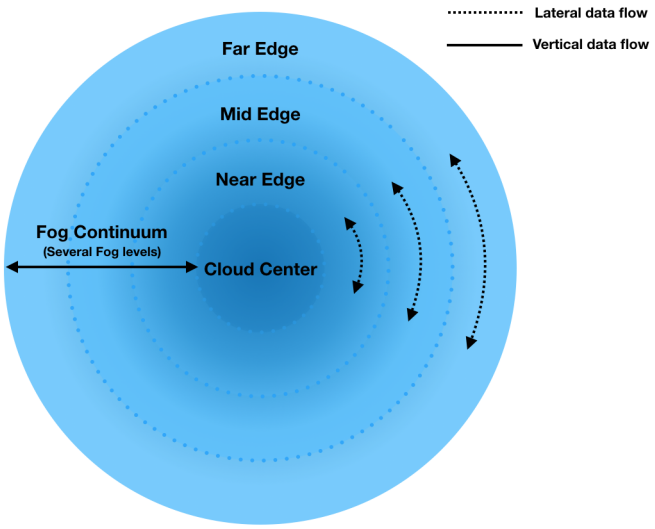

Figure 2.4.1: LiRA edge layers.

and then send the corresponding replies back (if any). As explained in Solution 2, LiRA suggests using databases that provide convergence and TCC to reduce the delays and the complexity of fog applications moving across fog levels or edge networks.

In addition, since the network in such cases is often reliable, some strong network guarantees, like *FIFO* and fixed membership, can be assumed. This makes the presence of a *reliable causal broadcast (RCB)* middleware a valid assumption. Consequently, LiRA often recommends using the *operation-based CRDT* variant [7, 23] at the Near edge. The operation-based CRDT model is generally easy and efficient as it relies on RCB to causally propagate and deliver updates on remote edge nodes.

The multi-cloud metadata search and multi-master geo-distributed storage use cases of Scality, in Section 3.2, are examples of Near edge.

2.4.2 Far edge

Far edge is the most distant layer from the cloud center. It usually comprises the cases where resources are scarce, i.e., where nodes are constrained on energy, computation, and storage, and the network is dynamic with limited bandwidth (see Figure 2.4.2). Therefore, LiRA proposes to push the significant amounts of data and computation to the other fog levels, whereas executing the affordable ones in this layer.

In this manner, LiRA supports the use of lightweight data structures or databases that hold the minimal amount of data needed to support a responsive fog application. Computation is also simple and based on statistical aggregation functions like the *sum, average, mode*, etc. If possible, more sophisticated lightweight computation like Machine Learning tools,

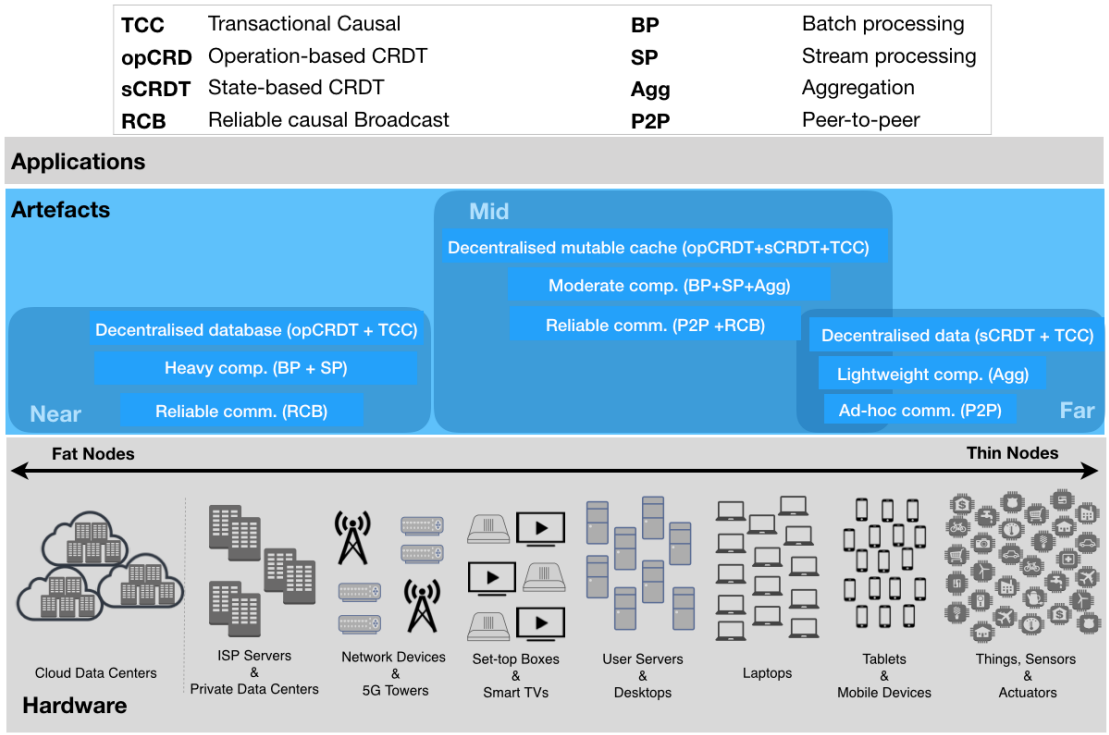

Figure 2.4.2: A figure that shows the characteristics of LiRA edge layers (in the three shaded tables). Within each edge layer, a type of decentralized lateral data sharing is used (Solution 1). Across the three layers, common semantics and possibly TCC are used to support convergent vertical data sharing (as specified in Solution 2). The used techniques shown in each layer are rough options that are likely to be used—although other options are possible.

e.g., TensorFlow, may be used. The rest of the data is however pushed into the lower layers (Near edge or Mid edge) for long-lasting storage and more powerful and accurate computation.

As for the workflow, Far edge nodes are always able to share data laterally following Principle 1 and 2. Since Far edge network incurs dynamic membership and mobility, it is hard to assume (and implement) a Reliable Causal Broadcast (RCB). Therefore, it is often recommended to use a state-based CRDT [3, 24] model that is not prone to duplication since merging updates is idempotent, and thus no need for an RCB. This is convenient since nodes in Far edge often propagate updates in a peer-to-peer fashion. In addition, the use of this model is tolerant to *cycles* that a Far edge workflow can exhibit, and allows the use of data flows that are automatically composable and convergent [21]. Finally, Far edge nodes often play the role of clients to lower layers' nodes in a server-client model.

