Project no.	732505
Project acronym:	LightKone
Project title:	*Lightweight computation for networks at the edge*

D3.2: Runtime System for Edge Computing with Application Support

Deliverable no.:	D3.2
Title:	Runtime System for Edge Computing with Application Support
Due date of deliverable:	January 15, 2019
Actual submission date:	January 15, 2019
Lead contributor:	INESC TEC
Revision:	2.0
Dissemination level:	PU
Start date of project:	January 1, 2017
Duration:	36 months

This project has received funding from the H2020 Programme of the European Union

Revision Information:

Date	Ver	Change	Responsible
15/01/2019	2.0	Revision ready for submission	INESC TEC
15/10/2018	1.1	Revision started	INESC TEC
27/06/2018	1.0	Ready for submission	INESC TEC

Revision information is available in the private repository https://github.com/LightKone/WP3.

Contributors:

Contributor	Institution
Ali Shoker	INESC TEC
João Marco Silva	INESC TEC
Georges Younes	INESC TEC
Annette Bieniusa	TUKL
Peter Van Roy	UCL
Igor Zavalyshyn	UCL
Vitor Enes	INESC TEC
Carlos Baquero	INESC TEC
Adriaan Leijnse	INESC TEC
Sébastien Merle	STRITZINGER
Adam Lindberg	STRITZINGER
João Leitão	NOVA
Nuno Preguiça	NOVA
Pedro Ákos Costa	NOVA
Pedro Fouto	NOVA

Contents

Chapter 1

Executive Summary

This deliverable (D3.2) presents the data and communication abstractions and components that can be used to build a generic edge computing runtime, as well as the cross-cutting concerns of LightKone work packages (WPs). LightKone advances state of the art of edge/fog computing through providing highly available and scalable replicated data management following the recent advances of Conflict-free Replicated DataTypes (CRDTs) and their supporting dissemination middlewares. Consequently, in this deliverable, corresponding to work package 3 (WP3), we convey the novel contributions on CRDTs as well as the underlying dissemination layer. Most of the works are progress to the work done started in Year 1, and delivered in D3.1. The report also presents a security threat analysis to LightKone use cases. The core contributions can be summarized as follows:

Data Abstractions. We continued the work to improve the efficiency of CRDTs through compressing the meta-data and thus reducing storage and dissemination. We have also started exploring a new model-agnostic approach to build CRDTs by composing basic primitive types.

Communication Protocols. We continued improving the causal middleware we started in D3.1 by optimizing the internal data structures for tagging operations. This was inspired by another study we made to present the pitfalls of classical implementations. We also continued the work on delta synchronization protocols and Partisan communication library through presenting their empirical evaluations.

Partial Replication. We continued the work started on Saturn's partial replication through extending it in two directions. The first (C3) to include more parallelism inside datacenters or heavy edge clusters, and the second (Gesto) that significantly lowers the intra-region migration latency at the cost of slightly increasing the size of the metadata.

Threat analysis. We provide a full security threat analysis to LightKone use cases.

The report also presents exploratory research not at the core of LightKone, and presents the software deliverables.

Chapter 2

Introduction

With the immense volumes of data generated and computed at the data centers and cloud servers, there is a need to move part of storage and computation towards the edge of the network. Indeed, the benefits turned out to be many, among them to: reduce the load on the data centers, reduce the amount of data traveling across continents, and improve the availability security of data, etc. However, the edge computing approach brings its own challenges like scalability, heterogenity, resilience, and security. In WP3, we develop the data abstractions and communication protocols that address the above challenges and support building generic edge computing runtimes as those developed in WP5 and WP6.

2.1 General Motivations and Approach

Contrary to Fog Computing [43], where sharing data is done through upper fog layers, the challenge we address in LightKone is to have the data and computation laterally distributed over loosely coupled machines without violating the semantics and properties of the applications. Having loosely coupled machines is crucial to provide availability even if parts of the network or nodes are unreachable, due to the fragile networks and commodity hardware used at the edge (contrary to data centers that are very well engineered and thus exhibit low failure rates). This strongly suggests using a relaxed data model provided that concurrent updates follow a clear conflict resolution mechanism. An interesting approach is to use Conflict-free Replicated DataTypes (CRDTs) [39] that are proven to ensure convergence in geo-replicated settings. In LightKone, we build on the success of CRDTs and extend them to address the new challenges at the edge considering the related subjects of dissemination and performance. In this work package (WP3), we focus on building the generic data and communication abstractions for developing generic edge computing runtimes. Building on these abstractions, WP5 and WP6 focus on how to build systems for light and heavy edge settings, respectively.

During the first year of LightKone, WP3 focused on scalability and efficiency at the edge, providing the initial data abstractions and communication protocols. This very deliverable, i.e., D3.2, is an extension to D3.1 presenting the progress of previous tasks with focus on convergence and communication for edge application support, and security specification and corresponding security models.

2.2 Contributions

We summarize the key contributions of WP3 for the first six months of Year 2 (Y2). The contributions follow a plan that roughly follows the LightKone proposal with more details as presented in Chapter 3. These contribution can be summarized by the following.

In Section 3.1.2, we start presenting the data abstractions for edge computing focusing on the Conflict-free Replicated Data Types (CRDTs) [39] approach. CRDTs represent the main data ingredient within LightKone as they guarantee convergence in a relaxed consistency models. While CRDTs are already in use in geo-replicated systems, the challenges at the edge are different due to hostile edge networks, scalability, and constrained devices. In deliverable D3.1, we targeted CRDT data abstractions that are tailored towards generic edge computing by addressing some of the mentioned challenges. In particular, we focused on improving the scalability and efficiency of the two CRDT models, i.e., state-based and operation-based, which can be used in different edge networks and under different assumptions. During these six months, and in addition to improving these models, we were able to understand the edge application space and patterns more by studying the formalizations in deliverable D2.2.

In particular, and to further understand the CRDT models, the consortium published several works on the fundamentals of CRDTs and the different models (state-based, operation-based, delta based, and pure operation-based), showing their properties and relation between them. Beyond CRDT models, the consortium introduced a novel model-agnostic language to define CRDTs regardless of the underlying infrastructure, and thus lead to better interoperability in data abstractions of different systems, and maybe address hybrid edge networks.

The above data abstractions rely on a communication layer that guarantees specific assumptions and properties. More specifically, modern applications edge applications adopt relaxed consistency models and thus tolerate reading stale data for better response time; however, many applications resort to causal consistency as the strongest relaxed consistency model that supports available applications, and provides intuitive application logic (e.g., respects dependencies). In Section 3.4, we emphasize this requirement and we present the contributions towards providing communication middlewares and abstractions that can provide causal consistency, delivery, and stability. Furthermore, we present the progress on communication causal middlewares and Partisan library started in deliverable D3.1, focusing on the implementation and evaluation details.

In addition, given the limited capacities of the constrained edge devices in many use-cases, it is not affordable to store big amounts of data, and thus there is a need to partition the data across edge nodes and use relaxed consistency acorss partitions to improve availability. To address this issue, we addressed in Section 3.5 the limitations of Saturn at the edge and developed another version called Gesto. Gesto uses partial replication with efficient meta data handling to improve the performance inside and outside a datacenter or heavy node. In line with this work, we extended Saturn, in C3, to allows for higher concurrency inside a datacenter or a heavy edge, thus boosting its availability.

In Sections 3.6 and 3.7 we present the software components in which part of the above works is implemented, and we conduct a literature review highlighting how this report advances state of the art.

In the final two chapters, we present a summary of the security analysis that is fully included in the Appendix. In particular, we studied the use case formalization in deliverable D2.2, and we provided a thorough threat model analysis to LightKone use-cases. We also tried to give security recommendations to use from state of the art. Finally, we present further exploratory research that is not at the core of LightKone. The works are mainly on the development of CRDTs and security contributions. We discuss the relation of these works to LightKone.

2.3 Relation to other WPs

This work package addresses the cross-cutting concerns across all other work packages and thus it has a natural overlap with most of them. The data and communication abstractions in this wor stand as generic components and techniques to build light and heavy edge runtimes as in WP5 and WP6. For instance, the designed CRDT libraries in this deliverable represent the building blocks of the data models in D4.1 and implementations in D5.1 and D6.1. In particular, AntidoteDB is developed in WP6 and uses the optimized op-based CRDTs, whereas Lasp and Legion developed in WP4 and WP5 uses the delta- and state-based CRDTs. Similarly, some of the communication protocols developed in WP3 like Partisan is being used as the underlying communication layer of Lasp. Finally, the security analysis in this deliverable builds on the use cases presented in D2.1 and formalizations in D2.2.

2.4 Summary of Deliverable Revision

This deliverable has been revised since its original submission to incorporate comments and modifications requested by the European Commission Reviewers. The main changes made to the deliverable are as follows:

- Introduced a literature review section in which we explain the advancement beyond state of the art.
- Introduced the plan we followed in Year 2 and the future plan to achieve the mentioned milestones.
- Revised most sections to improve presentation with relative mentioning to SOTA, the contribution, how it advances beyond SOTA according to the plan, and the respective future plan.
- Removed all publications in the appendix (with the exception of the threat analysis of use cases), and dedicated a section for dissemination activities carried in the context of this work package.

2.5 Organization of the Report

The rest of the report is organized as follows:

Chapter 3: Plan and Progress presents the plan we followed in WP3 and the work progress considering the data and communication abstractions and partial replication.

The chapter also presents the advances beyond state of the art as well as the software deliverables list.

Chapter 4: Security Analysis of Use-cases presents the summary of LightKone use case security analysis.

Chapter 5. Exploratory Research presents other research works that are minor to LightKone.

Chapter 6: Annotated Publications & Dissemination presents a synopsis of the publications and dissemination in WP3.

And finally, Appendix A includes the full security analysis of Chapter 4.

Chapter 3

Plan and Progress

Data abstractions are at the heart of edge computing in LightKone. Unlike classical Peer-to-Peer (P2P) system that managed mostly immutable data, the essence of LightKone is to support mutable data at the edge of the network. To this end, LightKone adopts the synchronization-free approach in Conflict-free Replicated Data Types (CRDTs) [41] that is proven successful in geo-replicated systems: it favors availability without compromising global data convergence. To some extent, distributed data in edge computing can be seen as an extension to approaches used in geo-replicated systems, with the need to consider a much larger number of devices, their capacity, network bandwidth, mobility, and heterogeneity. This poses new challenges that require a better understanding of the different CRDT models and extension for addressing different requirements. In this chapter, we introduce the main contributions on data abstractions for edge computing, mainly through using CRDTs, addressing the three main categories: light, heavy, and hybrid edge. In particular, we focus on improving the CRDT models for addressing the challenges of different edge scenarios and use-cases.

We start this chapter by providing the plan we followed during the past six months as well as the one we will follow in the rest of the project. We organize this chapter into four main sections. The first section depicts novel works towards understanding the fundamentals and models of CRDTs. Fundamentals of CRDTs are important to develop new variants and models that support edge use-cases and applications. This section, being at the very begining, serves as a background to understand the following works. In the same section, we go further and show that model-agnostic CRDTs can be an alternative to compose CRDTs from few primitive types. While state-based and op-based CRDT variants can be used in light and heavy edge, model-agnostic CRDTs help on the interoperability of these models which can be useful in the hybrid edge. The second section presents a progress report to two CRDT variants: the Pure op-based and the Delta state-based CRDT models presented in D3.1, focusing on implementations and evaluations. The third section presents the communication middlewares supporting the above models focusing on application level properties like causal consistency. In the fourth section, we address the scaling challenges of CRDTs at the edge. We present two extensions to the partial replication model of Saturn started in Year 1. We finaly present the software deliverables and the advancement beyond state of the art.

3.1 Plan and Milestones

We first present the plan we followed during these six months of Year 2 and the plan we put for the rest of the project. The plan continues the development of the data and communication abstractions of generic edge runtime with application-level support focusing on data convergence and consistency. Our plan is often consistent with that presented in the original proposal (unless explicitly mentioned otherwise) and deliverable D3.1. We also took into consideration the support for generic edge computing runtimes considering the LightKone use cases and pushing the envelope further to explore and understand the potential of LightKone technology beyond our use cases. We believe that is important to create more impact and innovate.

3.1.1 Plan for the first half of Year 2

At the end of Year 1, we introduced the basic data and communication abstractions necessary to build generic edge runtimes. We also identified three interesting edge/fog model considering the use cases analysis and their application support. We concluded that considering the different use cases, three patterns for edge/fog applications can be identified. The first is Heavy edge where applications communicate and use data across fog nodes, i.e., north-south in both directions. The second model is Light edge where edge nodes communicate and share data laterally, at the same edge/fog level, a.k.a., east-west. We also identified an interesting Hybrid edge model in which applications include both north-south and east-west communication and data sharing (replication) patterns. In this vein, in the first six months of Year 2, we planned to continue our work through addressing the application semantics already started in Y1. In particular, we wanted to continue improving the efficiency of the different CRDT models through optimizing the datatypes, mainly the pure operation-based and delta-state CRDTs, as well as their underlying communication layer. Our aim was to focus on the impact of the causal broadcast on concurrent applications and identified potential implementation pitfalls. We also planned to evaluate the state synchronization in the delta CRDT model and conduct empirical assessment to the performance of Partisan library. We also continued the work started on partial replication by addressing correctness and edge-tailored featured identified in the previous report D3.1. Finally, we planned to conduct a full security analysis to the LightKone use case threat models as cotinuation of the work doen in D3.1.

3.1.2 Plan for the second half of the project

Considering the comments of the reviewers of the European Commission, the plan for the second half of the project was subject to an update to design the Reference Architecture (LiRA) that governs the entire project. This work was part of WP3, and was presented as a revision for deliverable D3.1. Our plan for the end of Year 3 is to improve the performance and resilience of the developed protocols. Specifically, we aim to increase the scalability of CRDTs (to the number of edge nodes) to one order of magnitude. Current CRDTs often include meta data as version vectors that scale linearly with the size of edge network. We are planning to explore the idea of identity containment introduced in Handoff counters in D3.1. This will also bring other benefits in inter-operability across edge models. Another direction to address scalability is through extending the idea of

Borrow Counters developed in Year 1 to other datatypes. The purpose is to mitigate the identity explosion problem especially once nodes are transient or upon high churn rate. In line with this work on dynamic networks, we aim to develop CRDTs that can adapt with churn, e.g., using elastic version vectors. Finally, we will explore the potential of Hybrid edge applications defined in the previous chapter building on the outcome of WP5 and WP6 on Light and Heavy edge. A potential direction is to explore the EdgeAnt approach started in WP6. EdgeAnt supports data sharing across edge nodes at different levels and there is plan to extend this to include lateral data sharing at the same edge level. At the end of the project, we will deliver packaged libraries and protocols that can be used beyond LightKone.

3.2 Understanding CRDT Models

As a major building block for data abstractions in LightKone, Conflict-free replicated data types (CRDTs) [41] are yet to be fully understood especially at the Edge. Consequently, the consortium is in a continuous search for new models that are more efficient, sustainable, and easy to understand and implement. In this section, we present some of the works in that direction. The first explores CRDT fundamentals in a comprehensive way, the second goes deeper into advanced CRDT models, and the third presents a novel abstract conceptional model-agnostic approach to compose CRDTs given few primitive CRDTs.

3.2.1 Overview on Conflict-free replicated data types

Distributed systems often replicate data at multiple location for providing fault-tolerance, high availability and low latency. Systems that adopt a weak consistency model provide high-availability and low latency, by allowing a client to contact a single replica, which can execute the client's operation without coordinating with other replicas. This may lead replicas to temporarily diverge, requiring a mechanism for merging concurrent updates into a common state.

Conflict-free Replicated Data Types (CRDT) provide a principled approach to address this problem. A CRDT is an abstract data type, with a well defined interface, designed to be replicated at multiple nodes and exhibiting the following properties: (i) any replica can be modified without coordinating with any other replicas; (ii) when any two replicas have received the same set of updates, they reach the same state, deterministically, by adopting mathematically sound rules to guarantee state convergence.

Conflict-free replicated data types (CRDTs) are at the core of the solutions adopted in this project. Members of this project have proposed the concept, first by proposing a CRDT for concurrent editing [37] and later laying the theoretical foundations of CRDTs [41]. Since then, CRDTs have become mainstream and are used in a large number of systems serving millions of users worldwide. Currently, an application can use CRDTs by either using a storage system that offers CRDTs in its interface[1], by embedding an

[1]Examples of storage systems that offer CRDTs in their APIs include Riak distributed database (http://basho.com/products/riak-kv/), Redis CRDBs (https://redislabs.com/redis-enterprise-documentation/developing/crdbs/) and Akka distributed data (https://doc.akka.io/docs/akka/current/distributed-data.html).

existing CRDT library or implementing its own support.

During this period, members of the project have written several documents that present CRDT for general audiences, namely: (i) an entry on CRDTs for the Encyclopedia of Big Data Technologies [39]. (ii) an overview on CRDTs, as part of the documents produced by Nuno Preguiça for obtaining the *Agregação em Informática* degree (similar to *Habilitation in Computer Science*) [38].

These documents present on overview of CRDT's research and practice, and they have been written with the purpose of serving as a tutorial for general audiences and as a survey for researcher.

3.2.2 Advanced CRDTs

CRDTs follow two different approaches which both lead to convergence (eventually) [41]. The first is the operation-based (i.e., op-based) model in which operations are disseminated to other replicas via a reliable causal middleware. The second model is the state-based model, that builds the state as a *join semi-lattice* where special functions, called mutators, use the application operations to *inflate* the state; in this model, the entire state is propagated and merged remotely.

