Proofs of Timely-Retrievability for Third-Party Storage at the Edge

Rita Prates
rita.prates@tecnico.ulisboa.pt

Instituto Superior Técnico
(Advisors: Professor Luís Rodrigues and Professor Miguel Correia)

Abstract. Edge computing is a model that places servers close to the edge of the network, in order to assist applications that run in resource-constrained devices. Edge servers may be required to keep local copies of data that is also stored in the cloud, to serve clients with low-latency. Because the capacity of edge nodes is limited, providers of edge storage may be tempted to oversell their capacity and to hide this behavior by fetching, on-demand, data from the cloud instead of serving it from local storage. In this report, we study techniques that can help in detecting this type of misbehavior. We survey the main auditing mechanisms that have been proposed in the literature to cope with rational behavior in distributed storage systems, and propose a new proof of storage for edge scenarios, that we name *proof of timely-retrievability*. The report also describes the initial design of an auditing mechanism to obtain this type of proof.

Table of Contents

1 Introduction

Today, there are many scenarios where an end-user or an organization stores data on machines run by third-parties, either to ensure durability and availability, or to provide others with low latency when accessing data. Relevant examples include cloud storage (such as Dropbox, iClould, googleDrive, and many others), peer-to-peer storage (including the storage services used for crypto-currencies [1, 2]), content distribution networks (for instance, Akamai), and more recently, edge storage [3].

In these scenarios, the user has some expectations regarding the quality of service to be provided by the third-party. This, naturally, includes the expectation that the third-party will not discard or corrupt the stored data, but also that it stores the data while preserving some additional Service Level Agreement (SLA) properties. Those properties may be data storage in multiple distinct machines, data storage in specific geographic locations, or serve data clients with some bounded delay.

Unfortunately, a rational [4–6] third-party may opt to avoid complying to some of these expectations, if it can gain some benefit and pass unnoticed. For instance, the third-party can keep the data in fewer replicas and/or fewer locations than agreed, assuming that it can be impossible for the customer to audit how many replicas are used or where these replicas are located. This motivated the development of auditing techniques, able to extract proofs of storage, i.e, evidences that the third-party is complying with (or violating) the SLA.

In this work we address the problem of deriving audit techniques tailored for edge-storage scenarios, in particular, that are able to verify that data is placed by an edge-storage provider in locations that are able to guarantee that data is served with low-latency to end-users. Because the capacity of edge nodes is limited, providers of edge storage may be tempted to oversell their capacity and to hide this behavior by fetching, on-demand, data from the cloud instead of serving it from local storage. We survey the main auditing mechanisms that have been proposed in the literature to cope with rational behavior in distributed storage systems and propose a new proof of storage for edge scenarios, that we name *proof of timely-retrievability*. The report also describes the initial design of an auditing mechanism to obtain this type of proof.

The rest of the report is organized as follows. Section 2 briefly summarizes the goals and expected results of our work. In Section 4 we present all the background related with our work. Section 5 describes the proposed architecture to be implemented and Section 6 describes how we plan to evaluate our results. Finally, Section 7 presents the schedule of future work and Section 8 concludes the report.

2 Goals

This work addresses the problem of auditing edge storage providers, to assess if copies of data items are stored in fog nodes in a way that supports its retrieval with low latency.

Goals: To design a auditing mechanism that is able to extract a proof of timely-retrievability, i.e., a proof that a given fog node is able to serve requests without violating some given data access latency constraint δ.

To implement the auditing mechanism we plan to design a challenge that requires the fog node to assess, in sequence, a pseudo-random set of data items and to respond to the challenge in a timely manner. The challenge must be designed such that if the fog node does not store locally a significant fraction of the objects, it will be unable to respond in time. To ensure that the challenge is executed by the fog node being audited, and not in some other, we plan to leverage on the trusted execution environments supported by modern hardware, namely on the Intel Software Guard Extensions (SGX).

The project will produce the following expected results.

Expected results: The work will produce i) a specification of the auditing mechanism; ii) an implementation for nodes that support the Intel SGX, iii) an extensive experimental evaluation to assess the accuracy of the produced proofs of timely-retrievability.

3 Data Storage on Third-Parties

There are several scenarios where a user may opt to store data on machines controlled by a third-party. In this report we consider three main examples, namely, peer-to-peer storage, cloud storage, and edge storage. Using third-parties for data storage can bring several advantages, including reduced cost (it may be more cost efficient to run a large shared storage service than multiple small storage devices), fault tolerance (by keeping multiple copies of the data, possibly in different geographical locations), and reduced latency when accessing the data (copies may be placed in locations near to the users). However, it also brings challenges, given that the third-party may not comply with the service agreed. In this section we introduce different scenarios where storage by third-parties is used and list the main challenges that are raised in these settings.