Considering the communication layer, there is a need for efficient protocols that are robust to network failures, mobility, and churn. LiRA suggests the use of hybrid gossip-based protocols (e.g., Partisan [18–20])

that are able to (1) built efficient dissemination spanning trees to propagate updates and (2) adjust to adversarial conditions (e.g., node failures) without compromising the dissemination process. LiRA requires these protocols to be agnostic to the underlying network layers to support classical protocols like TCP and UDP, as well as wireless ad-hoc protocols for Internet of Things (e.g., 6LoWPAN, Zigbee, and Bluetooth).

The precision agriculture use case of Gluk and NoStop RFID conveyers of Stritzinger, in Section 3.2, are examples of Far edge.

2.4.3 Mid edge

This layer stands as a middle layer to Near edge and Far edge at an intermediate proximity to the cloud center. Its covers the spectrum where edge nodes and network have limited (but not scarce) capacities in terms of storage, computation, and network. The workflow in Mid edge can be like Near edge or Far edge, or likely a hybrid of both. In particular, Mid edge nodes play the role of servers to Far edge nodes and clients to Near edge. As in the other layers, Mid edge follows the LiRA principles to ensure lateral data sharing using CRDT variants and potentially TCC. Importantly, Mid edge nodes must also specify the same semantics in other layers to ensure the correct semantics of fog applications across fog layers or other edge networks (as shown in Figure 2.4.2).

Although a Near edge network can exist as an independent network, LiRA often uses it as an intermediate layer in the entire fog workflow. Therefore, Far edge nodes delegate some of the resource-demanding storage and computation to Mid edge nodes. The latter can then delegate the heavy loads to the Near edge nodes for further computation or long-lasting storage. In particular, Near edge nodes serve as *mutable cache* and near-by computational resources closer to Far edge nodes. This helps to reduce the cost and delays to the distant Near edge nodes or cloud center—that can be unreachable.

Finally, Mid edge networks can deploy the operation-based [7, 24] or state-based [3, 24] CRDT models depending on the case, and following the criteria discussed in Near edge and Far edge. This also holds for the communication protocols that can be anti-entropy hybrid gossip-based or RCB-based. In the latter case, RCB protocols should support bigger networks or dynamic membership.

The distributed monitoring for Guifi.net community network use case, in Section 3.2, is an example of Mid edge.

3. *i*-LiRA: an Implementation of LiRA

To demonstrate the feasibility of LiRA, we provide a reference implementation that provides software artifacts spanning the *Near edge*, *Mid edge*, and *Far edge*. To make clear the distinction between the generic LiRA view and the reference implementation discussed here, we map these three abstract layers to three types of artifacts: *Fat* artifacts that focus in supporting applications at the *Near edge*; *Medium* artifacts that focus on the support of applications in the *Mid edge*; and finally *Thin* artifacts that support applications taking advantage of the *Far edge*.

We structured the presentation according to the following "Views":

- **Architecture View:** this view presents the artifacts developed in the context of LightKone, covering the edge computing spectrum, and how they can be combined for developing edge computing solutions.
- **Use-case View:** this view depicts how the artifacts and components interplay to serve the purposes of each use-case.

We elaborate the above Views in the following explicitly referring to concrete artifacts or software components developed during the project, and accessible for the use of the community. The community is free to use them or implement similar alternatives as desired. The use cases we address are also representative of a diverse sample of edge applications, thus, they shall not be seen as a restricted set of supported use cases, but instead as concrete examples of the broad scope of applications being addressed by LiRA and *i*-LiRA. Last but not least, the proposed implementation focuses on bridging the gap between architecture and technology on LiRA. That said, we do not dismiss alternative technologies that are complementary to those proposed by *i*-LiRA for the construction of practical systems.

3.1 Architecture View

The Architecture View (Figure 3.1.1) presents the artifacts developed in the context of the project and how they address the challenges posed by developing applications for edge computing scenarios. We now give a brief overview of the type of artifacts developed and how they relate (the list of

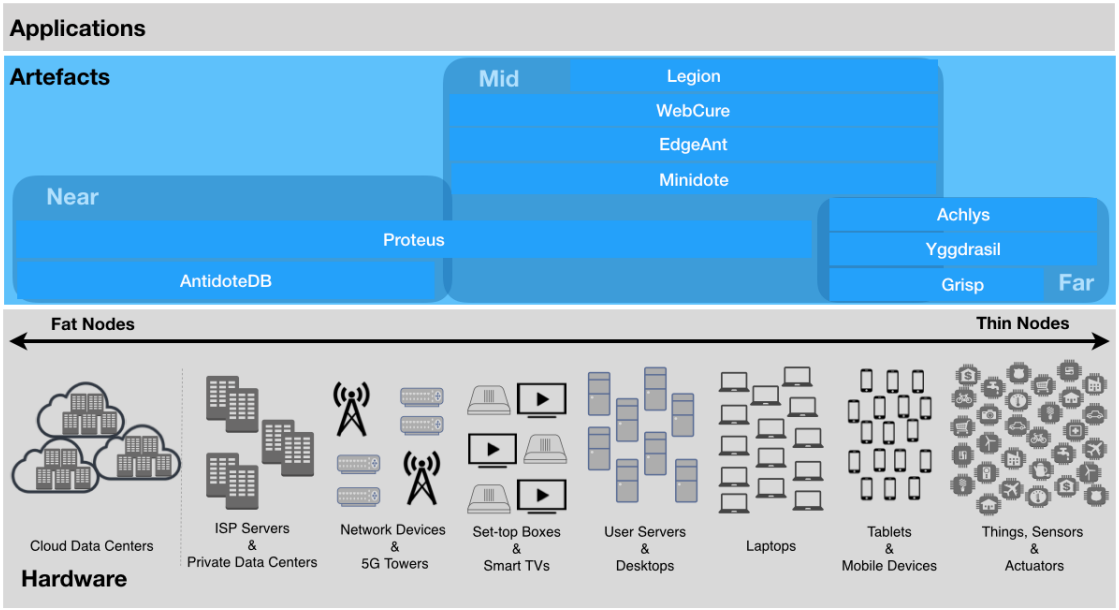

Figure 3.1.1: Architecture View.

artifacts is not exhaustive). A summary of these artifacts is presented in Table 3.1.1 (at the end of this chapter). In addition, Table 3.1.2 presents software components of useful data and communication protocols and libraries that have been used as essential building blocks in *i*-LiRA artefacts. These components are also made available for the community to build new artefacts.