Besides these two base model, there have been proposals of other CRDT models, with the most known being the Pure op-based CRDTs [8] and the delta-state (simply delta) CRDTs [3]. The pure op-based model is a variant of op-based CRDT where the disseminated data is "only" the operation submitted by the client (by opposition with the classical op-based CRDT, where the propagated operation can be a modified version generated after accessing the current object state). Upon receipt, the operations are stored in a partial-ordered log (POLog) and enter into a pruning process to reduce the state to a compact one. The delta CRDT model improves on the state-based model by propagating only "recent updates" rather than the entire state. This brings significant benefits on bandwidth utilization.

The above four models represent the leading edge of CRDTs, and therefore, there is a need for more understanding and dissemination to improve their adoption. Inline with this, we are in the final stages of publishing a section on CRDTs in the NESUS Action [2] book on ultrascale computations, within the "Data Management Techniques" chapter (to appear) (to appear).The contribution presents the four models in an incremental way following a common example across the chapter. The aim was to help the reader understand the models and their differences in an easy and comprehensive way. The contribution also included an application section to a distributed object oriented framework, called dataClay [3] that tackled two points:

- a simple implementation of a Counter CRDT showing how it integrates in the system; and
- applying the concept of causal stability to activate distributed class versions consistently.

[2]EU NESUS Action: http://www.nesus.eu/

[3]dataClay: https://www.tdx.cat/bitstream/handle/10803/405907/TJMF1de1.pdf?sequence=1&isAllowed=yi

3.2.3 Towards a model-agnostic CRDTs

In a radically different approach, we are currently working on an model-agnostic CRDT which allows the understanding and construction of CRDTs as type abstractions regardless of a particular implementation model, e.g., the op-based and state-based variants in the previous sections. Indeed, CRDTs in these models are closely linked to the type of communication layer used for their implementation. For example, state-based CRDTs are defined in such a way that arbitrary and ad-hoc communication between actors is made possible, while operation-based CRDTs assume exactly-once causal delivery of messages between actors. This leads to CRDTs which have the same semantics being defined in multiple ways.

In this work in progress, with more details explained in Chapter 5, we show that CRDTs can be defined in a communication-agnostic way using a limited number of primitives. The primitives are chosen in such a way that they abstract over common patterns in the semantics of CRDTs, and we aim to choose only primitives that can be compiled to efficient implementations. That is, the definitions should be mechanically translatable to an optimal implementation in an existing model. We identified four common patterns and defined four primitives which we will use to describe nine common CRDTs. Further details are explained in Chapter 5. We are exploring this approach to understand of it can help inter-operability between state and op-based CRDTs. We plan to continue exploring this topic in the future with the help of WP4.

3.3 Further Progress on CRDT Models

In this section, we continue the discussion on CRDT models focusing on the technical details and evaluation of Pure op-based CRDTs and delta-based CRDTs. This extends the research reported in deliverable D3.1.

3.3.1 Pure Operation-based CRDTs at the edge

In deliverable D3.1, we introduced our results on Pure op-based CRDTs. Since then, we have proceeded to revise and improve the content in the associated journal submission after receiving feedback from the reviewers.

For pure op-based CRDTs the state is organized as a *partially ordered log of operations* (POLog). In systems that do not require a full history of changes, it is important to compact this POLog in a way that preserves the results of queries while reducing the stored state. In our initial design, reported in D3.1, we had two mechanisms for compaction. The first compared operations that were causally delivered to a destination node with the operations already in its local POLog and removed potential redundancy. The second builds on the notion of *causal stability* that allows each destination site to identify when all operations that can be concurrent to a given operation in the POLog have already been delivered there, allowing further compaction.

In the revision of the paper we have identified that a third case of compaction can be provided by checking at the source if a new operation is redundant with respect to the operations in the source POLog. In this case the operation is not even transmitted to destinations and is effectively pruned at the source. To justify this improvement we have designed a new, non trivial, CRDT that benefits from detecting *redundancy at the* *source*. It provides a "maximum of values written under an evolving limit", having two update operations: $[\mathsf{wr}, v]$ (write value) and $[\mathsf{lim}, v]$ (set limit), and a query operation rd (read value). It assumes that values are totally ordered (e.g., integers), and the rd query returns the maximum of the set of values that were "accepted" under the limit at the time they were written. To decide if a value written is accepted, and given the possibility of concurrent limit setting operations, the limit is updated as a multi-value register which returns the maximum of its values when queried.

3.3.2 State-based CRDTs at the edge

In the previous deliverable, D3.1, we identified two sources of inefficiency in current synchronization algorithms for delta-based CRDTs, and proposed two optimizations: *avoid back-propagation of delta-groups* (BP) and *remove redundant state in received delta-groups* (RR). We also introduced the concept of join-decomposition of state-based CRDTs, essential for the RR optimization. We have been working to find sufficient conditions for the existence these decompositions, showing how they can be obtained (proofs are skipped), and present an evaluation of the optimizations proposed.

(a) Existence of Unique Irredundant Decompositions

Proposition 1 *In a distributive lattice $\mathcal{L}$ satisfying the descending chain condition (DCC), every element $x \in \mathcal{L}$ has a unique irredundant join decomposition.*

Proposition 2 *If $\mathcal{L}$ is a finite distributive lattice, then the irredundant join decomposition of $x \in \mathcal{L}$ is given by the maximals of the join-irreducibles below x.*

Most CRDT designs define the lattice state starting from booleans, natural numbers, unordered sets, and obtain more complex states by lattice composition through: cartesian product $\times$, lexicographic product $\boxtimes$, linear sum $\oplus$, finite functions $\hookrightarrow$ from a set to a lattice, powersets $\mathcal{P}$, and sets of maximal elements $\mathcal{M}$ (in a partial order). Typical CRDT designs simply consider building join-semilattices from join-semilattices, never examining whether the result is more than a join-semilattice. In fact, all these constructs yield lattices satisfying DCC, when starting from lattices satisfying DCC (such as booleans and naturals). Also, it is easily seen that most yield distributive lattices when applied to distributive lattices, with the exception of the lexicographic product with an arbitrary first component. Fortunately, the typical use of lexicographic products to design CRDTs is with a chain (total order) as the first component, to allow an actor which is "owner" of part of the state (the single-writer principle) to either inflate the second component, or to change it to some arbitrary value, while increasing a "version number" (first component). In such typical usages of the lexicographic product, with a chain as first component, the distributivity of the second component is propagated to the resulting construct. Table 3.3.1 summarizes these remarks about how almost always these CRDT composition techniques yield lattices satisfying DCC and distributive lattices, and thus, have unique irredundant decompositions, by Proposition 1.

Having DCC and distributivity, even if it always occurs in practice, is not enough to directly apply Proposition 2, as it holds for finite lattices. However if the sublattice given by the ideal $\downarrow x = \{y \mid y \sqsubseteq x\}$ is finite, then we can apply that proposition to this finite lattice (for which x is now the top element) to compute the decomposition. Again,

	$\mathcal{L}$						
	$A \times B$	$A \boxtimes B$	$C \boxtimes A$	$A \oplus B$	$U \hookrightarrow A$	$\mathcal{P}(U)$	$\mathcal{M}(P)$
A, B, P has DCC $\Rightarrow \mathcal{L}$ has DCC	✓	✓	✓	✓	✓	✓	✓
A, B distributive $\Rightarrow \mathcal{L}$ distributive	✓	✗	✓	✓	✓	✓	✓

Table 3.3.1: Composition techniques that yield lattices satisfying DCC and distributive lattices, given lattices A and B, chain C, partial order P and (unordered) set U.

	$\mathcal{L}$						
	$A \times B$	$A \boxtimes B$	$C \boxtimes A$	$A \oplus B$	$U \hookrightarrow A$	$\mathcal{P}(U)$	$\mathcal{M}(P)$
$\forall x \in \mathcal{L} \cdot x/\bot$ finite	✓	✗	✗	✗	✓	✓	✓
$\forall \langle x, y \rangle \in \mathcal{L} \cdot \langle x, y \rangle / \langle x, \bot \rangle$ finite	–	✓	✓	✓	–	–	–

Table 3.3.2: Composition techniques that yield finite ideals or quotients, given lattices A and B, chain C, partial order P, all satisfying DCC, and (unordered) set U.

finiteness yields from all constructs, with the exception of the lexicographic product and linear sum. For these two constructs, a similar reasoning can be applied, but focusing on a quotient sublattice in order to achieve finiteness (Given elements $a \sqsubseteq b \in \mathcal{L}$, the *quotient sublattice* b/a is given by $b/a = \{x \in \mathcal{L} \mid a \sqsubseteq x \sqsubseteq b\}$). Table 3.3.2 summarizes these remarks; the second row applies only to lexicographic products and linear sums.

(b) Evaluation

In this evaluation we compare classic delta-based synchronization against state-based, and show the benefits of employing BP and RR optimizations. The evaluation takes place in a Kubernetes cluster with 16 Quad Core Intel Xeon 2.4 GHz, deployed in Emulab. We have designed a set of of micro-benchmarks, with update operations on three distinct data types (*grow-only set*, *grow-only counter*, and *grow-only map*). The topologies employed are a partial-mesh with 16 nodes, each node with 4 neighbors; and a tree with 14 nodes, each node with 3 neighbors, with the exception of leaf nodes. With this evaluation, we conclude that:

- **in terms of transmission:**
  - in all configurations, classic delta-based represents almost no improvement, when compared to state-based;
  - in the tree topology, BP is enough to attain the best result, since the underlying topology does not have cycles;
  - with a partial-mesh, BP has little effect, and RR contributes most to the overall improvement.
- **in terms of memory:**
  - the size of delta-groups being propagated, not only affects the network bandwidth consumption, but also the memory required to store them in the buffer

for further propagation; with the proposed optimizations, we improve previous approaches by up-to 270% less memory, in some cases having almost no overhead when comparing to the optimal case (state-based).

- **in terms of processing time:**
  - since our solution with the optimizations proposed sends less (redundant) information, it has consistently lower processing overhead than classic delta-based synchronization;
  - if most of the CRDT state is updated between synchronization rounds, the processing overhead is higher for our solution when compared with both delta and state-based alternatives: this is explained by the fact that RR tries to remove redundant state when there is almost none (we note however that this is an extreme workload that we do not expect it to be a common case).

3.4 Communication Abstractions for Edge Computing

As it has been discussed in the previous section, data abstractions are often designed assuming some set of guarantees from the underlying communication layer. In this section, we present the design of communication layers by addressing two main aspects. The first is how to provide causal delivery, which is the base for providing causal consistency, the strongest consistency model that allows for available (non-blocking) applications [6]. In particular, we discuss the end-to-end causal delivery and pitfalls that can arise in distributed edge applications and we present a corresponding solutions. The second aspect we address is the design and implementation of the communication layers, by presenting the progress in the work previously introduced in deliverable D3.1. In particular, we present technical details on implementing causal delivery and stability in causal middlewares. It shows how to implement causal delivery and stability in op-based models that are likely to be used in heavy edge. We then present and empirical evaluation to Partisan communication library that implements protocols for message dissemination through hybrid gossip and group membership, known for their efficiency and resilience. These works are extensions to those presented in deliverable D3.1.

3.4.1 End-to-End Causal Delivery

Traditional causal delivery middlewares [10] provides a delivery order in each process that is *consistent* with causality, i.e., delivering messages at each process in some order which does not contradict causality. However, it does not provide client applications with knowledge about concurrency under the partial order of causality. Given two messages m_1 and m_2 delivered in sequence to some process, no information is provided to the application whether m_1 causally preceded $m2$ or if they were originated independently of each other.

Providing such knowledge to the application is a mandatory feature for several classes of applications that require knowledge of concurrency in order to apply arbitration rules [14]. An example is an application that given two concurrent bids will arbitrate to consider only the higher bid. Since current implementations of causal delivery middleware lack this feature, they are only suitable for a more limited class of applications where this knowledge is not needed, and thus cannot be used as a general abstraction.

We focused since the last deliverable on making a case for a *Tagged Causal Delivery* (TCD) middleware, which delivers messages together with causality tags (e.g. logical clocks) that characterize the *end-to-end* happens-before relation, defined according to client-visible events in the client process order, and ignoring purely internal middleware events (such as receiving a message and queuing it for later deliver).

This work on TCD was reflected in our implementation of the TCSB middleware and also is under submission in a paper that explains the aforementioned problem, the pitfalls in trying to achieve this end-to-end causal delivery using traditional middleware and presenting TCD as a solution. Next, we summarize those pitfalls and the TCD middleware as a solution.

(a) The pitfalls of exposing middleware timestamps

At first glance, it might seem trivial to implement tagged causal delivery, by using any classical causal delivery middleware service and exposing to the client code the timestamps (e.g., vector-clocks) that are used internally to ensure causal delivery. However, when considering concrete implementations, some unexpected problems arose. We show how naively exposing the timestamps would lead to an incorrect characterization of causality, in either of the two typical interaction models between middleware and client code: callback-based and with independent threads/processes.

(a).1 Callback-based In an event-driven architecture with a single process, the application code runs as callbacks invoked from the middleware code when messages need to be delivered to the application logic to be processed, e.g., $deliver(m,t)$ for message m tagged by t timestamp. To avoid reentrancy problems, when a send is invoked inside the deliver callback, the send simply adds the message to a queue, to be handled by middleware code when the callback finishes. It can happen that the middleware has a set of messages ready to be delivered, and invokes the deliver callback for each one, before handling sends which have been enqueued. If the middleware creates timestamps for messages to be sent only upon dequeuing them, then a message will be tagged as causally in the future of all messages that were delivered after the send action by client code and before dequeuing occurred. This means that some messages that are actually concurrent are tagged as causally related, making timestamps reflect a larger relation than happens-before, over-ordering some events. While this does not break causal delivery, it means that these timestamps cannot be exposed as precisely characterizing happens-before.

(a).2 Independent threads/processes/actors In other architectures we have two independent processes, a client process and a middleware process. Here, in addition to the queue of messages to be sent, as above, we will typically also have a queue of messages ready to be delivered. The middleware tags and enqueues messages to the deliver queue, while the client dequeues and processes them. When doing a send, the client enqueues a message to the send queue. This message will be tagged by the middleware process as in the future of other messages not yet delivered by the client (namely, those that are still in the delivery queue but have already been tagged), when they are in fact concurrent. Note that what defines the happens-before is the total order of send and deliver events as observed by each client process; other events, e.g., when a message was enqueued or dequeued by the middleware process, are irrelevant (i.e., events which happen in the system, but invisible to the API).

(b) Correct tagging of happens-before

To correctly characterize happens-before, a message being sent must be tagged reflecting the causal knowledge according to all delivery events at the client, up to the send event (at the client).

In the single process callback-based model, this can be achieved by making the middleware update the causal timestamp (e.g., vector-clock) just before invoking each deliver callback, and either making the send function tag the message before enqueuing, or making the middleware process all messages enqueued to be sent before invoking the next deliver callback.

As for the two independent processes model, the client process will need to maintain the causal timestamp, update it at each deliver event and use it to tag each message. The middleware process keeps a causal timestamp as before and uses it for the causal delivery order. The only difference is that now messages are tagged at the client process and not at the middleware process.

(c) A welcome side-effect

In classical causal delivery middleware, some concurrent messages are tagged as in happens-before relation to each other. This unnecessary over ordering, while not contradicting causal delivery, has the adverse effect of inducing extra delays on delivery, making some message wait for another when it should be possible to deliver it earlier. In tagged causal delivery middleware, which characterizes happens-before in a precise way, as we discussed above, each message will be able to be delivered as early as semantically possible.

3.4.2 Progress on Communication Layer Abstractions

We now present two contributions at the communication layer addressing different settings. The first shows how to implement causal delivery and stability in op-based models that are likely to be used in heavy edge. The second one presents the evaluation of Partisan library and protocol. These works are extensions to those presented in deliverable D3.1.

(a) Implementing Causal Delivery and Stability

In the previous deliverable, i.e., D3.1, we introduced TCSB (Tagged Causal Stable Broadcast) middleware that is a main component of pure operation-based CRDTs. We explained causal delivery and other concepts such as PO-Log (Partially-Ordered Log), sequential data type and causal stability. We discussed how causal stability would lead to a more compact state by discarding existing causality information that can be safely be removed once “stable”.

Our previous implementation consisted of using vector clocks as timestamps to track causality in the middleware and achieve causal order delivery of operations. Also regarding the causal stability mechanism, it was achieved using what we called a Recent Timestamp Matrix (RTM), a local data structure at each node i, where each entry consists of the last known (delivered) operation vector clock form every node j. Using that matrix, we could calculate a stable vector s as the component-wise minimum. Every operation with a vector clock v is stable *iff* $v \leq s$.

In order to move towards the edge and support a larger number of nodes, reducing the size of vector clocks was crucial (as their size grows linearly with the number of nodes). We were working since the last deliverable on using dependency dots, that are more compact and efficient than vector clocks.

The main idea behind the use of dependency dots is that vector clocks contain a lot of causality information that is not needed to track causality between operations. Instead of sending a vector clock with N entries where each represents the changes seen from each process p, we need to only send $Q \leq N$ entries of the processes that changed since the last broadcast. This solution provides more compact timestamp that grows linearly with the number nodes $Q \leq N$ from which operations where delivered since the last local broadcast. Moreover, as each process eventually delivers all operations, we can compact even more. We applied transitive reduction on the previous compact timestamp, which allowed reducing the previous number of changes using causality between the delivered operations. As a result to that, we have new more compact timestamps that we call dependency dots which only requires sending entries for concurrent operations delivered since the last broadcast.

The idea is not completely new as there exist works that use some compressed versions of vector clocks. However, this led to different changes in the middleware, stability mechanism as well as allowed us to support dynamic membership instead of being limited to static membership. For instance, the use of dependency dots led to changes in the delivery mechanism of operations. We expect that this would make delivery faster and we plan on showing that in future experiments.