3.1 Peer-To-Peer Storage

Peer-To-Peer (P2P) storage is a storage model where different parties coordinate to keep redundant copies of each other files. P2P storage is driven by two main observations. In first place, the capacity of storage devices, such as hard-disks, has increased significantly over the years, and many users consume just a fraction of their local storage capacity; the excess capacity remains underused. In second place, it may be impractical for many users to deploy their own devices in multiple locations, a requirement to tolerate the loss of local storage due to theft, fire, or other disasters.

The idea of P2P storage is that users can cooperate to store copies of files from other users, leveraging the spare capacity of each individual storage. This

approach has the advantage of being inexpensive and has the potential to provide a high-level of reliability. If the users are located in different geographical locations, and multiple copies of a file are created, the likelihood of loosing files can become arbitrarily small.

However, P2P storage relies on reciprocity: one party stores data for a third party under the assumption that the third party will also store data for other parties. This opens the door for free-riding [7–9], where a party benefits from the contribution of other nodes without doing its part (for instance, by failing to locally store copies of files that have been assigned to it).

3.2 Cloud Storage

Cloud storage is a model where large service providers offer storage services to the end users for a fee. Cloud storage is widely used today, and there is a large variety of companies that sell this type of service (Amazon, Google, Microsoft, and others). The main drive of cloud storage is the cost savings that can be achieved when the storage servers, and the infrastructure to maintain those servers (power supply, cooling, etc) are shared by a large number of users. This allows cloud storage providers to maintain multiple copies of the data, in different servers, possibly in different locations, for a fraction of the cost that would need to be payed by individual users, if they attempted to build a similar infrastructure for their own use only.

Users of cloud storage pay for the service under the assumption that the cloud provider implements the necessary fault-tolerance techniques to make highly unlikely that data can be lost due to the failure of individual components, such as a hard-drive or a server. However, it is not easy to the customer to audit if the expected levels of redundancy are actually implemented by the cloud provider. A dishonest provider could be tempted to maintain less copies of the data than advertised, to reduce costs.

3.3 Edge Storage

There is a range of new tasks, such as image processing for face or object recognition [10, 11], as marker detection [10], and just-in-time video indexing [12], that require a response time below 5–30 milliseconds [13]. For these application, it is fundamental to access data with low-latency, something that cannot be guaranteed with cloud storage alone.

Edge computing, that consists in placing computing resources closer to the edge of the network, emerges as a strategy to offer low latency access to processing and storage resources to these applications. Edge servers, also known as *fog nodes* or *cloudlets*, extend cloud services closer to users. Fog nodes, that can be seen as *micro data centers in a box* [3], are resource constrained, and are not able to keep a copy of all objects maintained in the cloud. Also, edge nodes are often more vulnerable to attacks by malicious players[14] and are not aimed at providing long-term reliable storage. Instead, they will typically keep copies of items that are primarily stored in the cloud. Because the capacity of edge nodes

is limited, providers of edge storage may be tempted to oversell their capacity and to hide this behavior by fetching, on-demand, data from the cloud or from other servers instead of serving it from local storage.

3.4 Service Level Agreement

In any case, regardless of the type of storage used, the client as some expectations regarding the properties of the service provided by the third party. These expectations are captured by a (implicit or explicit) *Service Level Agreement (SLA)*, that lists the requirements the service provider must comply with. In the context of third-party storage services, the SLA may cover the following aspects:

- **Integrity**: the stored documents have not been corrupted;
- **Retrievability**: the documents are retrievable;
- **Replication**: the provider keeps at least n replicas of an item, in independent storage servers/disks;
- **Geo-replication**: the provider keeps n replicas of an item, in different storage nodes, in different geographically distributed locations;
- **Timely Retrievability**: documents are retrieved within some (typically small) latency δ.

3.5 Addressing Rational Behavior

Like any local storage system, third-party storage can be affected by faults that may compromise data availability. Disks can malfunction and loose the data stored on them. A faulty controller can corrupt data when it is stored on disk. A malicious intruder may modify or delete stored data. Natural disasters may damage the storage equipment. These faults can be masked with the help of replication, by keeping multiple copies of each data item on different machines, ideally in different locations, using diverse hardware maintained by different teams, such that the chances of multiple correlated faults happen in a short time interval becomes small.