The Fat artifacts were designed to run in nodes with substantial storage and computation capacity (like nodes in public and private cloud infrastructures and ISPs). In this context, we focused in two types of artifacts: replicated data management systems and indexing services. A data management system will be used by applications to store application data. The key requirements of such system is to provide high availability, fault-tolerance and low latency to clients. AntidoteDB is a highly available geo-distributed database that provides address these requirements. The indexing services can be used by applications to provide efficient search of the application information, which can be potentially stored in different databases. Proteus is a system that can be used for such purpose. When necessary, we expect that applications will rely in other types of services, such as messaging services like Apache Kafka, or data processing frameworks like Apache Spark or Apache Storm to fulfill their needs.

The Medium artifacts were designed to run in personal nodes with different storage and processing capacities, including smartphones, laptops and users' servers. In this context, we focused mainly on artifacts that support data sharing and communication among devices. The key challenges in this context are to provide low latency of interaction among

devices, high availability despite node disconnection (including disconnected operations) and address the specific needs of different applications. EdgeAnt provides a cache that allows applications to access data stored in Antidote with low latency, by relying on the data stored in the local cache. It additionally provides support for disconnected operation. WebCure has similar goals, but it is designed to support web applications what run in browsers, thus having to address the challenges posed by running in the browsers' constrained environment. Legion extends WebCure goals by supporting direct interaction among clients, for low latency in interactions among clients. It also supports interaction with different cloud services. For applications that cannot run on top of the data sharing services provided by the LightKone artifacts, we have developed Partisan, a communication middleware that can be used for exchanging data among multiple nodes. Partisan can be used by applications with different requirements, providing an efficient communication substrate that simplifies the development of such applications.

The Thin artifacts were designed to run in the small devices with low memory and storage capacity, including sensor nodes, "things" and mobile devices. In this context we focused in the following types of artifacts. First, communication services for propagating information among nodes of the systems. Yggdrasil provides a generic framework for designing distributed protocols for ad-hoc networking. For example, we have used Yggdrasil for designing an aggregation protocol, that can be used for collecting information from sensors and eventually propagate it to an external service (e.g. Antidote or one of the Light-Edge artifacts). Second, software for embedded devices. Grisp software stack provides efficient communication for application running in ErlangVM in the Grisp nodes. Finally, data sharing services embedded devices. In this context, Lasp when combined with Grisp (from now on denoted as *LaspOnGrisp*) provides a key-value store that allows applications running in embedded devices an high-level abstraction for data sharing.

Table 3.1.2: LightKone software components and libraries used in *i*-LiRA.

Component	**Description**	**Features**	**Artefact use**
Delta/State CRDTs	State-based data management for relaxed consistency at the edge	Generic framework; many datatypes; efficient state sync via join-decomposition	Achlys, Legion
Op CRDTs	Operation-based data management for relaxed consistency at the edge	Generic pure op-based framework; optimized edge-tailored datatypes; support resets; many datatypes	AntidoteDB, Proteus, WebCure, Minidote
Tagged Causal Broadcast (TCB)	middleware for causal consistent systems (e.g., used for op-based CRDTs)	Scalable to tens of nodes. End-to-end application-level causality. Causal stability.	Minidote (CAMUS)
Hybrid gossip distributed communication (Partisan)	Edge-tailored alternative for distribution layer for Erlang.	Hybrid gossip-based with different net topologies and various clusters; mesh-based EVM.	Achlys
Ad-hoc communication (Mirage)	Protocol for aggregation in ad-hoc networks	Efficient handling of variation of input values using hybrid gossip protocols (Plumtree and HyParView)	Achlys, Yggdrasil

3.2 Use-case View

The use-case view shows how the components described above have been employed in several use cases in the context of the LightKone project. These use cases cover a wide spectrum of applications and address diverse aspects related to data sharing and consistency. In the following, we explain how each use case can be placed within LiRA and how the *i*-LiRA artifacts address their specific requirements.

3.2.1 Distributed monitoring for community networks (Guifi.net)

(a) Context

Guifi.net is a free and open community network with more than 35,000 active nodes. The Guifi.net Foundation operates the IP communication network in which a large part of the network consists of interconnected commodity wireless routers, while other parts are build with fiber optics connectivity. For the management of the network, the commodity wireless routers need to be monitored to confirm correct operation, to measure traffic, and to detect connectivity issues and faults.

(b) Data and System model

For the monitoring of these routers, hundreds of servers are provided inside of Guifi.net by individuals and several telecommunication organizations. These monitoring servers connect periodically to the routers and apply SNMP (Simple Network Management Protocol) to obtain operational information of the routers. The operational information is then collected by the monitoring servers and can be visually inspected in the Guifi.net Web.

The Guifi.net use case poses a number of challenges. First, there is the need to monitor each router by a set of monitoring servers to ensure that each router is monitored despite failures in some of the monitoring servers. In addition, the assignment of monitoring servers to routers must be made dynamically, i.e., the assignment can be updated by the servers in a self-organized and decentralized way subject to contextual information (e.g. temporary network situation, server load, relevance of the router). Second, the monitoring system should store and merge the collected data to obtain a consistent view of the system in a robust and fault-tolerant manner. To this end, the data collected by each monitoring server is to be merged and stored to enable further processing and analysis. This robust live monitoring enables the community to conduct a deeper network analysis which allows an efficient and sustainable operation of the network.

To provide reliability and fault-tolerance in the face of frequent network partitions, the Guifi.net use case requires (eventually) consistent replication of the router monitoring data to ensure its service requirements. Classical active-passive replication is unfeasible in this setting since it precludes to employ locality of data and decision, and unavailable under partitions. Further, modifications of associations between routers and servers must merged to obtain a consistent view.

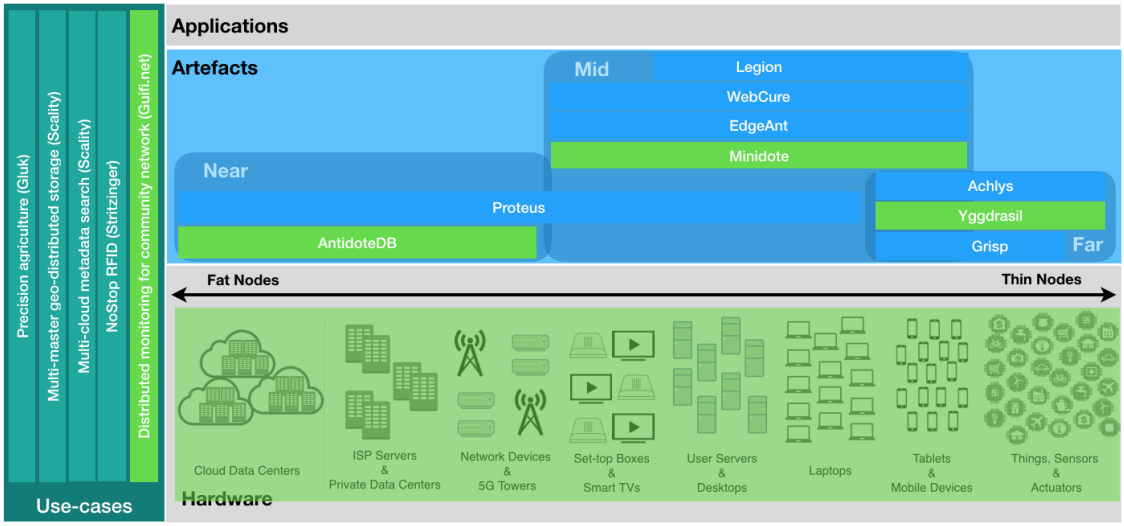

Figure 3.2.1: Distributed monitoring for community network (Guifi.net).