(b) Partisan

Partisan is a distributed communication library that provides an alternative communication layer for Erlang and Elixir. Partisan has a simple membership API and rich support for backend modules for different network topologies: static, full mesh, client-server, peer-to-peer, and publish-subscribe through implementing hybrid gossip protocols like HyParView and Plumtree [31]. This allows for a robust and easy to use communication layer for edge applications, especially in hostile networks and dynamic memberships. For this reasosn, Partisan is being used as the backbone of Lasp edge runtime. These interesting features are also demonstrated by the rapid adoption in the industry. In fact, Partisan is currently being used as part of product development by four industry adopters that we know of: Leapsight Bondy, and three more that do not wish to be mentioned by name in this public report.[4]

In a nutshell, Partisan is a distributed programming model and distribution layer for Erlang that is meant to be used as an alternative to Distributed Erlang. Partisan introduces two important improvements over Distributed Erlang: (1) the addition of multiple

[4]We will mention them during the review.

runtime-selectable cluster topologies, and (2) the ability to gain additional parallelism by distributing messages over multiple communication channels. Applications that are developed using the Partisan programming model can specify the cluster topology at runtime which allows applications to choose the most efficient topology at hand without having to modify application code.

Partisan was previously introduced in D3.1 at the end of the first year as ongoing work, and that deliverable explains its functionality and its programming model. Since that time, we have continued to work on Partisan through adding new features and providing empirical evaluations summarized as follows.

- the design of the Partisan programming model that supports the runtime specification of multiple cluster topologies;
- the design of the channel-based full mesh backend that enables greater parallelism than possible in Distributed Erlang; and
- a detailed evaluation of Partisan demonstrating increased parallelism through the use of multiple communication channels and increased scalability by specializing the topology to the application at runtime. By leveraging communication channels, we demonstrate up to a 30x improvement on point-to-point messaging, as well as an 13.5x improvement on the distributed database application. By enabling application developers to specify the topology at runtime, we demonstrate the ability to scale the lightweight key-value store application from a cluster of 256 nodes to a cluster of 1024 nodes.

3.5 Towards Partial Replication at the Edge

After presenting some CRDT models and data abstractions that are useful at different edge system models (light, heavy, and hybrid), we now address the data availability and scalability challenge at the edge, by providing a partial replication approach that is needed for applications that exhibit large data sets and desire high availability. The research focuses on providing causal consistency, being a useful consistency model for many modern (edge) applications. In particular, we continued the work started in D3.1 on Saturn service for partial replication, and disussed the need for edge-tailored features. In this section, we mention the progress made in this direction, a new Saturn variant called Gesto. On the other hand, we extend the work on Saturn to allow for concurrent operations using relaxed consistency inside datacenter or microdatacentrs at the heavy edge.

3.5.1 From Saturn to Gesto: towards partial replication at the edge

While the above approach addresses causal delivery as an integrated part of data abstractions, an alternative is to decouple data propagation from causality information that can be maintained through a separate metadata service. Saturn is a metadata service that can be added to any partially replicated data store to provide causal consistency. Saturn was previously introduced in D3.1 at the end of the first year as ongoing work, and details on its architecture and operation are given in that document. Since that time, the Saturn work has matured and led to a Ph.D. thesis by Manuel Bravo, which was defended publicly on July 2, 2018.

Nevertheless, Manuel Bravo's Ph.D. thesis does not address the problems listed in D3.1, falling short as a solution to provide causal consistency for edge networks. Gesto is a novel architecture that aims at tackling these novel challenges, departing for the lessons learnt from Saturn. We summarize the main results in this section[5].

Gesto is a hierarchical architecture that aims at extending—requiring minimal changes—causally consistent cloud storage services with mechanisms to operate in the edge, bringing cloud services closer to clients. Gesto proposes a two-level architecture, where the top level is compose by a handful set of datacenters and the bottom level is composed by possibly hundreds of edge replicas. Each edge replica has a local datacenter to which they are connected, namely their parent datacenter. A datacenter, together with its children edge replicas is called region. By having this two-level hierarchy, Gesto is able to still use the default mechanism of the causally consistent, could storage service that it is extending to operate among datacenters; and its custom, optimise mechanisms to operate within a region.

Gesto relies on constant size timestamps to track causality (the size of these timestamps does not grow with the number of edge replicas). As Saturn demonstrated, this is key to ensure scalability with the number of replicas. The design of Gesto assumes that migration within a region may be relatively frequent and quite rare across regions. Thus, it is optimise to exhibit low intra-region migration. Gesto uses a multipart timestamp and a novel protocol which permits it to significantly lower the intra-region migration latency at the cost of slightly increasing the size of the metadata. Furthermore, Gesto can be reconfigured online with a low overhead. It includes protocols to reconfigure the system in order to (i) change the replication set of a data item, (ii) add (or removing gracefully) an edge replica, (iii) cope with edge replica's failures.

We have built a prototype that implements these techniques. Our prototype, built using the Erlang/OTP programming language, extends a version of Saturn. We evaluate Gesto in Grid5000 using realistic benchmarks. We have compared it to four state-of-the-art solutions that make different trade-offs: COPS[33], Saturn[12] and two variants of Occult[35]. Results show that Gesto is the only solution capable of offering high throughput, fast update replication, scalability, and fast client intra-region migration simultaneously.

3.5.2 Practical Causal Consistency for Geo-replicated Stores

Context and Motivations

Saturn and Gesto both provide a meta-data service [12] that decouples the propagation of metadata, to track causality between operations, from the propagation of update operations between data centers. They introduce an inspiring way to maintain partially replicated systems via the meta-data service. In this section, we present C^3, a solution that builds on the proposed approach by improving the concurrency level. To avoid repetitions, we discuss C^3 as a self-contained solution, explictly mentioning the differences to Saturn's partial replication when it is convenient. We also discuss the directions being pursued for extending our proposals to the edge.

[5]For more information, please see the master thesis of Nuno Afonso available here: http://www.gsd.inesc-id.pt/ ler/reports/nunoafonsomsc.pdf.

C^3 is designed to extend an existing storage system by integrating our replication scheme to enforce causal+ consistency. Thus, our design assumes that the underlying storage system provides the following properties: partial geo-replication with eventual consistency across data centers; within a data center, after a write completes, all following reads must return the written value; data is sharded inside a data center.

Our design separates the system in two layers: the causality layer and the datastore layer. Clients interact with the datastore layer for executing client operations. The datastore layer is responsible for executing operations in the local data center, and propagating them to remote data centers. The datastore layer coordinates with the causality layer to decide when an operation can be executed in order to enforce causal consistency. The causality layer is responsible for propagating causality tracking information among operations. In our system, this information consists of *labels* – for each write operation, our system generates a *label* consisting of a unique identifier and the causal dependencies for the operation. The causal dependencies are a vector with one entry per data center.

Each data center contains a causality layer instance which is responsible for managing the causality information inside a data center and exchange causality-related information – labels – with other causality layer instances. Each causality layer instance communicates directly with every other causality layer instance. This approach differs from Saturn, where causality information was propagated through a single tree interconnecting all causality layers instances.

Our system design handles three types of operations: *(i) read* operations, to read the state of the database; *(ii) write* operations, to modify the state of the database; and *(iii) migrate* operation, to change the home data center of a client.

Enforcing causality The goal of the causality layer is to maintain the necessary information to guarantee that any operation execution respects causal consistency. As we assume that in general there are more read operations than write operations, our design favors the execution of read operations. Thus, we want to allow read operations to proceed without any coordination, executing directly in the datastore.

Under our system model, common to systems that use quorum-based replication, while a write is in progress, the value returned by a read might be either the old or the new value. For enforcing causal consistency we need to guarantee that: *(i)* for a write, $w(o)$, after a client returns a version that reflect $w(o)$ it cannot later read an older version that still not reflects the write; *(ii)* for two writes of the same or different objects, $w_1(o_1)$ and $w_2(o_2)$, where w_1 causally precedes w_2, if a client reads a version of object o_2 that reflects w_2, all following reads of o_1 must return a version that reflects w_1.

The first property could be enforced immediately by a system providing one-copy serializability (or linearizability). However, our system model assumes a weaker model, common in quorum-based replicated systems. This property can be enforced in the client layer by caching the values returned by read operations. For systems that use quorum-based replication, the same property can be enforced by always contacting the same quorum of nodes when reading a given object.

For enforcing the second property, we adopt the following strategy: first, we record information about the dependencies of each operation; second, we guarantee that operations that may depend on each other cannot execute concurrently.

We rely on the causality layer to implement this strategy. To this end, the causality layer maintains: an *operation counter*, used to timestamp operations; an *executed clock*, a vector with one entry per data center recording the timestamp of the latest operation executed from that data center; an *executing clock*, a vector recording the timestamp of the latest operation from each data center in execution in the local data center.

When the application issues a write, the client sends the operation to the datastore, which forwards the information to the causality layer and propagates the operation itself to the nodes that are responsible for executing the write. However, the operation is not immediately executed in any node.

When the causality layer receives the information about a new local operation, including a unique identifier and the set of data centers where the operation is to be delivered to, it increments the local operation counter, uses it to assign a timestamp to the operation, and sets the operation dependencies to be those of the executing clock – this information is the operation *label*. We note that this is necessary in our system model, as for operations that are being currently executed, it is not possible to determine if the client has read the old or new value of the objects. Thus, the only safe approach is to assume that the write depends on all writes that are in progress.

The causality layer puts the label of the new operation in a log of pending operations to execute and propagates it to the causality layers in relevant data centers (i.e., data centers that replicate the modified object). When a causality layer receives a label from a remote data center, it adds it to the log of pending operations. A pending operation is ready to execute when the operations it depends upon have already completed, i.e., all entries of the *executed clock* are larger or equal to the entries in the dependencies of the operation.

When an operation is ready to execute, the causality layer notifies the local nodes responsible for executing the operation that they can execute the operation; these nodes acknowledge the causality layer when the execution completes. In conjunction with our mechanism to record the causal dependencies of an operation, this approach guarantees that an operation only executes after all operations it depends upon have completed.

Migrate When reading or writing an object that is not replicated in the local data center it is necessary to guarantee that the system still enforces causal consistency. Intuitively, what is necessary is to guarantee that the operation of accessing the remote replica executes after all operations that have been observed by the client. As the client does not record any information about the operations it must depend on, this information needs to be obtained from the causality layer. Again, the only safe approach is to assume that the client might have observed a version of the database that reflects all operations under execution.

As in Saturn, we implement a migrate operation that allows a client to move to a different home data center. This operation executes similarly to a write operation, with the difference that it is only propagated to the target (new home) data center, and that when the operation is ready to execute in the target data center, the client is notified that the migration has completed. After the migration completes, a client can send its operations directly to the new home data center.

Discussion Our approach tracks a safe approximation of causal dependencies, while still allowing a high degree of concurrency as we discuss in this section.

When compared with solutions that track causal dependencies precisely, such as COPS [33], our approach has the advantage of requiring no additional information to be stored in the storage system. To track dependencies precisely, these systems require, at least, a version identifier to be recorded with every object version.

When compared with systems that also include a causality layer to control the execution of operations, such as Saturn [12] and Eunomia [26], our system relies on a weaker consistency model inside a data center, allows increased concurrency, requires no additional information to be stored with each object and does not require the client to manage any information about dependencies.

Both Saturn and Eunomia assume that each storage system implements linearizability inside a data center. Although linearizability can be implemented efficiently, current data storage systems that use quorum-based replication often adopt a weaker model – e.g. Cassandra and Riak with any quorum specification and MongoDB with majority read concern[6].

Additionally, our approach allows increased concurrency when compared with Saturn. In this system, as writes executed concurrently in a data center by multiple clients are serialized, all information about the fact that they are concurrent is lost. Thus, when executing these operations in remote data centers, they need to execute serially, i.e., an operation from a data center can only start executing after the previous operation from the same data center completes. On the contrary, in our system, as writes includes their dependencies, they can execute concurrently. We note that the trade-off is that, in our system, a write can only execute in a data center after the completion of the writes that are being executed when the write starts. However, if multiple writes are issued concurrently, they all have to wait for the completion of the writes being executed, but then they can all execute concurrently.

In deliverable D6.1 we discuss how we have integrated our proposal in an existing storage system and present some performance results. More information about this work is presented in the publication under submission list in the end of this report.

Providing Causal Consistency at the Edge For providing causal consistency in partially replicated settings, Saturn and C^3 decouple the propagation of causality information from the propagation of updates. For the proposed approaches to work, a key property is to be able to know that no operation that an operation depends on is missing. In general, this is challenging in partially replicated settings, as operations are only propagated to nodes that replicate the information they modify. In Saturn, this problem is solved by using a tree to disseminate information about operations – by the properties of dissemination in the tree, it is known that when the information for an operation reach a node, all dependent operations also had reached the node. In C^3 (and in Cure [2]), servers collect information that no other operation is missing with a given origin.

We are currently exploring combining the C^3 in the core of the network, with a Saturn-like style of propagating causality information for nodes located at the edge of the network, including mobile and sensor nodes. We expect to report a complete protocol in the next deliverable.

[6]MongoDB supports linearizability, but it is more expensive than majority.

3.6 List of Software and Prototypes

In this section, we present the software deliverables, libraries, and components in which the contributions presented in this report appear. Most of these software have been started in D3.1 and are direct artefacts that show in the LightKone Reference Architecture (LiRA), or used as backend components and libraries. Since this work package is meant to provide the support to build generic edge computing runtimes, we believe that developing fine-grained components is crucial to increase the impact of LightKone's work on external edge computing platforms. Indeed, although LiRA perfectly fits the set of LightKone use-cases, the latter represents a sample edge computing set of applications, and thus addressing more use-cases may require building other edge computing runtimes. To this end, the components provided in this deliverable can be used in building new edge runtimes or integrated in existing ones to leverage the technology LightKone provides.

- AntidoteDB: is a geo-replicated CRDT data store offering transactional causal consistency and fits nicely the heavy edge. It is publicly available under Apache 2.0 license. Available at https://github.com/SyncFree/antidote.
- Lasp: a framework that allows to design and execute scalable synchronization-free applications that resort to CRDTs as their data model. Available at https://github.com/lasp-lang/lasp.
- Antidote CRDT library: an Erlang library of operation-based CRDTs comprising counters, flags, maps, sets, integer, registers and sequence (RGA). The library can be used in any system, and is currently used in Antidote. Available at https://github.com/SyncFree/antidote_crdt.
- Yggdrasil: a framework and and runtime to build wireless edge applications. The library includes a set of aggregation protocols a prototype of a control protocol that simplifies the use of Yggdrasil as a benchmark platform. Available in https://github.com/LightKone/Yggdrasil-devel.
- Partisan: a TCP-based membership system written in Erlang/Elixir and implements the HyParView[31], hybrid partial view membership protocol, with TCP-based failure detection. Partisan is suitable for dynamic topologies being robust and lightweight. Available at https://github.com/lasp-lang/partisan.
- Efficient CRDTs library: a library of state-based, delta-based, delta-composition, and pure-op-based CRDTs written in Erlang. The library includes implementations of grow-only counter, positive-negative counter, lexicographic counter, bounded-counter, grow-only set, two-phase set, add-wins and remove-wins set, enable-wins and disable-wins flag, last-writer-wins and multi-value registers, and an add-wins map. Available at https://github.com/lasp-lang/types.
- TCSB middleware: a vector-based middleware that implements the TCSB (Tagged Causal Stable Broadcast) protocol, written in Erlang. The implementation supports efficient causal delivery and causal stability. Available at https://github.com/gyounes/trcb_base.

3.7 Advancing state of the art

All the works appeared in thesis report are continuation of those started in D3.1. Therefore, the state of the art (SOTA) review in D3.1 also holds here. Consequently, we avoid repetition (and refer the reader to report D3.1), and we thus only summarize the delta increment over D3.1. For the convenience of the reader, we summarize the main contributions during these six months in Table 3.7.1. Considering the overall plan presented in Chapter 3, we estimate the current progress regarding the final milestone to be between 50% and 60%.

Table 3.7.1: Summary of LightKone WP3 Contribution for the first six months of Year 2.

Component	**Description**	**Previous SOTA**	**Contribution**	**Software**	**Reference**
State CRDT	State-based data management for relaxed consistency at the edge	Efficient state synchronization using join-decomposition	state synchronization empirical evaluation	Legion, Lasp	Cha 4. Sec. 3.3
Op CRDT	Operation-based data management for relaxed consistency at the edge	Generic framework; optimized datatypes; support resets; many datatypes	Compression at the source	None.	Cha 4. Sec. 3.3
C3 and Gesto	Improved Partial replication (sharding) meta-data handling with causality support	Saturn	Improved concurrency on the server or heavy edge cluster, and edge-tailored low intra-region migration latency	None	Cha 4. Sec. 3.5
Causal Delivery and stability middleware	middleware for causal consistent systems and op-based CRDTs	reduced meta-data in causal delivery; causal stability concept;causality issues in callback-based and independent threads/processes.	End-to-End Causal Delivery; Correct tagging; Implementing Causal Delivery and Stability	TCSB	Cha 4. Sec. 3.4
Distributed Communication	Edge-tailored alternatives distribution layer for Erlang.	Partisan: Hybrid gossip-based with different net topologies and various clusters.	Partisan channel-based full mesh back-end; evaluation (# of nodes scalability)	Lasp, LaspOnGrisp	Cha 4. Sec. 3.4

We continued the work on improving the efficiency of CRDT models. We extended the Pure operations based CRDTs with a technique to do deduplication before sending the updates. This improves on SOTA pure op-based [7] by avoiding the overhead of sending redundant meta data, which is desired at the edge. The work is prepared as an extension to journal publication to be submitted soon. On the other hand, the work started on efficient delta CRDTs transfer using join decomposition has been continued [22]. In particular, we introduced sufficient conditions for the existence these decompositions, showing how they can be obtained, and presented an evaluation of the optimizations proposed. Our evaluation showed a substantial improvement in transmission overhead for delta CRDTs over classical state-based and other benefits regarding memory and computation. In addition, we have started exploring the potential of model-agnostic CRDTs. Our preliminary work shows that it is possible to compose CRDTs from few simple abstractions regarding of the model. This may help future work on interoperability of state and op-based CRDTs, a possible scenario when different edge nodes share data of different model. Finally, we have published more works to improve the understanding of CRDTs.