A significant difference between privately-owned storage and third-party storage is that, with the later, it may be difficult for the user to assess if the desired fault-tolerant and data placement policies are, in fact, deployed by the storage provider. In the absence of appropriate auditing mechanisms, the storage provider may have incentives to avoid deploying the desired redundancy levels, to save the costs associated with the additional storage and replica coordination.

As we have noted, in P2P storage, a node can simply not store the data it is supposed to store. When the data is requested, it may attempt to fetch it from another peer. Naturally, if other peers also do this, the chances that data is lost become high. This behavior is unlikely to occur in a cloud service provider, as it will be easily detected as soon as a client would attempt to access the data. However, the cloud provider may be tempted to keep less replicas of the data, in fewer locations, than the advertised. In turn, edge storage providers may have incentives to keep copies of data item in just a fraction of their fog nodes, to

augment the capacity they can sell to their customers, at the cost of providing increased latency to data users.

One way to avoid this type of misbehavior consists in deriving auditing mechanisms that can assess if the third-party storage service keeps the desired number of data copies in the desired locations. When the auditing detects a misbehavior, this can be used to penalize the provider. In a P2P system, a faulty node may be prevented from further using services provided by other nodes, cloud or edge storage providers may be forced to compensate clients for violations of the SLA.

4 Auditing Third-Party Storage Services

When a storage provider is subject to auditing, it must provide a proof that it is applying the data replication and data placement policies specified in the SLA. In the context of this report, such proof is generically called a *proof of storage* [15]. Because an SLA may cover different aspects of the storage implementation, such as the target number of replicas, the location of these replicas, or the latency observed by users when accessing data, it is possible to define different proofs of storage, each covering a different sub-set of the aspects enumerated above.

In this section we survey the main proofs of storage, and respective implementations, that have been proposed in the literature. First, we describe an overview of the different proofs of storage, and SLA aspects that they cover. Then, we describe the different auditing mechanisms and implementations for the introduced proofs. We have been inspired by the mechanisms used in these implementations for the construction of our *proof of timely-retrievability*.

4.1 Proofs of Storage

Each one of the following proofs of storage as been conceived to prove a different subset of the aspects that can be covered by a SLA, as described in Section 3.4.

- **Proof of Data Possession (PDP)** A proof of data possession proves integrity of the stored data;
- **Proof of Retrievability (PoRet)** A proof of retrievability proves documents are available to be downloaded;
- **Proof of Replication (PoRep)** A proof of replication proves that data copies are stored at different servers/storage devices;
- **Proof of Geographic Replication (PoGR)** A proof of geographic replication proves that data copies are stored at different geographic locations.

Although a storage node to build a PDP needs to have available the requested data items, the result of the proof may be a cryptographic hash over those data items, not proving that the auditor is able to download the stored data. Instead, with PoRet the proofs themselves return a part of the stored data, thus necessarily allowing to retrieve the stored data.

Table 1 matches the different proofs of storage with the quality of service aspects that they cover. As highlighted in the table, none of the proofs of storage previously proposed in the literature addresses timely retrievability.

	Integrity	Retrievability	Replication	Geo-replication	Timely Retrievability
PDP	✓	–	–	–	–
PoRet	✓	✓	–	–	–
PoRep	✓	–	✓	–	–
PoGR	✓	–	✓	✓	–

Table 1. Proofs of Storage vs SLA aspects

4.2 Obtaining Proofs of Storage

On a first approach, one might think that, in order to perform an audit, one could just read the objects stored and check if the provider is able to return their correct value. This simple method could, in fact, offer a proof of retrievability. Unfortunately, this method has several limitations. In first place, this approach is very expensive, as it requires the auditor to consume large amount of bandwidth to obtain the proof. In second place, this technique is unable to verify some of the QoS aspects, for instance, it cannot assess if the object is kept locally by the target or fetched on-demand from another location, nor is able to detect if the provider keeps just one or several replicas of the object. Therefore, more sophisticated approaches are needed, not only to save bandwidth, but to check the different aspects of the SLA.

Most audits are based on the notion of a *challenge*, a sequence of tasks that the storage providers need to execute in order to produce a proof, i.e., the content and the timeliness of the response to the challenge serves as a proof. Each tasks typically requires the provider to perform some cryptographic operation on the content of (a part of) a file, such that it can only obtain a correct and timely response if the data is stored locally. The challenge can be set-up in such a way that, in order to respond in a timely manner, the provider must perform several of the tasks in parallel, which in turn, requires the data to be replicated in different media. The number and sequence of tasks in a challenge depends on the type of proof the auditor aims at obtaining.