(c) Use-case Implementation

Figure 3.2.1 sketches the software stack that is employed in the Guifi.net use case. As depcited, the use case uses three different *i*-LiRA artefacts depending on the hardware capacity available. The devices are heterogeneous not only from the perspective of their their computing capability, but also from their connectivity, maintenance, ownership and operational policy.

In this setup, AntidoteDB serves as a convergent distributed storage service that is deployed on a selection of servers. For the server-to-router assignment, decentralized read and write operation are performed locally by each server to coordinate between the servers.

The devices involved in the use case include commodity wireless routers and also a variety of different hardware for the servers. These servers range in Guifi.net from Single-Board-Computers such as the Raspberry Pi, to more powerful mini-PCs up to server-class desktop PCs. For this, Guifi.net is also experimenting Minidote as a lightweight version of AntidoteDB that fits such devices. At the underlying communication layer, Guifi.net is also investigating the use of Yggdrasil as a hybrid gossip communication layer. Yggdrasil provides a flexible message dissemination

framework to transmit small and large chunks of (updated) files between far edge nodes.

3.2.2 Multi-master geo-distributed storage (Scality)

(a) Context

Scality's object storage system (RING) supports "active-passive" geo-replication. Data are geo-replicated across multiple sites (data centres), but can be updated on only one of those sites. The other sites are read-only during normal operation, and serve as backups that are ready to take over in case of a failure of the active site. The goal of this use case, shown in Figure 3.2.2, is to support active-active geo-replication in Scality's object storage system, where data on multiple sites will be updated simultaneously and changes will be merged deterministically.

(b) Data and system model

The data model is that of object storage: the system stores a set of objects, organised in buckets which form a flat namespace. Each object consists of a key that uniquely identifies the object within a bucket, a blob of uninterpreted data, and a set of metadata attributes, such as content size, creation timestamp and user-defined tags. Buckets also contain similar metadata attributes. More importantly each bucket maintains a primary index with the keys of the objects which it contains. Data is immutable, modifying the data creates a new version of the object, while metadata attributes can be updated. Scality's storage system performs separation of data and metadata. Data is stored in Scality's RING storage platform while metadata is stored in a separate metadata database.

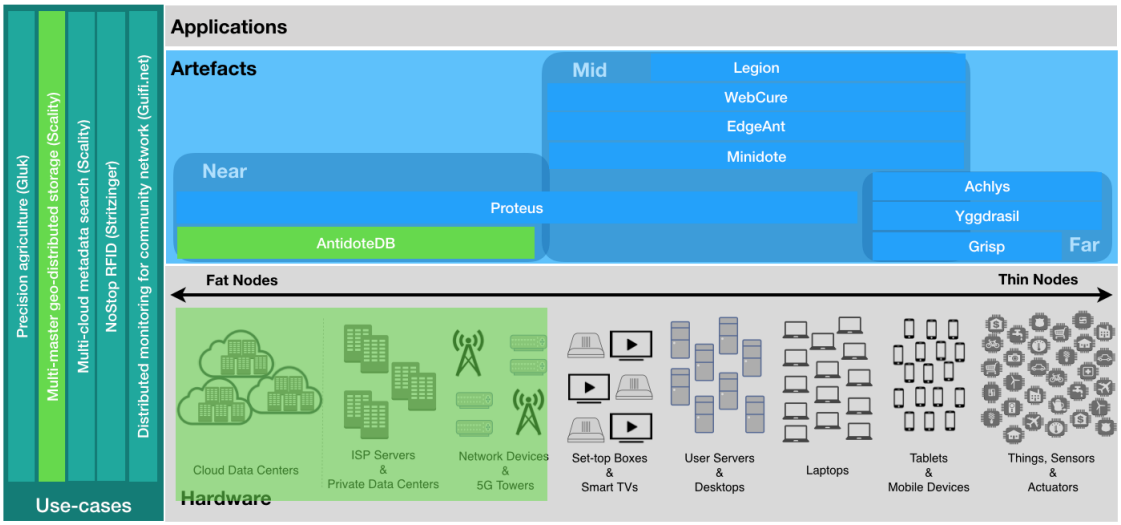

Figure 3.2.2: Multi-master geo-distributed storage (Scality)

(c) Use-case implementation

Scality is using AntidoteDB as a metadata database for Scality's object storage system (as shown in Figure 3.2.2). An Antidote data centre is deployed on each site. Metadata attributes are stored in Antidote as CRDTs, and Antidote handles the geo-replication and convergence of concurrent updates. Data is always stored in the Scality RING and replicated using the existing mechanism.

3.2.3 Multi-cloud metadata search (Scality)

(a) Context

Scality has developed an open-source framework, named Zenko, which is a multi-cloud controller, that enables applications to transparently store and access data on multiple public and private cloud storage systems using a single unified storage interface. Applications can use Zenko to access multiple cloud storage systems, including Microsoft Azure Blob Storage, Amazon S3 and Scality RING with the same API (AWS S3 API), and allows to additionally define policy-based data replication and migration services among these clouds.

In particular, Zenko supports federated metadata search across multiple cloud namespaces. This search service enables applications to retrieve data by performing queries on metadata attributes, such as file size, timestamp of last modification or user-defined tags and others, independent of the data location.

(b) System and Data Model

Zenko's system model consists of a small number of geo-distributed cloud data centres, and a larger number of client devices (user servers & desktops, laptops). Some of the data centres fully replicate data, representing geo-replicated cloud storage systems, while others store disjoint datasets, representing different cloud storage systems. A typical setup is to deploy DC1 & DC2 as AWS S3, and DC3 & DC4 as the Scality RING, a cloud-object storage. To coordinate this multi-DC cloud, an instance of Zenko is deployed on one of these data centres.