At the communication side, we conducted a study to understand the pitfalls of implementing causal consistency and their impact on concurrent applications. This is inline with the work started on developing efficient causal middleware. In particular, we have developed a novel technique to implement meta data logs of such middlewares. The work based on tree-structure tagging helps greatly pruning a partial order log upon delivery. We are planning to continue this progress in the future. We have also made progress on Partisan communication library to support multiple clusters and make use of parallel channels in Distributed Erlang. The evaluation demonstrated increased parallelism through the use of multiple communication channels and increased scalability by specializing the topology to the application at runtime. By leveraging communication channels, we demonstrated up to a 30x improvement on point-to-point messaging, as well as an 13.5x improvement on the distributed database application. By enabling appli- cation developers to specify the topology at runtime, we demonstrate the ability to scale the lightweight key-value store application from a cluster of 256 nodes to a cluster of 1024 nodes.

We have also made progress on partial replication. We have supported new features to Saturn meta-data service and developed a variant called Gesto that is optimized for the edge. Gesto optimize to exhibit low intra-region migration. It uses a multipart timestamp and a novel protocol which permits it to significantly lower the intra-region migration latency at the cost of slightly increasing the size of the metadata. Furthermore, Gesto can be reconfigured online with a low overhead. It includes protocols to reconfigure the system in order to (i) change the replication set of a data item, (ii) add (or removing gracefully) an edge replica, (iii) cope with edge replica's failures.

C3 is also another advancement over Saturn. It focuses on exploiting parallelism inside a data center or heavy edge cluster. It is designed to extend an existing storage system by integrating our replication scheme to enforce causal+ consistency. When compared with solutions that track causal dependencies precisely, such as COPS [33], our approach has the advantage of requiring no additional information to be stored in the storage system. To track dependencies precisely, these systems require, at least, a version identifier to be recorded with every object version. When compared with systems that also include a causality layer to control the execution of operations, such as Saturn [12] and Eunomia [26], our system relies on a weaker consistency model inside a data center, allows increased concurrency, requires no additional information to be stored

Chapter 4

Threat Analysis for LightKone Use-cases

In deliverable D3.1, we started to analyze the various use-cases (presented in deliverable D2.1) from a security perspective. In this section, we give a more detailed threat analysis given the use-cases formal specifications in D2.2.

The security analysis formalization consists in a high level threat model focused in both, the system assets and software. The composite approach is justified by the unavailability of detailed software specification in most use cases due to either their proprietary nature or development stage. For both scenarios, a comprehensive threat model provides early understanding of security requirements through an abstract representation rather than the code itself, which allows addressing security along the system design and component selection.

The threat modeling adopted in this report is composed of three steps:

System decomposition provides a clear understanding about its entities, applications and interconnections, which reveals their trust boundaries and attack surfaces. For each use case, the decomposition stage has provided a Data Flow Diagram (DFD) that supports the threat identification stage;

Threat identification consists in identifying and categorizing threats for every element of the system described in the previous stage. The methodology used classifies threats the systems are exposed to in six classes: (i) Spoofing; (ii) Data tampering; (iii) Repudiation; (iv) Information disclosure; (v) Denial-of-Service - DoS; and (vi) Elevation of privilege;

Countermeasures and mitigation for threats in elements with sufficient information, some approaches on how to address them either through off-the-shelf solutions or academic research approaches are provided.

Along the next sections we provide a brief description of the use cases addressed in LightKone's scope followed by a system decomposition and the number of threats to which their entities are exposed to. A detailed description encompassing all identified threats, actors and strategies to cope with them are further presented in Appendix A.

4.1 UPC - Coordination between servers and data storage for the Guifi.net monitoring system

The system described in Sections 1 and 2 of Chapter 3 of deliverable D2.1 is composed of three main components. First, there are approximately 34,000 active nodes, such as routers and switches, that collaboratively provide or consume diverse network services (e.g. Internet connectivity). Their state and available resources need to be monitored for the purpose of billing, capacity planning and service provision. This is done by the second component of the system - the monitoring servers which consist in instances of the SNPServices tool (approximately 200 active servers). They coordinate with each other in order to distribute the workload of monitoring tasks and maintain a shared database containing the results of monitoring. Finally, the main Guifi.net website and the central database server aggregate and display data provided by the monitoring servers. The database also maintains the list of nodes to be monitored which it shares with the monitoring servers.

Figure 4.1.1 presents a Data Flow Diagram (DFD) identifying all the elements into the monitoring system, their trust boundaries and interconnections while Table 4.1.1 summarizes the classes of threats to which each element in the monitoring system is exposed to.

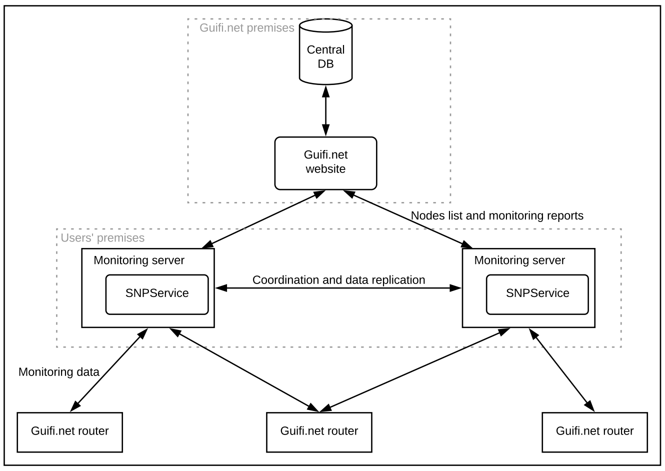

Figure 4.1.1: DFD - Guifi.net monitoring system

Threat	Centrad DB	Guifi.net	Monitoring server	Network node
Spoofing	-	2	1	1
Data tampering	-	-	4	2
Repudiation	-	-	1	2
Information Disclosure	-	3	1	1
Denial-of-Service (DoS)	-	1	2	-
Elevation of Privilege	-	-	1	1

Table 4.1.1: Number of threats identified per element.

General security requirements

This section presents a set of general security requirements derived from the security analysis of UPC's monitoring system.

- The system must ensure that each network node is monitored by at least three independent monitoring servers for cross-checking. If any persistent divergence in the reported data is detected the corresponding server must be flagged and reported;
- The communication channels among entities belonging to the monitoring system or its related messages must be encrypted;
- The system must protect the integrity of the monitoring database. All operations must be logged and authenticated. The monitoring servers can only write or modify the database entries for the nodes they were assigned to. Any modification of monitoring data for other nodes must be forbidden;
- The status data reported by the network node must be checked for correctness, integrity and authenticity before being added to the database. The monitoring software running on the node must be protected and verified for each report;
- The monitoring tasks distribution must not only be fair and efficient but also abuse-resistant. No monitoring server must be responsible for the majority of nodes;
- The system must be resistant to network partitions and database corruption. The collected data should be persistent and remain available when any of the monitoring servers fails or disconnects.
  These aspects can be directly assessed through core developments into this project, particularly, the hybrid gossip communication protocol in WP5 will provide resiliency to network partitions and node faults, while the efforts in WP6 together with AntidoteDB will enhance the Central DB corruption resistance.

4.1.1 Scality - Pre-indexing at the edge

The system described in Section 1 of Chapter 4 of deliverable D2.1 consists in Zenko Multi-Cloud Controller, an open-source project that provides a unifying storage interface and advanced search capabilities (by indexing the data) in multi-cloud backend data storage systems, including Amazon S3, Microsoft Azure and Google Cloud Platform.

Figure 4.1.2 presents the main components in this system, which comprises Client-applications that write data to the storage system and retrieve the search results; precomputing nodes at the edge (i.e., Zenko) that aggregate client requests and forward data to higher levels of the system; and backend storage systems where the client data are stored (i.e. Clouds). The precomputing nodes also perform various computations on client data including encryption, hash signature generation, indexing and index lookups. Moreover, some customers have legacy applications (i.e., external applications), that directly communicate with the cloud systems and are not yet connected to Zenko. Table 4.1.2 presents the number of threats identified for each entity in this system according to the classification previously discussed.

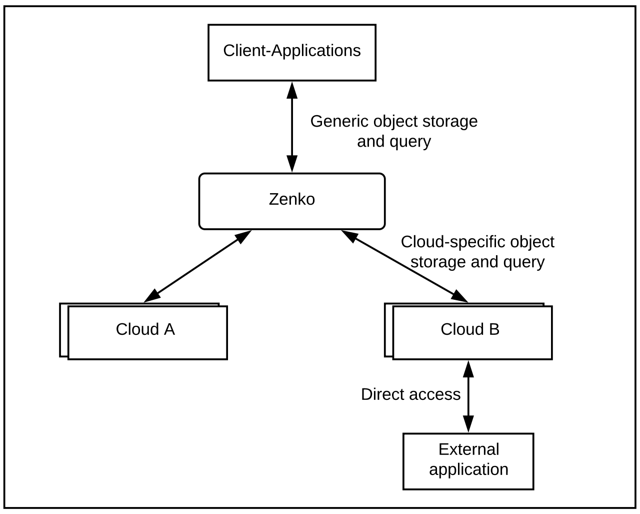
Client-Applications
Generic object storage and query
Zenko
Cloud-specific object storage and query
Cloud A
Cloud B
Direct access
External application

Figure 4.1.2: DFD - Pre-indexing at the edge

Threat	Client-Applications	Zenko	Cloud	External application
Spoofing	2	1	-	1
Data tampering	2	3	1	3
Repudiation	1	1	-	1
Information Disclosure	2	4	2	2
Denial-of-Service (DoS)	-	1	-	-
Elevation of Privilege	1	1	-	1

Table 4.1.2: Number of threats identified per element.

General security requirements

- The system must provide defense mechanisms preventing access to unencrypted data flows;
- All the parties involved in the system must be authenticated and authorized by the user;
- The communication channels among entities belonging to the system or their related messages must be encrypted;
- The system must provide the ways for the user to verify the integrity of data at any stage of processing.

An important functionality to be provided by Zenko solution is the ability to perform data search in data previously encrypted at the edge during the index computation and stored in cloud services (see D2.2). In this sense, the work on searchable encryption (i.e., BISEN) being developed under Lightkone's scope has the potential to provide such feature. BISEN is detailed in Section 5.3.

S3 model and assumptions

- The S3 REST (for Amazon cloud services) access model consists of every individual query having a verifiable signature based on a shared secret, which implies some differences from the others supported cloud services:
  - The secret must be in clear text on both ends, on the client and the server. It can be encrypted with symmetrical keys, even so, the client still must have the secret to transact. In this way, an attacker discovering such secret is an important threat;
  - Key/secret combinations can be time-based and are easily revoked, which can limit threats;
  - Very granular rights can allow access without allowing the modification of rights;
  - Every request is verified, so that stolen tokens are less useful;
  - Man-in-the-middle attacks are very difficult since signatures are based on numerous elements of a query.
- The storage of the secrets is a key point. The basic open source version has a simple flat file for storage, so more sophisticated models are important for sensitive data;
- The public facing S3 endpoint interfaces represent a fairly limited attack surface, even if a secret is stolen, only that specific users' data would be compromised;
- The Identity and Access Management (IAM) API represents a more significant threat as a stolen secret could permit tampering with rights for an entire enterprise;
- The complexity of the IAM security model is often recognized as a significant source of errors, misconfigurations, and breaches as a consequence.
- In a private environment, a malicious or irresponsible administration can cause breaches;
- In public clouds this attack vector exists but is difficult to measure, since these environments are generally not forthcoming with there internal policies;

4.1.2 Stritzinger - No-Stop RFID

We now focus on the use case presented in Section 1 of Chapter 5 of deliverable D2.1 and described in Figure 4.1.3. This system is composed of the following components: the RFID tags that are moving on the conveyor belt, the RFID readers that can read and write the data to and from the RFID tags, and a distributed cache of RFID content featuring completed and missing steps. The readers communicate with each other through Ethernet network by flooding all the latest updates to keep the cache data consistent.

Despite the protected environment in which the No-Stop RFID system is deployed, its elements face threats that could compromise the manufacturing process. The classes of threats to each element is exposed to are presented in Table 4.1.3 and further discussed, in Appendix A.

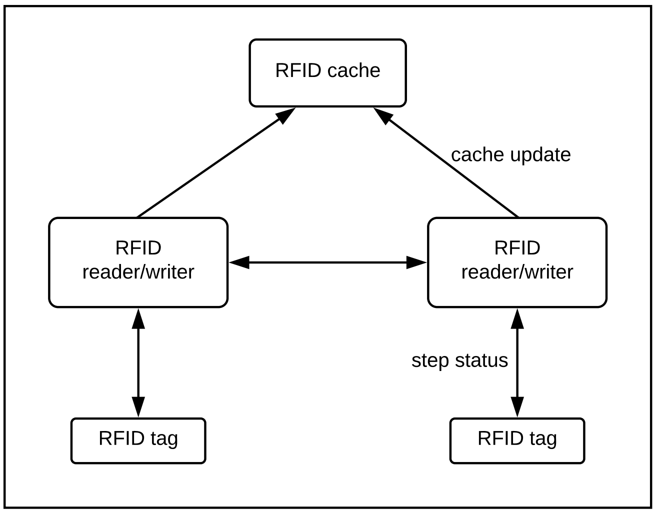

Figure 4.1.3: DFD - No-Stop RFID

Threat	RFID tag	RFID reader/writer	RFID cache
Spoofing	1	1	1
Data tampering	1	2	2
Repudiation	1	1	1
Information Disclosure	1	1	1
Denial-of-Service (DoS)	-	1	1
Elevation of Privilege	-	-	-

Table 4.1.3: Number of threats identified per element.

General security requirements

- The system must be able to protect the cache data from eavesdropping or manipulating by unauthorized parties;

- The communication between the readers must be secured and resistant to DoS attacks;

- The system must provide a way for the factory owners to verify the authenticity of RFID tags and reader software and hardware;

- Any modifications to the cache data must be detected and flagged without affecting the manufacturing process.

The main security requirements yielded by the No-Stop RFID threat model will benefit from the work in WP5 (i.e., M19-M36) where we will extend the suit of distributed protocols and mechanisms designed, implemented, and evaluated in the previous tasks with security features, namely, data privacy, data integrity and strategies to tolerate DoS attacks.

4.1.3 Gluk - Self-sufficient precision agriculture management for irrigation

The use case scenario described in Chapter 6 of deliverable D2.1 presents a sensor-based platform for precision agriculture which allows for collecting, analyzing and reacting to field sensor data in near real-time. It collects the readings from thousands of field sensor nodes, aggregates these values at the edge gateway and matches the processed values with the most suitable irrigation rule for the field state. Then, such irrigation pattern is applied to the crops through actuators. Figure 4.1.4 shows all the main entities and relations into this use case and Table 4.1.4 summarizes the classes of threats identified for every element.

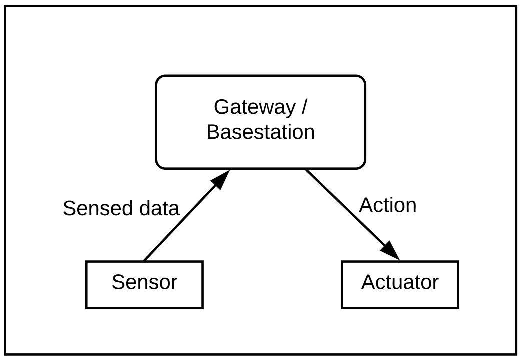

Figure 4.1.4: DFD - Agriculture sensing analytics

Table 4.1.4: Number of threats identified per element.

Threat	Gateway	Sensor	Actuator
Spoofing	1	1	1
Data tampering	2	2	1
Repudiation	-	1	1
Information Disclosure	2	1	1
Denial-of-Service (DoS)	1	2	2
Elevation of Privilege	3	-	1

General security requirements

- Every node in the network must be identified and checked for any software or hardware modifications before being added to the network. All nodes failing this check must remain isolated from the rest of the network;
- The gateways must be able to identify the sensor nodes abusing the wireless and battery resources and have ways to isolate those from the rest of the network;
- The system must prevent any changes to the sensor readings and the actuators commands at the time they are generated, transmitted and processed. Any manipulation with the data along the way must be detected and reported;

- The system must remain available even in the presence of connection failure of several days or more;
- The data generated by the system must be only accessible to the authorized entities (e.g. network nodes, applications, system administrators). The system must ensure secure communication channels and storage;
- The system must be robust and fault tolerant. It must adapt to changes in the wireless environment and withstand the DoS attacks. Each communication channel must be replicated to ensure data availability and timely delivery;
- The sensor data must be verified for correctness before the action is taken upon it. The system must detect the deviations in the sensor reading reported by the sensors that are close to each other, cross-check the values and discard suspicious readings. Any repeated, conflicting or inconsistent actuators commands must be detected and discarded as well.

Chapter 5

Exploratory Research

In this chapter, we present the research exploratory work that is related but not at the core of LightKone. These works have the potential of more exploration or inspiration in the future. Specifically, we present more details on the model-agnostic CRDT approach mentioned in Section 3.2, and two security works on Efficient Boolean Searchable Symmetric Encryption and Securing Smart Hubs through N-Version Programming.