The challenge must guarantee that the computed proof of storage is *unique* (i.e, that the provider cannot re-use outputs from previous audits), *computed on-demand* (i.e., the provider cannot pre-compute an answer before the audits starts), and *authentic* (the proof is produced by the target storage node). It must also prove the required QoS aspects.

4.3 Components of a Proof of Storage Challenge

As a challenge is issued to obtain a proof of storage, a challenge is typically characterized by four main components: i) the set of objects that the nodes need to access; ii) the sequence by which these objects should be accessed; iii) the output that the node should produce (a function of the content of the objects accessed) and; iv) a deadline, by which the proof must be produced. We briefly discuss each one of these components next.

4.3.1 Set of Objects A challenge could force the storage node to access all objects that it is required to store. However, in most cases, this would be extremely expensive. Therefore, most challenges resort to sampling, i.e., to produce the proof the node only needs to access a subset of the data objects and, from these objects, it may also be required to access some sample blocks. Then, each challenge must select a different set of objects/blocks, using some pseudo-random function unknown to the node being audited, such that the node can not predict which objects it needs to maintain to answer a given challenge. The number of objects included in the challenge depends on different aspects, that will be discussed later in the report. In any case, this number should be selected such that the probability of producing a proof without having the entire set of objects is negligibly small.

4.3.2 Sequence of Objects In some cases, the challenger may want that the node being audited accesses the objects in a specific sequence, in particular, the challenger may want to prevent the audited node from knowing which is the next object in the sequence, unless it has already accessed the previous objects. To avoid excessive communication between the auditor and the node being audited, the identifiers of objects to be accessed, except the first object, may be sent encrypted. Then, it is possible to use the content of the i^{th} object to decrypt the identity of the $(i+1)^{th}$ in the sequence, such that the audited node is forced to respect the desired sequencing.

4.3.3 Cryptographic Hashes For each object or data block included in the challenge, the audited node may be forced to include the entire content in the proof of storage. Again, this can be very expensive, namely, it may consume a significant amount of bandwidth to transfer the proof. Due to this reason, most challenges require the audited node to send just a cryptographic hash of the accessed content. The auditor can then check if the hash that is produced as part of the proof matches the genuine content of the data. Naturally, it is important to ensure that the audited node has the original data, and not only a set of pre-computed hashes. For this purpose, the challenge include a random nonce η, that is unique and that cannot be guessed by the audited node. The hash to be returned as part of the proof is a function of the data stored and the nonce η, and can only be computed in response to the challenge.

4.3.4 Deadline Most challenges need to be answered by a given deadline, otherwise the audited node would have enough time to download, from another source, the data required to produce the proof. The deadline should be defined such that it can only be met if the audited node has the data, possibly stored in different servers. To define the right deadline for a challenge, the auditor needs to have models that characterize the delays involved in the production of the proof. These include the delays associated with reading data from persistent storage, computing hashes, and communicating with the auditor. As we will discuss, as

non-negligible part of defining a challenge, and a proof of storage is related with the correct definition of such models.

For the time to read a data object from disk, and because storage nodes may be heterogeneous (have different storage media), the auditor may bound this metric with estimation measurements from the storage media available in the market. The same approach could be used to estimate the time required to compute the cryptographic operations required to produce the proof.

Unfortunately, to measure experimentally the latency of the communication between the auditor and the audited node can be difficult without the participation of the target node, which can then introduce the artificial delays in the experiments to lead the auditor to over-estimate the communication latency which, in turn, would create a slack that the node could exploit to defeat the challenge. Therefore, this latency is often not obtained experimentally, but inferred from other parameters.

A technique that is often used consists in estimating the network latency from the physical location of the nodes involved in the communication [6]. In a first step, a metric of physical distance between the two nodes is obtained. Then, the physical distance is used to predict the network latency. These predictive models are built with the help of a set of storage nodes in known locations to the auditor, referred as *landmarks*.

The distance between two nodes in the network may be determine with the the following distance measurements [6]:

- **Haversine Distance**: distance between two points in a sphere;
- **Driving Distance**: distance between two cities, using Google Maps API;
- **Topology-based Distance**: distance between hosts in the Internet based on the network topology.

Then, the distance can be used to estimate the latency, using techniques such as [6]:

- **Speed of Light**: $\frac{4}{9}\times$(speed of light) as an upper bound for Internet latency;
- **Best Fit Line**: With the set of sampling measurements (distance, latency) between landmarks, use a linear regression function to obtain the best fit line that better accommodates these values;
- **Site Expected**: With latency measurements, for a given node, calculate the linear regression of these points.