Clients perform reads and writes using the S3 API either through Zenko, which then forwards operations to the appropriate clouds (*in-band operations*), or by communicating directly with a backend cloud storage service (*out-of-band operations*). Clients can also perform metadata search queries through Zenko using an SQL-like interface.

To provide this metadata search, Zenko captures and stores object attributes in a dedicated database, i.e., MongoDB. To this end, the metadata attributes are stored as JSON object and Zenko takes advantage of MongoDB's indexing and search capabilities to support the metadata

search. For fault-tolerance, the database is replicated within a data centre using the Raft protocol to prevent divergence of the replicas.

For out-of-band write operations, the metadata attributes are eventually propagated to Zenko's metadata database using event notification mechanisms provided by the cloud services.

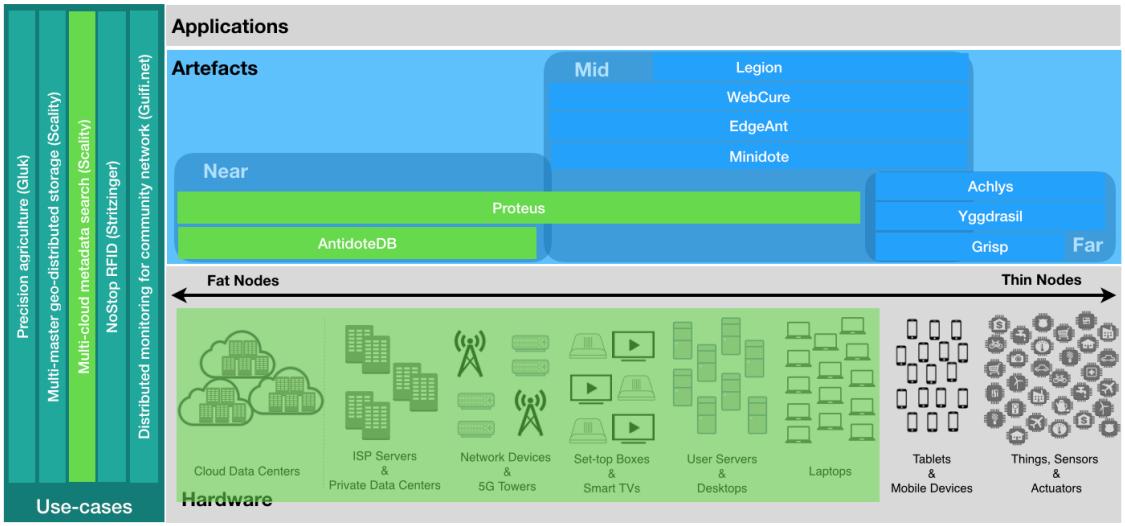

Figure 3.2.3: Multi-cloud metadata search (Scality)

(c) Use-case implementation

Scality built a geo-distributed metadata service as a replacement to the more centralized approach which gathers and stores metadata attributes on a database placed on a single data centre. For this, Scality is using two *i*-LiRA artifacts: (1) Proteus and (2) AntidoteDB, as conveyed in Figure 3.2.3.

Proteus allows to deploy a modular geo-distributed hierarchical network of microservices. Each microservice is responsible for receiving an input stream of data, optionally maintain a user-defined materialized view of the input steam, and provides a query service by performing data-flow computations on the input data and/or its materialized view. Proteus enables flexible placement of data and computations by allowing microservices to be placed in different locations within a geo-distributed system.

Proteus' query-processing microservices receive write operations that have been executed in a cloud storage system as an input stream, maintains a (partial) index on metadata attributes, and provides a metadata search service using this index. These microservices are organized as a hierarchical network that implements a geo-distributed index. The index is partitioned, and index partitions are distributed across data centres. Metadata search queries are executed by performing distributed computations on this network: Each query is incrementally split into sim-

pler sub-queries, and sub-queries are processed by different microservice components using their index partitions.

Microservices in Proteus use AntidoteDB as a backend database for storing indexes as CRDTs. It also provides causality and atomicity guarantees for search results (a search query should either observe all the updates performed within a transaction, or none).

3.2.4 Precision agriculture (Gluk)

(a) Context

The Self Sufficient precision agriculture management for irrigation use case is applied for irrigation management in Subsurface Drop Irrigation method (citrus trees cultivation - but can be applied in every farming activity indoor or outdoor). The water is pumped out from the well (or other water source) and transmitted to the polymer tubes. The water is usually used to irrigate multiple farms. However, as every farm has different characteristics (soil, area, etc.), the irrigation should be adapted taking account these parameters. For example, some parcel of land may need more water while other parcels need less water.

In order to avoid under-watering, the farmers usually irrigate more time than it is necessary. This raises the issues of water waste, energy waste (electricity for the pump), and drainage problems (since the same time many farmers irrigating and the water in the underground water dump is not enough for everyone). In this situation, there is a need to know which part of the farm is either over-watered or under-watered. A non-proper irrigation could affect 25-30% of the annual production.

(b) System model and Requirements

In a self sufficient management system, the farm is divided in clusters with a network of tubes. In every tube, a smart node is installed containing the management unit, sensors and actuators. In this way, the farm is divided into zones. When a zone is sufficiently irrigated (given the retrieved value from the sensor), the actuators stop the water flow into the relevant parts of the tubes. Meanwhile, the under-watered zones of the farm continue to be irrigated. The sensor array has basic computational abilities at the sensor edge nodes, and basic communication abilities between sensor nodes (e.g., Wifi or Zigbee). Decent node reliability is provided by off-the-shelf hardware.

The core management ability must be completely autonomous and low-cost (no need for PC or cloud connectivity), i.e., it should run on the sensor array itself. The system must be able to be installed by the farmer without any configuration capabilities from his side (zero config). This gives modularity for the farmer: he pays only for what he needs, and Internet connectivity is not needed for basic management abilities.