5.1 A model-agnostic CRDT definition language

In a radically different approach, we are currently working on an model-agnostic CRDT definition language which allows the understanding and construction of CRDTs as type abstractions regardless of a particular implementation model, e.g., the op-based and state-based variants in the previous sections. Indeed, CRDTs in these models are closely linked to the type of communication layer used for their implementation. For example, state-based CRDTs are defined in such a way that arbitrary and ad-hoc communication between actors is made possible, while operation-based CRDTs assume exactly-once causal delivery of messages between actors. This leads to CRDTs which have the same semantics being defined in multiple ways.

In work in progress, we show that CRDTs can be defined in a communication-agnostic way using a limited number of primitives. The primitives are chosen in such a way that they abstract over common patterns in the semantics of CRDTs, and we aim to choose only primitives that can be compiled to efficient implementations. That is, the definitions should be mechanically translatable to an optimal implementation in an existing model. We identified four common patterns and defined four primitives which we will use to describe nine common CRDTs.

(a) Base assumptions

We assume that a CRDT is state shared amongst an arbitrary number of actors, and that this state can be modified by *operations*. Actors gain knowledge of operations performed by other actors, and derive from these known operations their local *value* of the state. In an actual implementation of a CRDT knowledge of individual operations might only be implicit: where possible a good implementation will summarize operations to conserve space and bandwidth.

For defining CRDTs we only describe the semantics and list the operations performable on the CRDT, as well as how the value derives from the known operations. An actual implementation does not always require individually distinguishable operations, but we assume in our descriptions that operations are uniquely identifiable.

(b) Patterns and primitives

We define the following primitives: *aggregation* of commutative and associative operations, *reinterpretation* of an existing CRDT's operations and value to define a new one, causing the *forgetting* the operations in the causal past of an actor by means of a *clear* operation, and finally *mapping* of values to other CRDTs. While we do not provide a formal semantics for these primitives, we do annotate the language with types. Each term has a type (O,V), where O denotes the set of operations of the CRDT, and V denotes the set of values the CRDT can assume.

Aggregation is the most basic primitive, defined on a set of values E, and a commutative and associative binary operation $\oplus$. This is known in mathematics as a *commutative semi-group*. We write it as follows:

$$\textsc{Agg}\langle E, \oplus\rangle \;:\; (E,\ \{\bot\} \cup E)$$

The set of "operations" are the same as the values in E, and the value of the aggregation is obtained by combining all known operations into one value using o. If no operations are known, the value of the CRDT is $\bot$. Aggregation is never used as is, but is instead used in combination with *reinterpretation* to provide user-friendly operations and a useful default value instead of $\bot$.

Reinterpretation uses an existing CRDT C to produce a new CRDT C', where the operations of C' are mapped to one or more operations of C using a function o, and the value of C is mapped to the values of C' using a function v.

$$\textsc{Interpret}\langle C : (O', V'),\ o : O \to O',\ v : V' \to V\rangle \;:\; (O,V)$$

Adding a *clear* operation to an existing CRDT, which causes the actors which gain knowledge of it to forget the operations which were known by the actor who performed the operation, at the time it was performed. CRDTs which have this operation implicitly *clear* when an operation is performed.

$$\textsc{Clear}\langle C : (O,V)\rangle \;:\; (\{clear\} \cup O, V)$$

Mapping from keys to CRDT instances of a particular type partitions operations of that type per key. The single operation on a map is *apply(key,operation)*, and the value of the map are key-value pairs, where the values are those of the type of the embedded CRDT and are derived from all known operations per-key.

$$\textsc{Map}\langle K, C : (O,V)\rangle \;:\; (\{apply(k,o) : k \in K,\ o \in O\}, \{(k,v) : k \in K, v \in V\})$$

Name	E	$\oplus$	Operation	Default
PNCOUNTER	$\mathbb{I}$	$+$	$inc \to 1$ $dec \to -1$	0
GSET⟨V⟩	$\mathcal{P}(V)$	$\cup$	$add(v) \to \{v\}$	$\emptyset$
EOFLAG	$\mathbb{B}$	$\vee$	$enable \to true$ $disable \to false$	$false$
EDOFLAG	$\mathbb{B}$	$\wedge$	$enable \to true$ $disable \to false$	$false$

Table 5.1.1: The model-agnostic definitions of four CRDTs.

(c) Aggregate & Interpret

CRDTs which can be reduced to a set of values and a commutative and associative operation on them can be constructed using the AGG and INTERPRET primitives. Where INTERPRET is used to give the CRDT user-friendly operations and a default value other than $\perp$. For example, the positive-negative counter (PNCOUNTER) can be defined as follows on the integral numbers and the addition operation:

$$\text{PNCOUNTER} : (\{inc, dec\}, \mathbb{I}) = \text{INTERPRET}\langle \text{AGG}\langle \mathbb{I}, +\rangle, o, v\rangle$$

where o is a function mapping $inc \to 1$ and $dec \to -1$, and v is a function mapping $\perp$ to 0, or preserving the value of the aggregation otherwise.

That is, a PNCOUNTER is a CRDT with *increment* and *decrement* operations, and its value is the total of all known increments and decrements, 0 if none are known.

In Table 5.1.1, we summarize the implementation of four CRDTs by listing the values and operation of a semi-group, providing a translation of operations to the values, and finally a default value for when there are no known operations. The grow-only set (SET) of a set of values V is a CRDT with only an *add(v)* operation, and the value of the set are all the known added values. We can define it using the semi-group consisting of the powerset of V and the set union operation. The *add(v)* operation is translated to the singleton set.

Finally, we define two flag data types which have not been described before, but will be useful for defining other CRDTs. These are the enable-once flag (EOFLAG) and the enable-disable-once flag (EDOFLAG). The enable-once flag reads *false* when no or only *disable* operations are known, but once an *enable* is known the flag will always read *true* from that point on. This can be reduced to the logical *or* operation on booleans, by translating enable to true and disable to false.

Similarly, the enable-disable-once flag starts out *false*. When only *enable* operations are known it reads *true*, but once a *disable* is known it will read *false* from that point on. Like the enable-once flag this can be achieved by translating the flag operations to boolean values, but to achieve the semantics our combining operation must now be logical *and*.

Base	$\textsc{Clear}\langle Base \rangle$	o	v
GSET	Multi-Value Register	$set(v) \to \{v\}$ $clear \to clear$	—
EOFLAG	Enable-Wins Flag	—	—
EDOFLAG	Disable-Wins Flag	—	—

Table 5.1.2: The model-agnostic definitions of three CRDTs with CLEAR.

(d) CRDTs with a clear operation

By applying the CLEAR primitive to three of the four previously defined data types we can define three more CRDTs, in Table 5.1.2, which do *conflict resolution* on concurrent operations. These are the multi-value register, the enable-wins flag, and the disable-wins flag. Informally the conflict resolution process for CRDTs which *clear* can be described from the point of view as an actor communicating to another as "I acknowledge all the previous operations I have received, but I want my operation to win, unless you hear about operations I did not know about; then combine those operations with mine."

The multi-value register is a CRDT with a single $set(v)$ operation which aims to change the value of the register and with a value which is the set of concurrent register changes. This is analogous to the grow-only set, whereby set operations are translated to singleton sets, and also cause other actors to disregard the operations known by the issuing actor. The enable-wins and disable-wins flags resolve concurrent conflicts by letting either *enable* and *disable* operations win. Like the multi-value register which can be derived from the grow-only set, the two flags can be derived from the enable-once flag and enable-disable-once flag respectively, simply by implicitly causing operations on the flags to disregard the operations known by the issuer when becoming known by the recipient.

(e) Add-remove sets with conflict resolution

CRDT sets which aim to support a *remove* operation as well as an *add* need to resolve a conflict which occurs when an element is concurrently added or removed. The two usual variants are either *add-wins* or *remove-wins* conflict resolution.

An add-wins set can be seen as participants voting on the presence or absence of an element in the set, with concurrent "presence" votes winning, while in the case of remove-wins set the "absence" votes win. We can encode this intuition as a map from set values to a flag CRDT, the latter denoting the votes of presence about the keys in the former:

$$\textsc{Set}\langle V, F : (\{enable, disable\}, \mathbb{B})\rangle : (\{add(v), remove(v) \mid v \in V\}, V) = \textsc{Wrap}\langle \textsc{Map}\langle V, F\rangle, o, v\rangle$$

where the function o maps additions to the set to *apply(v,enable)* and removals to *apply(v,disable)*, and the function v returns the set of keys of the wrapped map for which the value is *true*. By using the EWFLAG or DWFLAG we can define both types of conflict resolution using existing CRDTs:

Flag	MAP$\langle V, Flag \rangle$
ENABLE-WINS FLAG	ADD-WINS SET
DISABLE-WINS FLAG	DISABLE-WINS SET

5.2 Securing Smart Hubs through N-Version Programming

The privacy threats of IoT applications processing sensitive IoT data are often overlooked. The applications run at the remote cloud of a service provider and tend to request more access permissions than they actually need to perform a given task. This fact, if left unnoticed, may cause a severe damage to users' privacy.

In order to address these privacy problems, several recent proposals suggested to retain all sensitive IoT data at the edge under the control of the homeowner [21, 23]. Rather than depending on the cloud for all processing and storage needs, there is a locally deployed *edge hub* that collects and aggregates data from sensor devices and provides a platform for the execution of third-party IoT applications installed by the users. This hub provides an API functions, so-called *Trusted Functions* (TF), for the applications to access and process the sensitive IoT data according to the privacy policy of the user.

Trusted functions aim to implement high-level operations that mediate access between the application and the raw IoT data. In some cases a TF interposes between the application and a *data source*, e.g., a camera device. The motivation for such a TF can be, for instance, to provide a face recognition service over raw image data collected from the camera without revealing the raw data to client home apps. TFs can also mediate access to *data sinks*, for example to encrypt or anonymize sensitive data before sending it to a remote server. Some home hub solutions support TFs at data sources [21], others at data sinks [36], and others in both [23]. Once installed into the hub, trusted functions can be invoked by local IoT apps running on the hub. TFs must be developed by third-parties and installed by the hub administrator.

However, by definition, *trusted functions must necessarily be safe to execute*. As also referred by Davies et al. [21], this requirement constitutes a significant drawback since if a trusted function is malicious, it can easily compromise the security of the home hub platform. Having a direct access to the sensitive IoT data, such function can cause a privacy breach instead of preventing it. In this work, we aim to address this limitation by presenting a solution to the design of an edge hub platform that can retain the original flexibility and versatility of trusted functions without requiring full and exclusive trust on the trusted function developers to ensure its correct behavior.

To overcome this challenge, we proposed a new solution based on N-version programming (NVP) [27]. The key insight of our solution is that, rather than depending on a single unreliable trusted function implementation, the idea is to depend on multiple implementations (versions) of this function that run side by side and must agree on a common result. Assuming that these versions were developed independently and cannot collude, such approach ensures that a single malicious version cannot force an incorrect outcome to be sent to the application thereby enhancing trust.

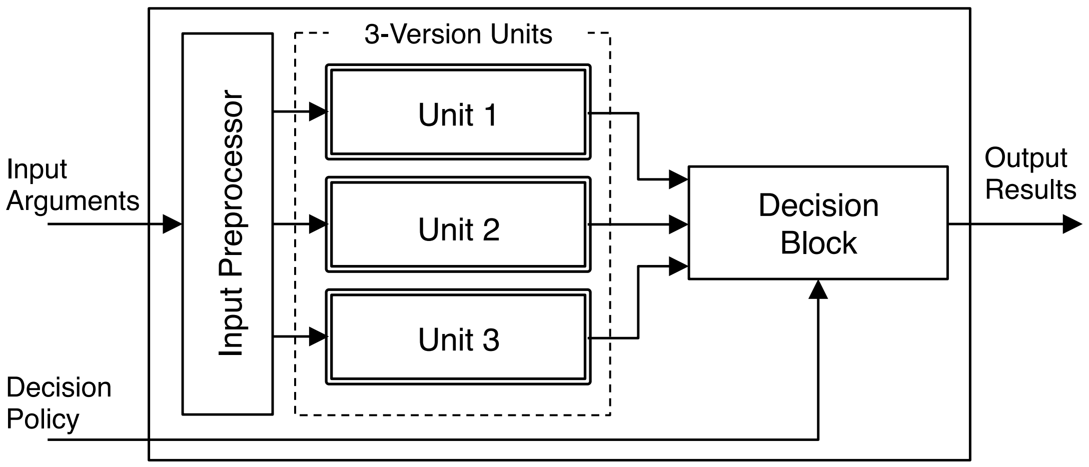

Figure 5.2.1: N-version trusted function module (with N=3).

5.2.1 Edge Hub Architecture based on N-Version Trusted Functions

In this section, we present a general security architecture for edge hubs based on N-version programming. In this architecture, edge hub extensions consist of *N-version trusted function modules* (henceforth called "modules"). A module provides the functionality of a single TF implemented internally in a N-version fashion, with each of the N versions being provided by independent developers. Each of these versions, called *units*, are required to implement the same *trusted function specification*.

Whenever an application issues a request, the input parameters are forwarded to all N units and their outputs are compared with each other before a final output is returned back to the application. Deciding whether or not a final output result is provided and what that output result will be depends on a *decision policy* defined by configuration. In a particular policy, all N units must produce the same result, which is then returned as output result, otherwise the application is informed that no result was generated. Thus, if any single unit implementation produces a malicious output, this output will differ from the remaining N-1 units (assuming no collusion) causing the final result to be suppressed, preventing the malicious unit from propagating its effects to the application.

Figure 5.2.1 shows the internals of a module implemented by 3 units. The input arguments are passed by the client application and the output results are returned to the application. The input preprocessor feeds the input arguments to each unit and the decision block implements a decision algorithm according to the provided decision policy. The decision policy is a configuration parameter decided by the hub administrator. Each unit is implemented by a program that runs in an independent sandbox. The input processor and the decision block logic must belong to the hub platform, which must also be responsible for setting up the units' sandboxes and the data paths represented by arrows in Figure 5.2.1.

5.2.2 Detection of Unit Result Divergence

The decision taking process is at the core of what makes N-version programming effective at countering adversarial units. In the perfect scenario, each unit is assumed to execute one of two possible versions: *benign* or *adversarial*. A version is benign if it

consists of a flawless implementation of the module's trusted function specification. A version is adversarial if it deviates from the intended specification in order to tamper with or leak sensitive data. Thus, if deviations exist between unit outputs, then at least one adversarial version is present. Since different security properties can be attained depending on the number of units in agreement, we define two decision policies providing two agreement conditions:
Total agreement (TA) policy: This policy offers the strongest security guarantees. All N units must agree on the same output result in order for an output to be returned. If this condition holds, the resulting value is returned, otherwise an error is yielded. Thus, 1 benign version only is required to exist in order to suppress the return of a corrupted result. In fact, for an attacker to be successful, all N versions must be both adversarial and collude in producing the same output.
Quorum agreement (QA) policy: Only a quorum $Q = \lfloor N/2 \rfloor + 1$ units (i.e., a majority) needs to reach consensus on a common return value. If Q is found, the module returns the agreed upon value, otherwise it reports failure. The QA policy is weaker than the TA policy because $Q > 1$ benign units need to be present to thwart an attack. Furthermore, a successful attack requires $Q < N$ colluding adversarial units.

5.2.3 Nondeterministic Inputs

One cause of unit divergence is *operational* and occurs whenever a specific trusted function depends on nondeterministic inputs, e.g., a random number, the system time or date, etc. If different units obtain different readings for the same intended input value, units' computations will likely return different results which may lead to failure in reaching a total or quorum agreement conditions and harm module's utility.

To avoid this problem, all nondeterministic inputs must be provided by the preprocessor. Sandboxes must prevent units from issuing nondeterministic system calls. If the version code depends on such calls, the input preprocessor can execute those upon request and pass the same value to all units.

5.2.4 Main Findings

In spite of its conceptual simplicity, adopting NVP in the design of *N-version trusted function modules* causes multiple side-effects in terms of the resulting utility, security, and performance. Through an in-depth study of these side-effects, we learn that:

- For N-versions that implement the same algorithm and follow the algorithm specification, it is possible to provide an N-module offering high utility as long as the software flaws in each version are residual,
- For N-versions that do not follow the same algorithm but still perform the same task, we observe that although the module's utility can be negatively affected by the outputs divergence, it can be increased by using a custom agreement algorithm specified according to the problem domain space, and
- The performance of an N-version trusted function module is typically bound by its slowest version, a condition that can be relaxed considerably by taking advantage of versions redundancy.

The experiments results showed the effectiveness of different merging approaches for the studied modules, as well as their negligible overhead compared to the worst of the modules' units. Additionally, the study features a relevant analysis on the impact of naive and malicious implementations with respect to the utility, performance, and security of the modules' output results. In conclusion, we believe N-version programming to be a valid approach in bootstrapping trust in data handling modules within an IoT environment.

5.2.5 Discussion

The NVP approach can be efficiently used at the edge for fault-tolerance and privacy preservation. On one hand, the use cases that involve operations on critical infrastructure, e.g. electricity or natural gas grid, would benefit from the controller software that is resilient to failures of certain NVP-based components. On the other hand, the third party software components with access to such a critical infrastructure might abuse their permissions and act maliciously, e.g. trigger unauthorized and potentially dangerous actions. The NVP-based system might account for such malicious activity and protect the underlying infrastructure and its users.

Within LightKone scenario, we envision the NVP-based data processing modules developed by independent developers/researchers/service providers, which act upon sensitive sensor data. For instance, in case of smart agriculture scenario (as in Gluk's use-case), there is a high risk of performing the actions that might be harmful for crops (e.g. over watering). This actions could be performed due to erroneous logic implementation, or sabotaged by the competitors. Therefore the quorum-based NVP system might be a good solution to prevent such actions from taking place. At the same time, the very same NVP system can prevent extracting sensitive sensor data about the crops, by restricting the module specifications and output format/data type.

5.3 BISEN: Efficient Boolean Searchable Symmetric Encryption

Cloud computing has had a profound impact on the way that we design and operate systems and applications. In particular, data storage and archiving is now commonly delegated to cloud infrastructures, both by companies and individual users. Companies typically want to archive large volumes of data, such as e-mails or historical documents, overcoming limitations or lowering costs of their on-premise infrastructures [5], while individual users aim at making their documents easily accessible from multiple devices, or simply avoid consuming storage capacity of their mobile devices [17].

However, data being outsourced to the cloud is often sensitive and should be protected both in terms of privacy and integrity. Private information incidents are constant reminders of the growing importance of these issues: governmental agencies impose increasing pressure on cloud companies to disclose users' data and deploy backdoors [18, 25]; cloud providers are responsible, maliciously or accidentally, for critical data disclosures [16, 24]; and even external hackers have gained remote access to users data for a limited time window [32]. Cloud outsourcing services are thus highly incentivized to address these security requirements. In particular, when storing and updating large volumes of data in the cloud it is essential to offer efficient and precise mechanisms to search and retrieve relevant data objects from the archive. This highlights the need for cloud-based systems to balance security, efficiency, and query expressiveness.

To address this tension, Searchable Symmetric Encryption (SSE) [1] has emerged as an important research topic in recent years, allowing one to efficiently search and update an encrypted database within an untrusted cloud server with security guarantees. Efficiency in SSE is achieved by building an encrypted index of the database and also storing it in the cloud [20]. At search time, a cryptographic token specific to the query is used to access the index, and the retrieved index entries are decrypted and processed. As a much necessary communication complexity optimization, most SSE schemes delegate these cryptographic computations to the cloud, as multiple index entries would otherwise have to be downloaded to the client side. However, performing sensitive operations in the cloud also leads to significant information leakage, including the leakage of document identifiers matching a query, the repetition of queries, and the compromise of forward and backward privacy [42] (respectively, if new update operations match contents with previously issued queries, and if queries return previously deleted documents). These are common, yet severe, flavors of information leakage that pave the way for strong attacks on SSE, including devastating file-injection attacks [44].

Another relevant limitation in SSE schemes is query expressiveness, as most solutions only provide single keyword match [15] or limited boolean queries (e.g. forcing queries to be in Conjunctive Normal Form and not supporting negations) [28]. This hinders system usability and may force users to perform multiple queries in order to retrieve relevant results, which leads to extra communication steps and increased information leakage.

5.3.1 Solution

In this work we address these limitations by presenting BISEN (Boolean Isolated Searchable symmetric ENcryption), a new provably-secure boolean SSE scheme that improves query expressiveness by supporting arbitrarily complex boolean queries with combinations of conjunctions, disjunctions, and negations. This is a significant improvement over the current state of art, since supporting boolean queries is fundamentally more challenging than single-keyword queries and addressing negations is a non trivial task. Furthermore, BISEN also boosts performance by minimizing the number of communication steps and data transference between clients and cloud servers. A central insight in the design of BISEN is the fact that we can securely delegate critical computations to the cloud by leveraging on a hybrid solution that combines standard symmetric-key cryptographic primitives (e.g. Pseudo-Random Functions and Block-Ciphers [30]) with remote attestation capabilities offered by modern trusted hardware, formally captured by an abstraction called Isolated Execution Environments (IEEs) [9].

An IEE is an environment that allows applications to execute in isolation from all external interference (including co-located software and even a potentially malicious Hypervisor/OS) and that provides a mechanism for the remote attestation of computed outputs. Until recently, such an abstraction could only be built through hardware that was infeasible to deploy in commodity cloud infrastructures [29], however recent advances in trusted computing have made IEEs available in commodity hardware. Prominent examples include Intel SGX [19] and ARM TrustZone [4], which are being deployed in

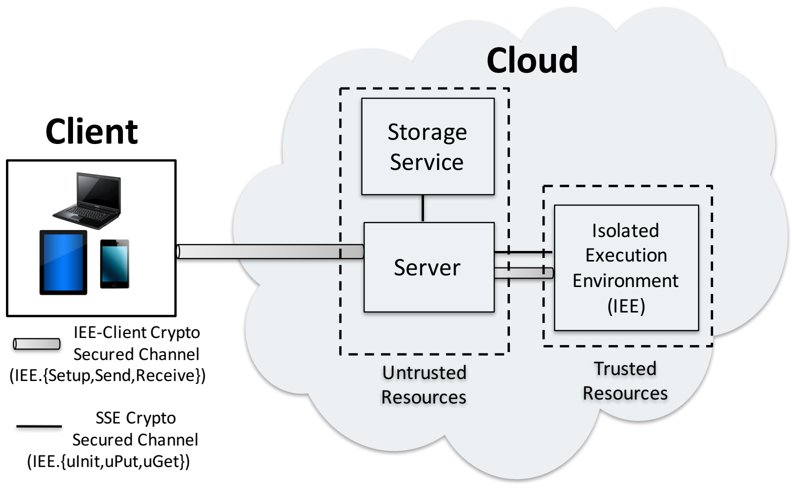
Cloud
Client
Storage Service
Server
Isolated Execution Environment (IEE)
Untrusted Resources
Trusted Resources
IEE-Client Crypto Secured Channel (IEE.{Setup,Send,Receive})
SSE Crypto Secured Channel (IEE.{uInit,uPut,uGet})

Figure 5.3.1: Overview of the proposed approach.

current desktop and mobile processors and will soon become available as part of many cloud infrastructures [40].

A main advantage of designing our system to leverage the IEE abstraction lies in its portability, as our solution can be easily instantiated using different existing or future IEE-enabling technologies as they become available in cloud platforms, while preserving security guarantees. This is also relevant when considering recent attacks on trusted hardware [34] and subsequent patches [13]. To further increase this portability, we extend the IEE formalization to support very lightweight hardware technologies (such as Intel SGX, with its limit of 128MB EPC size), complemented with cryptographically protected accesses to more abundant untrusted resources in the machine hosting the IEE or in other external cloud storage services. This extension allows us to minimize assumptions regarding the underlying technology employed in practice, while simultaneously being able to efficiently and securely support very large databases.

This approach empowers BISEN (to the best of our knowledge) to be the first forward and backward private boolean SSE scheme with minimal leakage, in the sense that updates reveal no information and queries only reveal which encrypted index entries are accessed, and verifiability against fully malicious adversaries, with reduced computation, storage, and communication overheads.

5.3.2 Technical Overview

We now present a high level view of BISEN. Figure 5.3.1 provides an overview of our approach, and the communication patterns between the central components of BISEN. In BISEN, there are four main components: the client, the trusted hardware (IEE), the cloud server, and a cloud storage service. The main idea in BISEN is to leverage IEEs as remote trust anchors, responsible for performing secure computations over sensitive cloud-stored data, which would otherwise require complex cryptographic mechanisms for performing server-side computations. To achieve this goal, the cloud server will operate the IEE and manage its communications with both the client and the cloud storage service, while the storage service will act as an extended storage for the IEE (as the IEE can potentially be lightweight, possessing small trusted storage capacity) and store BISEN's main index.

In this model, we consider both the server and storage service to be fully malicious, i.e. they may attempt to break data privacy, integrity, or computation correctness. Denial of service attacks are considered out of scope for this work.

The system model of BISEN is comprised of two main stages: bootstrapping and operational. In the bootstrapping phase, the client establishes a secure communication channel with the IEE (IEE-Client Crypto Secured Channel in Figure 5.3.1). This will consist in executing a key exchange protocol, with the server acting as intermediary, where the IEE uses hardware-specific cryptographic proofs that the code being run exactly matches that of BISEN. After this stage, the client and IEE will use this secure channel for communication, and the operational phase begins.

In the operational phase, the client can add/remove keywords to documents (i.e. update the database), as well as search for documents matching a boolean expression with multiple keywords. These functionalities are fulfilled by having the client interact with the IEE, sending encrypted messages with the desired inputs. In response, the IEE processes the clients' requests and interacts with the storage service to store/retrieve index entries (SSE Crypto Secured Channel in Figure 5.3.1), returning results to the client. BISEN's high efficiency lies in exploring the interplay between a lightweight client-side structure, isolation of cryptographic keys and secure processing within server-side IEE, and verifiable storage to a cloud storage service.

5.3.3 Discussion

The proposed approach can be used by client applications running in the edge of the network and storing data in cloud infrastructures. Thus, it is suitable for applications adopting a heavy edge model.

An interesting application scenario for BISEN is encrypted archival of email in the cloud. In such a scenario, users would be able to securely outsource the storage and management of their emails to a third-party cloud provider, while still being able to have rich search features that are commonly found in todays unsecured email cloud archival services. As studied by Zheng et al. [44], cloud email is an example scenario that can be easily targeted by file-injection attacks, hence this application enforces the need to improve the security of SSE schemes to withstand fully malicious adversaries. Furthermore, forward privacy is known to help mitigate such attacks [44], and backward privacy may have important implications in future attacks as well [11]. Overall, minimizing information leakage should be a top priority when deploying SSE schemes in practical scenarios.

Other scenarios where the proposed approach could be used is for IoT applications, where client nodes collect and store data at cloud nodes. Using BISEN, this data can be later searched without disclosing it, thus preserving privacy.

Chapter 6

Annotated Publications & Dissemination

We present a list of the scientific papers and reports where the work towards D3.2 has been presented:

- Nuno Preguiça, Carlos Baquero, Marc Shapiro. Conflict-free Replicated Data Types (CRDTs). Springer. Chapter in Encyclopedia of Big Data Technologies. 2018. https://doi.org/10.1007/978-3-319-63962-8_185-1.

  Abstract. A summary on state of the art of CRDTs.

- Ali Shoker, João Leitão, Peter Van Roy, and Albert van der Linde. Programming Models and Runtimes: Towards General Purpose Computations at the Edge. The IET. Book chapter in Ultrascale Computing Systems. Ino Press, 2018.

  Abstract. A book chapter that explains the overall approach we follow in LightKone.

- Ali Shoker, Anna Queralt, and Toni Cortes. Data Management Techniques: Advanced Conflict-free Replicated DataTypes. The IET. Book chapter in Ultrascale Computing Systems. In Press, 2018.

  Abstract. This is another work the summarizes the recent advances of the four CRDT models.

- Ali Shoker. Brief Announcement: Sustainable Blockchains through Proof of eXercise. In PODC'18: ACM Symposium on Principles of Distributed Computing. 2018.

  Abstract. This work is an exploration work given the rise of the blockchain area as a hot topic that often overlaps with highly scalable and available systems, P2P, and edge computing.

- Nuno Preguiça. Conflict-free Replicated Data Types: An Overview. Report for *Agregação* degree.

  Abstract. A habilitation report that discusses the innovations of CRDTs.

- João Leitão, Pedro Ákos Costa, Maria Cecília Gomes, and Nuno M. Preguiça. Towards enabling novel edge-enabled applications. CoRR, abs/1805.06989, 2018.

  Abstract. This work dicusses the edge system models and spectrum.

- Carlos Baquero, Paulo Sérgio Almeida, and Ali Shoker. Pure Operation-Based Replicated Data Types. arXiv CoRR. October, 2017. Submitted to IEEE TPDS.

  Abstract. In this work, we extend state of the art pure op-based CRDTs to inlucde compression at the source, to reduce the meta-data shipping overhead.

- Georges Younes, Paulo Almeida, Ali Shoker, Carlos Baquero. Tagged Causal Delivery: End-to-End Happens-Before as a Middleware Service. Submitted to Middleware 2018.

  Abstract. This work discusses the pitfalls of implementing causal middlewares presented in this report.

- Vitor Enes, Paulo Sérgio Almeida, Carlos Baquero and João Leitão. Distributing Lattices with Optimal Deltas and Join Decompositions. Submitted to DISC 2018.

  Abstract. This work presents the advances on delta state CRDT synchronization presented in this report.

- Christopher S. Meiklejohn and Heather Miller. Partisan: Enabling Cloud-Scale Erlang Applications. Submitted to SoCC 2018.

  Abstract. This submission includes the recent advances and evaluation of Partisan, presented in this report.

6.1 Dissemination

- Carlos Baquero. *CRDTs and Redis – From Sequential to Concurrent Executions*. Invited talk. Redis Conference 2018, April 24-26, 2018, San Francisco. CA.
- Carlos Baquero. *Causality is Simple*. Papers-we-Love Porto. March 22, 2018.
- Ali Shoker. *As Secure as Possible Eventual Consistency*. Invited talk. Protocol Labs workshop. Lisbon, May 2018.
- Vitor Enes. *Borrowing an Identity for a Distributed Counter*. Invited talk. Protocol Labs Research Meeting 2018.
- Nuno Afonso, Manuel Bravo, and Luís Rodrigues. *Causality for the Cloudlets: Offering Causality on the Edge With Small Metadata*. Dagstuhl Seminar 18091, Feb. 25 -– Mar. 2, 2018, Wadern, Germany.
- Peer Stritzinger and Adam Lindberg. *1000 nodes, large messages, we want it all! Prototype with new OTP 21*. Code BEAM STO, Stockholm, Sweden, May 31 - June 1, 2018.

Bibliography

[1] A Survey of Provably Secure Searchable Encryption. *ACM Computing Surveys (CSUR)*, 47(2):18:1—-18:51, 2015.

[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] Tiago Alves and Don Felton. TrustZone: Integrated hardware and software security. *ARM white paper*, 3(4):18–24, 2004.

[5] Michael Armbrust, Armando Fox, Rean Griffith, Anthony D Joseph, Randy Katz, Andy Konwinski, Gunho Lee, David Patterson, Ariel Rabkin, Ion Stoica, and Matei Zaharia. A view of cloud computing. *Communications of the ACM (CACM)*, 53(4):50–58, 2010.

[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] Carlos Baquero, Paulo Sérgio Almeida, and Ali Shoker. Pure operation-based replicated data types. *CoRR*, abs/1710.04469, 2017.

[9] Manuel Barbosa, Bernardo Portela, Guillaume Scerri, and Bogdan Warinschi. Foundations of hardware-based attested computation and application to SGX. In *Proceedings of the 1st IEEE European Symposium on Security and Privacy - EURO S&P'16*, pages 245–260, 2016.

[10] Kenneth Birman, Andre Schiper, and Pat Stephenson. Lightweight causal and atomic group multicast. *ACM Transactions on Computer Systems (TOCS)*, 9(3):272–314, 1991.

[11] Raphael Bost, Brice Minaud, and Olga Ohrimenko. Forward and Backward Private Searchable Encryption from Constrained Cryptographic Primitives. In *CCS'17*. ACM, 2017.

[12] Manuel Bravo, Luís Rodrigues, and Peter Van Roy. Saturn: a distributed metadata service for causal consistency. In *Proceedings of the Twelfth European Conference on Computer Systems*, pages 111–126. ACM, 2017.

[13] Peter Bright. Intel releases new spectre microcode update for skylake; other chips remain in beta. https://arstechnica.com/gadgets/2018/02/intel-releases-new-spectre-microcode-update-for-skylake-other-chips-remain-in-beta/, February 2018.

[14] Sebastian Burckhardt. *Principles of Eventual Consistency*, volume 1 of *Foundations and Trends in Programming Languages*. now publishers, October 2014.

[15] David Cash, Joseph Jaeger, Stanislaw Jarecki, Charanjit Jutla, Hugo Krawczyk, M Rosu, and Michael Steiner. Dynamic searchable encryption in very-large databases: Data structures and implementation. In *Proceedings of the The 21th Annual Network and Distributed System Security Symposium -NDSS'14*, volume 14, 2014.

[16] Adrian Chen. GCreep: Google Engineer Stalked Teens, Spied on Chats. Gawker. http://gawker.com/5637234, 2010.

[17] ComScore. The 2017 U.S. Mobile App Report. https://www.comscore.com/Insights/Presentations-and-Whitepapers/2017/The-2017-US-Mobile-App-Report, 2017.

[18] Tim Cook. A Message to Our Customers. Apple. https://www.apple.com/customer-letter/, 2016.

[19] Victor Costan and Srinivas Devadas. Intel sgx explained. Technical report, Cryptology ePrint Archive, Report 2016/086, 2016. https://eprint.iacr.org/2016/086, 2016.

[20] Reza Curtmola, Juan Garay, Seny Kamara, and Rafail Ostrovsky. Searchable Symmetric Encryption: Improved Definitions and Efficient Constructions. In *Proceedings of the 13th ACM Conference on Computer and Communications Security - CCS'06*, pages 79–88, 2006.

[21] Nigel Davies, Nina Taft, Mahadev Satyanarayanan, Sarah Clinch, and Brandon Amos. Privacy Mediators: Helping IoT Cross the Chasm. In *Proc. of HotMobile*, 2016.

[22] Vitor Enes. Efficient Synchronization of State-based CRDTs. Master's thesis, Universidade do Minho, 2017.

[23] Earlence Fernandes, Justin Paupore, Amir Rahmati, Daniel Simionato, Mauro Conti, and Atul Prakash. Flowfence: Practical data protection for emerging iot application frameworks. In *Proceedings of USENIX Security*, 2016.

[24] Terry Frieden. VA will pay $20 million to settle lawsuit over stolen laptop's data. CNN. http://tinyurl.com/lg4os9m, 2009.

[25] Glenn Greenwald and Ewen MacAskill. NSA Prism program taps in to user data of Apple, Google and others. The Guardian. http://tinyurl.com/oea3g8t, 2013.