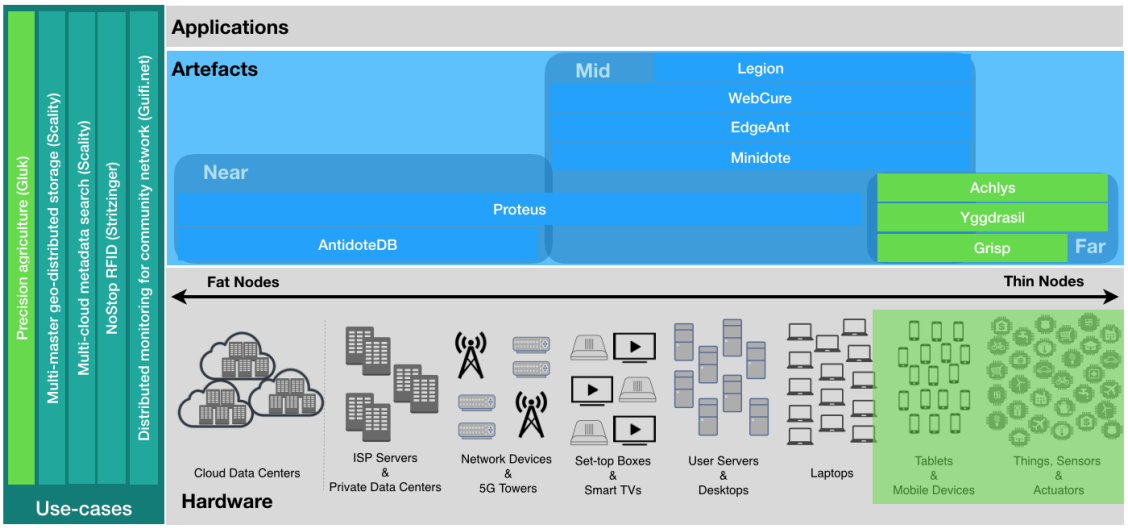

Figure 3.2.4: Precision agriculture (Gluk)

(c) Use-Case Implementation

As shown in Figure 3.2.4, Gluk built a proof of concept solution using three *i*-LiRA artefacts: Achlys, Yggdrasil, and Grisp. Achlys implements a task model on top of a reliable convergent replicated key/value store (called Lasp) that runs with very little computational resources. It runs on top of a communication layer, called Partisan, that ensures reliable hybrid-gossip communication despite highly unreliable connectivity. Achlys also uses basic connectivity and self-management of the system membership via Yggdrasil. The entire node is run on GRiSP software and hardware board that provides native *Erlang* functionality running with low power and basic processor, memory, and wireless connectivity. This entire software and hardware stack used in this use case was (partially or fully) developed within LightKone.

With this solution, the system is able to perform in an autonomous fashion. GRiSP nodes can be powered by solar batteries, whereas occasional problems in edge nodes are solvable by the redundant nature of Lasp data store used by Achlys. Lasp is always convergent thanks to the periodic regeneration of individual components through the underlying robust hybrid-gossip layer (provided through Partisan and Yggdrasil).

Note that additional management policy control is provided by a connection to the sensor array, either by PC, a cloud, or even a mobile device in close vicinity to one of the sensors, which the farmer can rely on at any time.

3.2.5 NoStop RFID (Stritzinger)

(a) Context

Industrial automation systems, such as smart conveyor belts, guide the production steps for a single work piece in manufacturing using RFID tags. These systems communicate with a reusable RFID tag mounted on work piece carriers for manufacturing tracking. Such industrial automation systems can be very large scale. As such, performance and reliability are of utmost concerns. For a large factory with thousands of processing stations and inter-connecting conveyor belts, the least amount of dead time or congestion can cause long reaching inefficiencies.

Traditionally, such a conveyor belt system stores a product-related state on the memory of the RFID tag, which gets transported around together with the product. The memory on the tag is the canonical data storage. As a result, every piece of information, which is needed on stations further down the path, must be written to the tag. This process takes time and is somewhat unreliable (because of radio interference). Therefore all changes to the tag data must also be verified, which means that the tag must stay close to the antenna until all read and write operations have been performed and verified, delaying the overall manufacturing process.

Stritzinger developed a novel efficient solution exploiting LiRA innovations. Although Stritzinger did not make use of *i*-LiRA artefacts, they implemented their own proprietary implementations following the LiRA principles.

(b) Data and system model

In a traditional implementation of a smart conveyer belt, the RFID tag is used as the main data storage device in the system. This means that all data is written to the tag, and physically transported on the conveyor belt to the next antenna (typically located at the next manufacturing station).

This implementation requires the work piece carrier to stop very often, to either read more data, or reliably write new data. For write operations to be reliable, a read, modify, write, read, verify sequence must always be performed. This, in turn, means that the throughput and utilization of the factory is reduced by constant waiting times, which occur before and after each processing station, or in the station itself. In either case, this results in unnecessary pauses, where the processing station is idle.

Each node (i.e., antenna) of the conveyor belt system can maintain a local cache for each tag that appears. These local caches provide an optimization abstraction over the block structure of most modern tags, which allows the system to speed up reading procedures (if the same block is read several times) and to handle short disconnections between the antenna and the tag. This can be seen as a purely local software optimization of the raw low-level antenna API.

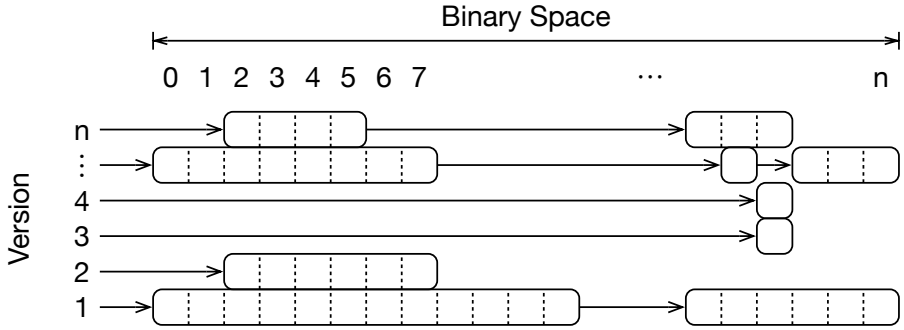

Figure 3.2.5: Illustration showing several stacked sparse binary layers.

To address these limitations, Stritzinger implemented a solution in which the delay produced by constant read and write operations in tags is reduced to become close to zero. To achieve this, instead of using tags as the canonical storage device, tags only contain a single piece of information, a logical clock that acts as a version number. The remainder of data is stored within the network, at each node, on a data structure that follow the convergence properties of CRDTs in LiRA. This can be viewed as a distributed (virtual) tag. This also means that data between nodes must be synchronized. To this end, Stritzinger implemented solution contains a synchronization protocol that follows the concepts found in hybrid gossip, being efficient under normal network conditions and quickly allowing to recover under adverse network conditions (by resorting to additional redundancy).

(c) Use-case implementation

Stritzinger developed an anti-entropy protocol following the hybrid gossip model suggested in LiRA. This protocol enables the efficient and fast dissemination of information related to each (physical) tag. To achieve this, the protocol combines different gossip approaches by using three types of messages: *hello*, *update*, and *request*. Update messages are efficiently disseminated by only disseminating the latest information, and resorting to *request* and *hello* messages to recover missing updates quickly (through a less efficient procedure) only when necessary. This ensures efficiency when in a stable environment, and fast recovery, at the cost of additional communication, when there are changes in the execution environment.