[26] Chathuri Gunawardhana, Manuel Bravo, and Luís Rodrigues. Unobtrusive deferred update stabilization for efficient geo-replication. In *Proc. of USENIX ATC 17*, pages 83–95, Santa Clara, CA, 2017. USENIX Association.

[27] Nuno Santos Igor Zavalyshyn, Nuno O. Duarte. An Extended Case Study about Securing Smart Home Hubs through N-Version Programming. In *Proceedings of 15th International Conference on Security and Cryptography (SECRYPT)*, 2018.

[28] Seny Kamara and Tarik Moataz. Boolean Searchable Symmetric Encryption with Worst-Case Sub-Linear Complexity. In *EUROCRYPT'17*. IACR, 2017.

[29] Jonathan Katz. Universally composable multi-party computation using tamper-proof hardware. In *Eurocrypt*, volume 7, pages 115–128. Springer, 2007.

[30] Jonathan Katz and Yehuda Lindell. *Introduction to Modern Cryptography*. CRC PRESS, 2007.

[31] João Leitão, José Pereira, and Luis Rodrigues. Hyparview: A membership protocol for reliable gossip-based broadcast. In *Proc. of DSN'07*. IEEE, 2007.

[32] Dave Lewis. iCloud Data Breach: Hacking And Celebrity Photos. Forbes. https://tinyurl.com/nohznmr, 2014.

[33] Wyatt Lloyd, Michael J. Freedman, Michael Kaminsky, and David G. Andersen. Don't settle for eventual: Scalable causal consistency for wide-area storage with cops. In *Proceedings of the 23rd ACM Symposium on Operating Systems Principles*, SOSP '11, pages 401–416, Cascais, Portugal, 2011.

[34] LSDS Group, Imperial College London. Spectre attack against sgx enclave. https://github.com/lsds/spectre-attack-sgx, 2018.

[35] Syed Akbar Mehdi, Cody Littley, Natacha Crooks, Lorenzo Alvisi, Nathan Bronson, and Wyatt Lloyd. I can't believe it's not causal! scalable causal consistency with no slowdown cascades.

[36] Richard Mortier, Jianxin Zhao, Jon Crowcroft, Liang Wang, Qi Li, Hamed Haddadi, Yousef Amar, Andy Crabtree, James A. Colley, Tom Lodge, Tosh Brown, Derek McAuley, and Chris Greenhalgh. Personal Data Management with the Databox: What's Inside the Box? In *Proc. WCAN CoNEXT*, 2016.

[37] Nuno Preguiça, Joan Manuel Marques, Marc Shapiro, and Mihai Letia. A Commutative Replicated Data Type for Cooperative Editing. In *Proceedings of the 2009 29th IEEE International Conference on Distributed Computing Systems*, ICDCS '09, pages 395–403, Washington, DC, USA, 2009. IEEE Computer Society.

[38] Nuno Preguiça. Conflict-free replicated data types: An overview. *Agregação* degree document, Universidade NOVA de Lisboa.

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

[40] Mark Russinovich. Introducing Azure confidential computing. https://azure.microsoft.com/en-us/blog/introducing-azure-confidential-computing/, 2017.

[41] Marc Shapiro, Nuno Preguiça, Carlos Baquero, and Marek Zawirski. Conflict-free replicated data types. In *Proceedings of the 13th International Conference on Stabilization, Safety, and Security of Distributed Systems*, SSS'11, pages 386–400, Berlin, Heidelberg, 2011. Springer-Verlag.

[42] Emil Stefanov, Charalampos Papamanthou, and Elaine Shi. Practical Dynamic Searchable Encryption with Small Leakage. In *Proceedings of the The 21th Annual Network and Distributed System Security Symposium -NDSS'14*, 2014.

[43] 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.

[44] Yupeng Zhang, Jonathan Katz, and Charalampos Papamanthou. All Your Queries Are Belong to Us: The Power of File-Injection Attacks on Searchable Encryption. In *Proceedings of the 25th USENIX Security Symposium - Security'16*. USENIX Association, 2016.

Appendix A

Threat Model of LightKone use-cases

Security analysis

João Marco C. Silva

1 Introduction

The security analysis formalization consists in a high level threat model focused in both, the system assets and software. The composite approach is justified by the unavailability of detailed software specification in most use cases due to either their proprietary nature or development stage. For both scenarios, a comprehensive threat model provides early understanding of security requirements through an abstract representation rather than the code itself, which allows addressing security along the system design and component selection.

The threat modelling adopted in this report is composed of three steps:

System decomposition provides a clear understanding about its entities, applications and interconnections, which reveal their trust boundaries and attack surfaces. For each use case, the decomposition stage has provided a Data Flow Diagram (DFD) that supports the threat identification stage;

Threat identification consists in identifying and categorising threats for every element of the system described in previous stage. The methodology used classifies threats the systems are exposed to in six classes: (i) Spoofing; (ii) Data tampering; (iii) Repudiation; (iv) Information disclosure; (v) Denial-of-Service - DoS; and (vi) Elevation of privilege;

Countermeasures and mitigation for threats in elements with sufficient information, some approaches on how to address them either through off-the-shelf solutions or academic research approaches are provided.

1.1 UPC - Coordination between servers and data storage for the Guifi.net monitoring system

The system described in Sections 1 and 2 of Chapter 3 (i.e. Deliverable 2.1) is composed of three main components. First, there are approximately 34,000 active nodes, such as routers and switches, that collaboratively provide or consume diverse network services (e.g. Internet connectivity). Their state and available resources need to be monitored for the purpose of billing, capacity planning and service provision. This is done by the second component of the system - the monitoring servers which consist in instances of the SNPServices tool (approximately 200 active servers). They coordinate with each other in order to distribute the workload of monitoring tasks and maintain a shared database containing the results of monitoring. Finally, the main Guifi.net website and the central database server aggregate and display data provided by the monitoring servers. The database also maintains the list of nodes to be monitored which it shares with the monitoring servers.

Figure 1 presents a Data Flow Diagram (DFD) identifying all the elements into the monitoring system, their trust boundaries and interconnections. The security analysis presented in Section 1.1.2 addresses the threats each element is exposed to.

1.1.1 Trust model and assumptions

We assume the central website and database server administrators to be trusted and not maliciously collude with the administrators of monitoring servers. The list of nodes to be monitored provided by the central server is assumed to be initially complete and containing accurate nodes description.

In addition, we assume that data provided by routers and switches being monitored through Simple Network Management Protocol (SNMP) are trustworthy, however its administrator might be malicious.

1.1.2 Security analysis

This section discusses the per element security analysis encompassing their threats, actors and strategies to cope with them. Table 1 summarises the classes of threats to which each element in the monitoring system is exposed to. As detailed, the system entity with the major range of identified threats is also the one being addressed into the LightKone project scope, i.e., the

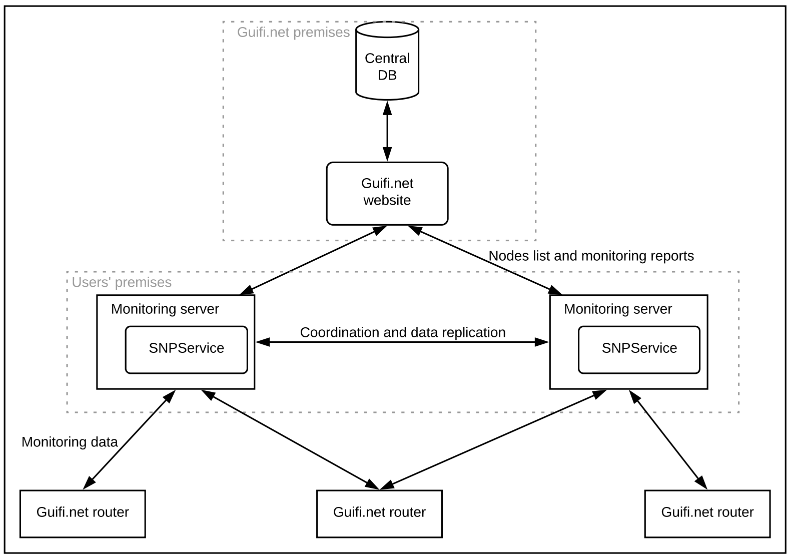

Figure 1: DFD - Guifi.net monitoring system

monitoring servers (see Figure 1).

Table 1: Class of threats identified per element.

Threat	**Centrad DB**	**Guifi.net**	**Monitoring server**	**Network node**
Spoofing	-	✗	✗	✗
Data tampering	-	-	✗	-
Repudiation	-	-	✗	✗
Information Disclosure	-	✗	✗	✗
Denial-of-Service (DoS)	-	✗	✗	-
Elevation of Privilege	-	-	✗	✗

Guifi.net website

Although being considered a trusted entity, the attack surface revealed by connecting the website with external entities with different privileges reduce the central service trust boundaries. The main threats identified in our analysis include:

Spoofing

- A "phishing attack" could steal credentials from system administrators as current website[1] does not offer secure connection. It is easily solved by deploying the HTTP Secure (HTTPS) extension in order to provide authentication for the website, users' privacy and integrity for the exchanged data while in transit;
- An attacker can connect to the website resorting to a week authentication system or a stolen credential. It might be mitigated through multi-factor authentication approaches.

Information disclosure

- Sensitive information regarding additional details about network nodes (i.e., their roles in terms of network topology, performance or economic interest) could be exposed if messages or channel are not encrypted;
- Although not receiving clear data about individual users and services through reports from monitoring servers, the aggregated data can reveal sensitive information about them if messages or channel are no encrypted.
- With the absence of authentication for the endpoints in the network connection, an attacker can act as a "man in the middle" and eavesdrops the entire communication, including monitoring data or administrators' credentials.

Denial-of-Service (DoS)

- As a central service, an attacker can make the guifi.net website unstable or unavailable through a Distributed Denial-of-Service (DDoS) attack. Defining a DDoS response planning which includes mitigation services or distributing the website workload across the network is usually efficient and straightforward.

[1]http://guifi.net/

Monitoring server

As a consequence of the crowd-funded nature of guifi.net infrastructure, monitoring servers are deployed in users' premises. It increases their attack surfaces, therefore making them the main target asset in such system. The main threats we identified in such element are:

Spoofing

- A malicious user could set up a fake monitoring server that would not actually monitor nodes, or that would do it inaccurately. This might be addressed resorting to strong authentication, mainly with public-key based certificates, which also encompasses other threats further described;

Data tampering

- An attacker could tamper with data collected from network nodes and locally stored. As it is deployed in users' premises, this is further stressed without integrity protection for the local database and/or week Access Control Lists (ACLs);
- Monitoring servers' administrators might modify the monitoring data especially for the nodes they control. This could be done for the purpose of improving their nodes statistics and/or lowering the statistics of other nodes in the network;
- Assigning two monitoring servers to each network node, as suggested in D2.1, is not enough in order to guarantee integrity of collected data. Collusion attacks or a Byzantine server are examples in which this approach fails;
- Replicating the local measurements with other monitoring servers increases the system resilience and availability, however it raises some different threats, such as data tampering or omission. As measurement data is only consumed by the central database, it might be encrypted with the DB public key before being transmitted to other serves. The computational burden of such validation can also be transferred to the

network edge, namely, the monitoring servers. This is, indeed, an on-going research led by the security team in LightKone and potentially extensible to all use cases and to be integrated into the runtime system.

Repudiation

- One of the main concerns in the security aspects of guifi.net monitoring system is the total absence of auditing logs and rolling back offending updates mechanisms. This might be coped by deploying one off-the-shelf solution from several currently available.

Information disclosure

- The same threats previously identified for data transmission, i.e., details about network nodes, traffic patterns and disclosure of sensitive information are applied to data stored locally. Here, there are diverse vector attacks, such as, bad or no ACLs, weak authentication, malicious local administrators, etc. Although frenquently running in low power devices, i.e., Single-Board-Computers (SBCs), an encompassing solution for such large attack surface might be to resort to cross-platform software and/or hardware sandboxing in order to ensure contained operations in third-parties and heterogeneous entities. This is another solution extensible to the majority of use cases with current research under development into the Lightkone scope;

Denial-of-Service (DoS)

- An attacker can make a server unusable or unavailable through the local network, mainly for the instances of SNPSevices running on Single-Board Computers (SBCs);
- Every monitoring server probing all 34,000 network nodes (in automatic assignment mechanism) could make the network unstable or even unavailable. Passive measurement approaches are usually applied to avoid intrusive traffic from probing systems. In addition, considering that normally checking a node's status takes some time, such technique may potentially leave benign monitoring servers out of work.

Elevation of privilege

- In such distributed architecture, with local administrators, an attacker could elevate its privilege in order to compromise a server by deploying any of the attacks previously identified for this element.

Network nodes

As discussed in Section 1.1.1 routers and switches are considered trusted entities. However, the monitored nodes also include others monitoring servers and different devices running diverse services, such as firewalls, proxies, web services, etc. Hence, a deeper analysis has surfaced following threats for this general entity (i.e., network nodes).

Spoofing

- A malicious user could set up a fake network node that would not provide the actual status of the intended node. It might be done combining IP spoofing and poor security mechanisms in the most deployed versions of the SNMP (i.e., version 2). A strategy for this consists in resorting to SNMPv3 or encrypted transport layer protocols.

Data tampering

- Local administrators might tamper with the nodes settings and report fabricated information. By doing so, they might appear providing a better service and thus attract more user traffic. It might be coped by using mutual authentication mechanisms. Additionally, they could abuse the monitoring tasks pool by assigning themselves to new tasks as they appear and immediately reporting fake data;

- An attacker could intercept and tamper with values collected from network nodes that usually use plain SNMP messages and an unencrypted link with the monitoring server. As the monitoring results traverse the general guifi.net infrastructure, such threat can be materialized through a ”man in the middle” attack;

Repudiation

- As previously discussed, this system lacks an auditing mechanism, hence an attacker could use a common shared key to authenticate as different principals, confusing the information in nodes’ logs.

Information disclosure

- An attacker could seek aggregate information between network nodes and monitoring server with unencrypted messages or communication channel.

Elevation of privilege

- An attacker could elevate link priority by changing its class of service in routers using community credentials.

1.1.3 General security requirements

This section presents a set of general security requirements derived from the security analysis of UPC's monitoring system.

- The system must ensure that each network node is monitored by at least three independent monitoring servers for cross-checking. If any persistent divergence in the reported data is detected the corresponding server must be flagged and reported;
- The communication channels among entities belonging to the monitoring system or its related messages must be encrypted;
- The system must protect the integrity of the monitoring database. All operations must be logged and authenticated. The monitoring servers can only write or modify the database entries for the nodes they were assigned to. Any modification of monitoring data for other nodes must be forbidden;
- The status data reported by the network node must be checked for correctness, integrity and authenticity before being added to the database. The monitoring software running on the node must be protected and verified for each report;
- The monitoring tasks distribution must not only be fair and efficient but also abuse-resistant. No monitoring server must be responsible for the majority of nodes;
- The system must be resistant to network partitions and database corruption. The collected data should be persistent and remain available when any of the monitoring servers fails or disconnects.

1.2 Scality - Pre-indexing at the edge

The system described in Section 1 of Chapter 4 (i.e, Deliverable 2.1) consists in Zenko Multi-Cloud Controller, an open-source project that provides a unifying storage interface and advanced search capabilities (by indexing the data) in multi-cloud backend data storage systems, including Amazon S3, Microsoft Azure and Google Cloud Platform.

Figure 2 presents the main components in this system, which comprises Client-applications that write data to the storage system and retrieve the search results; precomputing nodes at the edge (i.e., Zenko) that aggregate client requests and forward data to higher levels of the system; and backend storage systems where the client data are stored (i.e. Clouds). The pre-computing nodes also perform various computations on client data including encryption, hash signature generation, indexing and index lookups. Moreover, some customers have legacy applications (i.e., external applications), that directly communicate with the cloud systems and are not yet connected to Zenko.

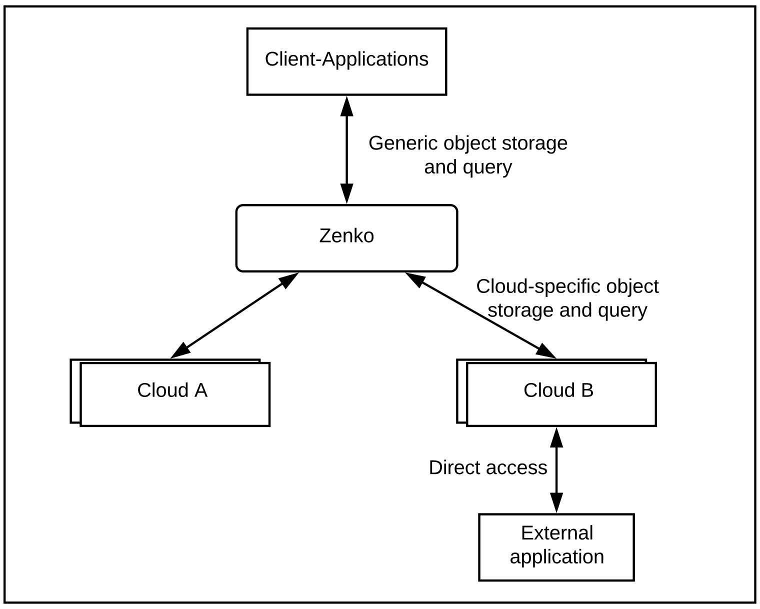
Client-Applications
Generic object storage and query
Zenko
Cloud-specific object storage and query
Cloud A
Cloud B
Direct access
External application

Figure 2: DFD - Pre-indexing at the edge

1.2.1 Trust model and assumptions

We assume that the edge precomputing service provider does not maliciously collude with the application developer or cloud storage provider. We also assume that the software and hardware platform where the client applications are executed to be secure, however application developers and local administrators could try to exfiltrate or temper with sensitive user data, as detailed in Section 1.2.2. We do not address attacks on user data performed by any malicious applications running on the same platform that are not involved in the described use case scenario.