To enable the efficient dissemination of updates, Stritzinger implemented a convergent data structure: a *stacked sparse binaries*, as depicted in Figure 3.2.5, to efficiently store the data of tags. (A sparse binary is a list of ranged values that differ from any designated default value the space might have, e.g., all zeroes.) This structure allows for storing only the changes made to the binary. Furthermore, the sparse binaries can be stacked on top of each other to represent changes over

time to a binary space.

The stacked sparse binary structure follows the convergence principles of LiRA CRDTs. It can be viewed as a delta-based CRDT with Last-Writer-Wins (LWW) semantics. At each node, the replica stores for each tag, the stacked sparse binary values, each one associated with a version number. Whenever the data associated with a tag gets updated in a node, that node gossips to each of the neighboring nodes the layers that might be missing in that node (according to the latest received neighbor clock). What this process does is computing the delta that needs to be propagated to each of the neighbors to make it up-to-date. When an update is received in a node, the received layers are *merged* to the local replica. The value of each element of the binary space is that of the update with the largest timestamp that modified that element.

Table 3.1.1: **Sumamry of the main *i*-LiRA artefacts.**

Artefact	**Description**	**Previous SOTA**	**Contribution**
AntidoteDB	A highly available geo-distributed database	Geo-replicated databases with different consistency semantics, typically either weaker (EC) or stronger (Serializability within shards)	Causal transactions + CRDTs
WebCure	Client-side data replication for web applications using AntidoteDB as backendTODO: Annette	Read-only caches / roll back on updates on conflict	Simplified programming model with conflict resolution on CRDTs
Legion	A framework for extending web applications to the edge, by running code in the client devices that interact directly.	Systems that support disconnected operation, but no peer-to-peer synchronization; Mobile systems that support peer-to-peer interaction, but that are not designed to support web applications.	Simple programming model for extending web application with peer-to-peer synchronization.Big delta CRDTs.Model for interacting with cloud services.Security mechanisms.
EdgeAnt	A consistent, mutable cache at the edge. Data is backed up in Antidote. EdgeAnt supports the same API as Antidote, and guarantees the same TCC+ consistency. A cache can transparently disconnect and reconnect to any data centre. Ongoing work: (i) A client has the option to place any individual computation, either at the edge or in a data centre; both guarantee the same consistent view of data. (ii) Co-located EdgeAnt clients can collaborate in a group, even disconnected from the infrastructure, and can migrate between groups.	Edge caching for immutable data; or non-AP "sticky" or ad-hoc caches with ill-defined guarantees	Consistent, mutable AP cache. Uniform (DC to edge) AP guarantees Client can migrate Place computation @edge or @centre P2P group communication Client can change groups
Yggdrasil	Framework for designing distributed protocols for ad-hoc networking.	Frameworks for developing distributed protocol, but no specific one for wireless ad-hoc networking.Multiple protocols for ad-hoc networking.	Simple programming model for defining new protocol, hiding the complexity of configuring wireless radios and exchanging messages among multiple communication.parties.
Proteus	A geo-distributed framework for analytics computations on federated data stores. Proteus maintains materialized views and performs stateful data-flow computation. Admins place computation and data according to SLA considerations.	Apache Spark,Distributed search for federated clouds, Federated query processing on linked data,Lasp??	Bidirectional data-flow computations using materialized views stored as CRDTs. Modular distributed architecture that enables flexible data and computation placement in geo-distributed systems.
Grisp	A Unikernel approach running the Erlang VM directly on smaller hardware without intervening operating system level. There is a software stack that allows for mixed critical systems with hard and soft real-time parts. A evaluation and development board for this was developed outside the project and provided to partners.	Running Erlang on Embedded Linux like operating systems. Soft real-time only.	Erlang on smaller IoT devices which wouldn't be able to run Linux. Erlang as part of mixed critical systems. Preparation for allowing hard real-time Erlang processes.
Achlys	Autonomous management of generic computations in light edge applications.	SOTA edge applications do not run on sensor networks, but on gateways that manage these networks.	Resilient data storage and resilient communication directly on sensor networks; generic distributed task model; autonomous membership management; function-as-data approach.

4. Relationship to State of the Art

State of the art of LightKone reference architecture (LiRA) is addressed in details in Sections 2 and 3. For completeness, we summarize the relation to state of the art focusing on the data management aspect that is a key contribution of in LightKone. Being a promising extension to cloud computing, fog and edge computing have been very active areas in research and industry. Consequently, several edge/fog architectures have been proposed in SOTA like the OpenFog Reference Architecture (OpenFog RA) [12], Edge Foundry [15], Microsoft Azure IoT [22], Amazon IoT Green grass [4], ECC Edge Computing [11], and ETSI MEC [14]. However, there is an existing gap in all these architectures in the data management level of the application layer in which data cannot be efficiently shared/replicated unless through an upper layer intermediary (a higher layer fog node or the cloud center). This represents a single point of failure and imposes unacceptable response time to edge applications. The main innovation in LiRA is the support for generic application-level data and computation through developing artifacts and software components that support replicated data that is highly available and proven to converge at once. Importantly, LiRA allows data sharing across the hierarchy of the edge system as well as at the same layer. In addition, LiRA artifacts span a wide spectrum of heavy, medium, and light edge/fog devices thus supporting a wide range of applications and patterns.

LiRA is compatible and complementary to SOTA architectures. For instance, the OpenFog RA,[12] is a generic architecture that set standards and recommendations to the required features and properties at the entire software stack. It emphasizes the importance of autonomy and availability without proposing solutions to them at the application layer as in LiRA. In the discussed use cases, OpenFog RA highlights the difficulty of data sharing at the same edge layer, which LiRA provides in particular. On the other hand, Microsoft Azure IoT is based on a time streaming where data is basically pushed to the cloud center for processing. To improve response times, a ”warm” database is used to provide data for edge IoT devices mainly, for a recent date and time range, aggregated data for one or many devices, etc. Therefore, there is no support for generic edge applications semantics or data management at the edge/fog layer as we do in LiRA. ECC Edge Computing follows the Azure IoT approach ensur-

ing a fast time series (centralized) database that stores immutable data associated with timestamp for speed.

Contrary to Azure IoT and ECC Edge Computing, Amazon IoT Greengrass extends Amazon's AWS cloud to edge devices, allowing edge devices to run AWS Lambda functions, execute predictions based on machine learning models with or without connection to the cloud center. Edge nodes can also integrate with third party applications but without data sharing as in LiRA. EdgeX Foundry is another edge framework maintained by Linux Foundation with the ambition to be a key edge/fog open source platform for IoT applications. Data in EdgeX is only handled across layer (north-west) leaving unilateral data management to a future plan. Other platforms like FIWARE FogFlow and ETSI MEC do not make use of replicated data and thus focus more on data at the cloud, databases, or corresponding dissemination.