1.2.2 Security analysis

Although the presented use case is mainly focused in the edge node (i.e., Zenko), we present threats identified for all elements which might impact the security aspects of the whole system. Table 2 summarises the classes of threats to which the system entities are exposed to.

Table 2: Class of threats identified per element.

Threat	**Client-Applications**	**Zenko**	**Cloud**	**External application**
Spoofing	✗	✗	-	✗
Data tampering	✗	✗	✗	✗
Repudiation	✗	✗	-	✗
Information Disclosure	✗	✗	✗	✗
Denial-of-Service (DoS)	-	✗	-	-
Elevation of Privilege	✗	✗	-	✗

Client-applications

As previously discussed, we generically assume that software and hardware platforms where the client applications are executed to be secure, however, some identified threats might impact the whole system. Therefore, we discuss them for the sake of the analysis completeness.

Spoofing

- An attacker could confuse a client by setting a fake edge node through IP spoofing and a phishing process in the known transport layer port. It might be coped with mutual authentication and/or encrypted communication;
- An attacker could steal credentials stored in Client-application and reuse them to access sensitive data. Resorting to strong authentication mechanisms should solve this threat.

Data tampering

- Malicious application developers could try to tamper with sensitive user data. Having direct access to the unencrypted data, such an adversary could modify;
- An attacker could intercept and tampering with data being transmitted between client-applications and edge node with unencrypted messages or communication channel.

Repudiation

- Malicious application developers could compromise the log system.

Information disclosure

- An external attacker can intercept and eavesdrop on the communication between the client-applications and the edge computing node with unencrypted messages of communication channel;
- Malicious application developers could try to exfiltrate sensitive user data. Having direct access to the unencrypted data, such an adversary might leak it to unauthorised parties.

Elevation of privilege

- A local administrator can elevate some user privilege locally giving access to sensitive user data in the system.

Edge node - Zenko

Spoofing

- An attacker could use authentication tokens locally stored to access cloud systems;

Data tampering

- Malicious application developers could try to tamper with sensitive user data. Having direct access to the unencrypted data, such an adversary could modify it according to his needs;
- In case of pre-encryption indexing, the edge node maintainer and service provider could perform the same attack;
- Edge node maintainer can also abuse its cloud storage access rights and manipulate the data without the user consent.

Repudiation

- An attacker could use a shared key to authenticate as different entities, confusing the information in the logs;

Information disclosure

- Malicious application developers could try to exfiltrate sensitive user data. Having direct access to the unencrypted data, such an adversary might leak it to unauthorised parties;
- In case of pre-encryption indexing, the edge node maintainer and service provider could perform the same attack;
- An attacker could access sensitive user information through a search indexer;
- An attacker could access authentication tokens stored locally for accessing cloud storage backends.

Denial-of-Service (DoS)

- Malicious applications could perform DoS attacks preventing the application and potentially other service providers to operate as intended;

Elevation of privilege

- An application developer can provide a pointer across a trust boundary, rather than data which can be validated.

Cloud systems

Data tampering

- Cloud storage providers can have unlimited and untraceable access to the user data stored at their premises.

Information disclosure

- The user data can be analyzed for user profiling or shared with third parties;
- An attacker can act as a "man in the middle" with end points of a network connection with no authentication or encrypted messages;

External applications

Although outside of Scality control, some threats identified in legacy application (i.e., without Zenko integration) might compromise the whole system. Most of them are the same observed for the Client-applications integrated with Zenko.

Spoofing

- An attacker could steal credentials stored locally and reuse them to access sensitive data.

Data tampering

- Malicious application developers could try to tamper with sensitive user data. Having direct access to the unencrypted data and with no validation through Zenko, such an adversary could modify it;
- Poor local ACLs could allow external applications tamper with data already stored in cloud systems;

- An attacker could intercept and tampering with data being transmitted between external application and cloud systems with unencrypted messages or communication channel.

Repudiation

- Legacy solutions might not provide log system.

Information disclosure

- Malicious application developers could try to exfiltrate sensitive unencrypted user data already stored in cloud systems;
- An external attacker can intercept and eavesdrop on the communication between the client applications and the cloud systems with unencrypted message or channel;

Elevation of privilege

- A local administrator or an attacker can elevate some user privilege locally giving access to sensitive user data.

1.3 General security requirements

- The system must provide defense mechanisms preventing access to unencrypted data flows;
- All the parties involved in the system must be authenticated and authorised by the user;
- The communication channels among entities belonging to the system or their related messages must be encrypted;
- The system must provide the ways for the user to verify the integrity of data at any stage of processing.

1.4 Stritzinger - No-Stop RFID

Within this use case scenario a system presented in Section 1 of Chapter 5 (i.e., Deliverable 2.1) and described in Figure 3 is composed of the following components: the RFID tags that are moving on the conveyor belt, the RFID readers that can read and write the data to and from the RFID tags, and a distributed cache of RFID content featuring completed and missing steps. The readers communicate with each other through Ethernet network by flooding all the latest updates to keep the cache data consistent.

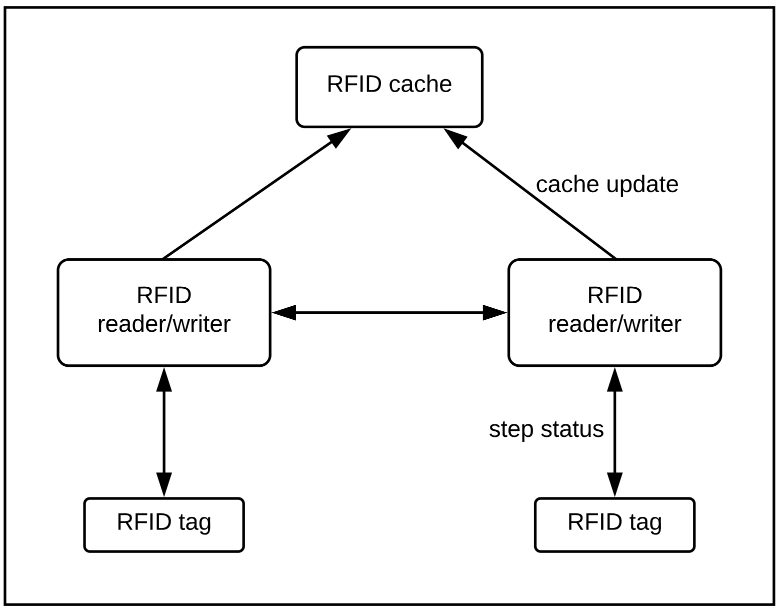

Figure 3: DFD - No-Stop RFID

1.4.1 Trust model and assumptions

We assume that the factory premises are protected from physical access by unauthorised parties that could otherwise gain full control over the cached data. Given such restricted environment of the factory premises the only actors constantly interacting with the system are the factory workers. However, the RFID readers manufacturers and firmware developers have limited access to the system during its deployment, testing or upgrading.

In addition, due to resource constraints of the readers, their communication is unencrypted and solely relies on efficiency of firewall rules at the higher level of the network and the aforementioned physical access control at the factory.

1.4.2 Security analysis

Despite the protected environment in which the No-Stop RFID system is deployed, its elements face threats that could compromise the manufacturing process. The classes of threats to each element is exposed to are presented in Table 3 and further discussed.

Table 3: Class of threats identified per element.

Threat	**RFID tag**	**RFID reader/writer**	**RFID cache**
Spoofing	✗	✗	✗
Data tampering	✗	✗	✗
Repudiation	✗	✗	✗
Information Disclosure	✗	✗	✗
Denial-of-Service (DoS)	-	✗	✗
Elevation of Privilege	-	-	-

RFID tag

Spoofing

- An attacker can use cheap RFID tags to provide bogus data to the reader and consequently to the cache system.

Data tampering

- The factory workers might be intentionally or unintentionally tampering with the data in RFID tags.

Repudiation

- Without authentication, it is impossible to identify if data is being provided by an authentic tag.

Information disclosure

- Without cryptography, an attacker could read data in the tags resorting to ordinary RFID readers.

RFID reader/writer

Spoofing

- Without authentication, an attacker could write bogus data in RFID tags, compromising the factory control.

Data tampering

- The factory workers might be intentionally or unintentionally tampering with the data being read through unencrypted channel;
- RFID readers manufacturers and their respective software developers might also carry a stealthy attack.

Repudiation

- Without authentication, it is impossible to identify if data was provided by an authentic writer.

Information disclosure

- By overpassing the firewall, an attacker could eavesdrop data transmitted through unencrypted ethernet communication among readers and between reader and cache system.

Denial-of-Service (DoS)

- By using cheap RFID tags an attacker can perform a DoS attack to the readers slowing down or stoping the manufacturing process

---

RFID cache

Spoofing

- An attacker can compromise the firewall and gain unrestricted access to the readers' network;

Data tampering

- The factory workers might be intentionally or unintentionally tampering with the cache data;
- An attacker could manipulate the data transmitted through unencrypted ethernet communications.

Repudiation

- Without authentication, it is impossible to identify if data is being provided by an authentic RFID writer.

Information disclosure

- By overpassing the firewall an attacker can access sensitive information about the factory.

Denial-of-Service (DoS)

- By overpassing the firewall an attacker can perform a DoS attack to the cache system slowing down or stoping the manufacturing process.

1.4.3 General security requirements

- The system must be able to protect the cache data from eavesdropping or manipulating by unauthorised parties;
- The communication between the readers must be secured and resistant to DoS attacks;
- The system must provide a way for the factory owners to verify the authenticity of RFID tags and reader software and hardware;
- Any modifications to the cache data must be detected and flagged without affecting the manufacturing process.

1.5 Gluk - Agriculture sensing analytics

The use case scenario described in Chapter 6 (i.e., Deliverable 2.1) presents a sensor-based platform for precision agriculture which allows for collecting, analyzing and reacting to field sensor data in near real-time. It collects the readings from thousands of field sensor nodes, aggregates these values at the edge gateway and finally sends them to the cloud service for further processing. The cloud backend provides an interface for statistical analysis and data visualisation (i.e., Dashboard in Figure 4), and creates rules and patterns for actuators using machine learning. These rules are then uploaded back to the edge gateway which allows for local real-time decision making and action without the cloud support. Figure 4 shows all the main entities and relations into this use case.

The proposed system is planned to be used by several parties. Farmers rely on the system to facilitate and automate the process of farming, generate agriculture analytics and provide useful insights on their activity. Insurance companies review statistical data in order to calculate a potential risk for an insured product (e.g. crops). Scientists and researchers share trigger rules and patterns that allow for an optimised cultivation. Software developers create applications for edge nodes and actuators to react to changes in the environment according to the configured rules and patterns.

1.5.1 Trust model and assumptions

We assume the hardware of sensor and gateway nodes to be trusted and operating as stated in the specification sheets. In this way, potential attackers might be competitors of the system owner, application developers, rules and patterns creators, cloud backend service and wireless connectivity providers. The system owner himself might appear as an attacker. For instance, by manipulating the statistical data before releasing it to the insurance company. A detailed model encompassing all the identified threats to which the system is exposed to is presented in Section 1.5.2.

We also assume the front-end (i.e., Dashboard) to only read data from the cloud in order to provide visual information to its users, e.g., farmers.

1.5.2 Security analysis

Table 4 summarises the classes of threats identified for every element of the system detailed in the data flow diagram depicted in Figure 4.

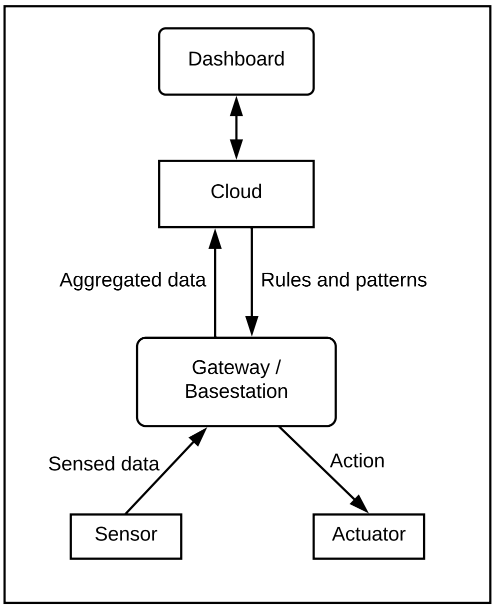

Figure 4: DFD - Agriculture sensing analytics

Table 4: Class of threats identified per element.

Threat	**Dashboard**	**Cloud**	**Gateway**	**Sensor**	**Actuator**
Spoofing	✗	✗	✗	✗	✗
Data tampering	✗	✗	✗	✗	✗
Repudiation	-	✗	-	✗	✗
Information Disclosure	✗	✗	✗	✗	✗
Denial-of-Service (DoS)	-	-	✗	✗	✗
Elevation of Privilege	-	-	✗	-	✗

Dashboard/GUI

According to Gluk, due to the lack of technological skills of the main users (i.e., farmers), the system has to provide simple and transparent ways of interaction at the front end.

Spoofing

- A "phishing attack" could steal credentials from system administrators;
- An attacker can connect to the dashboard resorting to a week authentication system.

Data tampering

- In presence of unencrypted messages or communication channel, a "man in the middle" attack could provide bogus data to farmers or other stakeholders.

Information disclosure

- An attacker can act as a "man in the middle" and access sensitive information in communications without endpoints connection authentication.

Cloud - backend system

Spoofing

- An attacker could use the lack of encryption or weak authentication factors to access the backend system as an analyst.

Data tampering

- Cloud storage providers can have unlimited access to the client data stored at their premises, allowing untraceable modifications;
- An analyst could provide bogus rules and patterns forcing a wrong behavior by the actuators in the field;
- Farmers could manipulate the statistical data before releasing it to insurance companies;
- The backend system could provide bogus information about potential failures (reset, replace, or recalibrate the sensor, etc).

Repudiation

- Cloud storage providers can have untraceable access to the client data.

Information disclosure

- The client data can be analyzed or shared with third parties. Such information might be of a commercial secret, thus leaking it might cause the company to loose its competitive advantages on the market.

Gateway - basestation

Spoofing

- An attacker could set a fake gateway either providing crafted data to the backend system or accessing business intelligence. A spurious gateway could also read data transmitted from sensors and more critically to actuators.

Data tampering

- By modifying the rules and patterns an attacker could manipulate the actuators state and affect the utility of the provided services;
- The gateway can provide crafted sensed data to the backend system, compromising data analytics.

Information disclosure

- An attacker might configure a malicious version of the gateway application to leak the sensor data, rules and patterns to third parties;
- Unencrypted communications between sensors/actuators and the gateway might expose sensitive data.

Denial-of-Service (DoS)

- An attacker with sufficient resources and competence could carry a DoS attack on wireless medium. By simply broadcasting radio signals on the same frequency the nodes use for communication and overpowering the original signal, an attacker can efficiently jam the network.

Elevation of privilege

- An attacker could reprogram the gateway to accept commands from unauthorised entities;
- Bandwidth resources can be abused by a malicious application running on the gateway by generating significant volumes of traffic and causing higher bills;
- An attacker could control the state of actuators and send false administrative commands.

Sensor

Globally, the main threats to which sensors are exposed to arise from the lack of authentication and encrypted communication channels.

Spoofing

- Without node authentication, an attacker can easily set fake sensors.

Data tampering

- An attacker might send crafted sensor readings to the gateway node to trigger certain automations that might have devastating effects on the crops;
- In multi-hop topologies, an unauthorised sensor could tamper with data from many other sensors, which might be catastrophic.

Repudiation

- Considering the lack of authentication, it might be impossible to identify data from fake sensors;

Information disclosure

- Without encryption, an attacker can read information from the wireless communications.

Denial-of-Service (DoS)

- An attacker might manipulate the sensor node software so that it constantly transmits sensor readings or any arbitrary data to the gateway node or other sensor nodes, draining its battery faster;
- An attacker can broadcast radio signals on the same frequency the sensors use for communication, jamming the network.

Actuator

As observed for the sensors, the absence of secure communications is the main source of threats in actuator nodes.

Spoofing

- In multi-hop topologies, a fake actuator could distribute diverse bogus administrative commands.

Data tampering

- An attacker could send crafted action requests or reprogramming codes to compromise actuators;

Repudiation

- It might be impossible to identify which entity has sent a specific action request to the actuators.

Information disclosure

- Without encryption, an attacker can read information from the wireless communications, which correlated with sensed data might expose sensitive data about the farm analytics.

Denial-of-Service (DoS)

- An attacker could provoke a battery depletion through continuous requests;

- An attacker can broadcast radio signals on the same frequency the actuators use for communication preventing them to receive authentic requests.

Elevation of privilege

- An attacker could control the state of actuators from the cloud backend and send false administrative commands.

1.5.3 General security requirements

- Every node in the network must be identified and checked for any software or hardware modifications before being added to the network. All nodes failing this check must remain isolated from the rest of the network;
- The gateways must be able to identify the sensor nodes abusing the wireless and battery resources and have ways to isolate those from the rest of the network;
- The system must prevent any changes to the sensor readings and the actuators commands at the time they are generated, transmitted and processed. Any manipulation with the data along the way must be detected and reported;
- The data generated by the system must be only accessible to the authorised entities (e.g. network nodes, applications, system administrators). The system must ensure secure communication channels and storage;
- The system must be robust and fault tolerant. It must adapt to changes in the wireless environment and withstand the DoS attacks. Each communication channel must be replicated to ensure data availability and timely delivery;
- The sensor data must be verified for correctness before the action is taken upon it. The system must detect the deviations in the sensor reading reported by the sensors that are close to each other, cross-check the values and discard suspicious readings. Any repeated, conflicting or inconsistent actuators commands must be detected and discarded as well.