5. Final Remarks

This paper has presented a first vision of the LightKone Reference Architecture (LiRA) and its initial reference implementation *i*-LiRA. We have discussed how the software artifacts of *i*-LiRA have been used to build different use cases of edge computing across different application domains. We note that LiRA, and naturally *i*-LiRA, are continuous evolving entities that are currently being evolved actively by the LightKone consortium and their partners.

LiRA, and *i*-LiRA, are not closed entities. They are complementary to existing reference architectures, and open for integration with other software artifacts produced by the community. Our main goal with the presentation of LiRA is to identify and address, in a provably correct and efficient way, a key challenge in edge computing: data management. The current proposal of LiRA is based on two simple and complementary ideas: *decentralized lateral data sharing* and *convergent vertical semantics*. These ideas can also be materialized using different techniques, and this will empower the community to have additional flexibility when building their edge-enabled applications.

5. Bibliography

[1] Raka Mahesa (IBM). How cloud, fog, and mist computing can work together. https://developer.ibm.com/dwblog/2018/cloud-fog-mist-edge-computing-iot/, 2018. Accessed: 2018-05-16.

[2] D. D. Akkoorath, A. Z. Tomsic, M. Bravo, Z. Li, T. Crain, A. Bieniusa, N. Preguiça, and M. Shapiro. Cure: Strong semantics meets high availability and low latency. In *Proceeding of the IEEE 36th International Conference on Distributed Computing Systems*, ICDCS'16, pages 405–414, Nara, Japan, 2016.

[3] Paulo Sérgio Almeida, Ali Shoker, and Carlos Baquero. Delta State Replicated Data Types. *J. Parallel Distrib. Comput.*, 111:162–173, 2018.

[4] Amazon. Amazon IoT Greengrass reference architecture, 2018. https://aws.amazon.com/greengrass/.

[5] IEEE Standards Association. IEEE 1934-2018 - IEEE Standard for Adoption of OpenFog Reference Architecture for Fog Computing, 2018. https://standards.ieee.org/standard/1934-2018.html.

[6] Hagit Attiya, Faith Ellen, and Adam Morrison. Limitations of highly-available eventually-consistent data stores. *IEEE Transactions on Parallel and Distributed Systems*, 28(1):141–155, 2017.

[7] Carlos Baquero, Paulo Sérgio Almeida, and Ali Shoker. Making operation-based crdts operation-based. In *Distributed Applications and Interoperable Systems - 14th IFIP WG 6.1 International Conference, DAIS 2014, Held as Part of the 9th International Federated Conference on Distributed Computing Techniques, DisCoTec 2014, Berlin, Germany, June 3-5, 2014, Proceedings*, pages 126–140, 2014.

[8] David Bol. Ecological transition in ict: A role for open hardware? Invited talk at RISC-V 2019, Paris, France, October 2019.

[9] Flavio Bonomi, Rodolfo Milito, Jiang Zhu, and Sateesh Addepalli. Fog computing and its role in the internet of things. In *Proceedings of the First Edition of the MCC Workshop on Mobile Cloud Computing*, MCC '12, pages 13–16, New York, NY, USA, 2012. ACM.

[10] Cisco. Cisco visual networking index: Global mobile data traffic forecast update. https://tinyurl.com/zzo6766, 2016.

[11] Edge Computing Consortium and Alliance of Industrial Internet. ECC Reference Architecture, 2018. http://en.ecconsortium.net/Uploads/file/20180328/1522232376480704.pdf.

[12] Open Fog Consortium. OpenFog Reference Architecture, 2017. https://www.iiconsortium.org/pdf/OpenFog_Reference_Architecture_2_09_17.pdf.

[13] The Syncfree Consortium. Antidote db, 2018. http://syncfree.github.io/antidote/.

[14] ETSI. MEC Reference Architecture, 2018. https://www.etsi.org/.

[15] Linux Foundation. EdgeX Foundry Reference Architecture, 2018. https://docs.edgexfoundry.org/Ch-Architecture.html.

[16] Seth Gilbert and Nancy Lynch. Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services. *SIGACT News*, 33(2):51–59, 2002.

[17] Leslie Lamport. Time, Clocks, and the Ordering of Events in a Distributed System. *Commun. ACM*, 21(7), July 1978.

[18] J. Leitão, J. Pereira, and L. Rodrigues. Epidemic broadcast trees. In *Proceedings of the 26th IEEE International Symposium on Reliable Distributed Systems*, pages 301 – 310, Beijing, China, October 2007.

[19] João Leitão, José Pereira, and Luis Rodrigues. HyParView: A membership protocol for reliable gossip-based broadcast. In *Dependable Systems and Networks, 2007. DSN'07. 37th Annual IEEE/IFIP International Conference on*, pages 419–429. IEEE, 2007.

[20] Christopher Meiklejohn and Heather Miller. Partisan: Enabling cloud-scale erlang applications. *arXiv preprint arXiv:1802.02652*, 2018.

[21] Christopher Meiklejohn and Peter Van Roy. Lasp: A language for distributed, coordination-free programming. In *Proceedings of the 17th International Symposium on Principles and Practice of Declarative Programming*, pages 184–195. ACM, 2015.

[22] Microsoft. Azure IoT reference architecture, 2018. http://download.microsoft.com/download/A/4/D/A4DAD253-BC21-41D3-B9D9-87D2AE6F0719/Microsoft_Azure_IoT_Reference_Architecture.pdf.

[23] Nuno Preguiça, Carlos Baquero, and Marc Shapiro. Conflict-free replicated data types. To appear in Encyclopedia of Big Data Technologies.

[24] Marc Shapiro, Nuno Preguiça, Carlos Baquero, Marek Zawirski, et al. A comprehensive study of convergent and commutative replicated data types. 2011.

[25] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu. Edge computing: Vision and challenges. *IEEE Internet of Things Journal*, 3(5):637–646, Oct 2016.

[26] W. Tärneberg, V. Chandrasekaran, and M. Humphrey. Experiences creating a framework for smart traffic control using aws iot. In *2016 IEEE/ACM 9th International Conference on Utility and Cloud Computing (UCC)*, pages 63–69, Dec 2016.

[27] Shanhe Yi, Cheng Li, and Qun Li. A survey of fog computing: Concepts, applications and issues. In *Proceedings of the 2015 Workshop on Mobile Big Data*, Mobidata '15, pages 37–42, New York, NY, USA, 2015. ACM.