However, this latency is a lower bound on the communication delay, as this delay has other components, such as the delay introduced by buffering routers.

4.4 Challenges Previously Proposed in the Literature

In the following sections, we describe different challenges that have been proposed in the literature.

4.4.1 Time Bounded Encoding Challenge *Benet et al.* [15] provide a time bounded encoding challenge to check data replication, i.e., to build a *proof of replication*.

To guarantee independent storage of n replicas of data D, before an audit, each storage node encodes the data replica stored locally with a different key, creating n encoded replicas between all nodes. Then, when subject to an audit, each storage node has to prove compliance with a given encoding, leading all nodes together to prove local storage of n unique replicas. To avoid a storage node to resort to a neighbor node to fetch the replicas that are discarded, or to re-encode them from D, the proof of replication must be produced on time. A proof of replication issued by a storage node is only valid if it is correct, i.e., proves n replicas, and it was built within a time limit. The time limit is set as the network delay plus the delay to retrieve the already encoded replicas. In case the n replicas are not stored locally, or the storage node needs to re-compute them by re-encoding D, the extra network or computation delay will exceed the time limit, and the proof will be considered invalid. Therefore, the encoding function is the main point to detect a rational cloud storage provider. The encoding function must be slow enough, to detect re-encoding on-demand, by the target node or by a neighbor one. *Benet et al.* set block cipher with block chaining and layering as the more accurate encoding function: it provides sequential encryption, where each cipher block in each iteration depends on the ciphered blocks from previous ones. By using this function, the auditor guarantees that data encryption can not be parallelized.

Benet et al. ensure each proof is unique, and computed on-demand by forcing each storage node to commit to a specific encoding, and authentic due to proof computation within a time limit.

4.4.2 Parallel and Sequential Time Bounded Challenge *Li et al.* [4] introduce a parallel and sequential time bounded challenge to build a proof of physical reliability (PoPR). A PoPR is equivalent to *proof of replication*, as proves data replication in independent storage. At the same time, a PoPR may be extended to prove data geo-replication.

Files are broken into ns subfiles and encoded with an error correction code (ECC), producing a verifiable version, which is later subdivided in n l-bit blocks, referred as *symbols*. Later, symbols are distributed across the cloud storage provider n storage nodes, defining a symbols layout, that must be followed by the cloud storage provider. Because each storage node will keep one symbol of each subfile, when retrieving a document, it is required to access all storage nodes. Symbols in the same storage node are retrieved serially, and in parallel to other nodes, and hashed before being returned to the auditor.

Once a rational cloud storage provider may store symbols in fewer nodes than the defined in the layout, when retrieving a document to build a PoPR, more symbols need to be read serially than in parallel, leading documents retrieval to take longer than expected, and allowing the auditor to detect misbehavior. Therefore, as each storage node is challenged in parallel, the auditor sets the time

limit as the delay to read from disk the symbols expected to be stored locally in that node, plus the computation delay to build the proof. Because storage nodes may be heterogeneous, the read delay is set between loose boundaries: the upper bound is the fastest data access rate known so far plus an error margin, and the lower bound the lowest reading delay registered from that node, plus a margin. These loose boundaries guarantee that the reading delay is independent from the storage nodes hardware, and that a storage node with fast disk access, if it misbehaves, it is detected.

In addition, as each storage node will have a different response deadline due to the different number of symbols to be retrieved, when issuing a challenge, the auditor must mask the challenge length. Otherwise, the cloud storage provider detects what is the smallest challenge, and process it first; and just then process the larger ones. Because the larger challenges will have a larger deadline, even if the cloud storage provider processes the smallest challenge first, it could compute valid proofs (correct and on time) for both challenges. Therefore, the challenge length must be hidden, which can be achieved by creating a dependency between the symbols to be retrieved (the next one to be retrieved is only known after the previous one is retrieved) and by hashing the last symbol to be retrieved. A challenge is only finished when the hash of the retrieved symbol matches the hash of the last one. With this method and by choosing pseudo-randomly the symbols to be retrieved, the auditing entity guarantees that each built PoPR is unique and computed on-demand.

To prove data geo-replication and increase proof authenticity, the above mechanisms may be applied to storage nodes in different known geolocations. If a response to the challenge is not received within the deadline, it is assumed that the clients files are not kept locally, and then the data is not in that location.

4.4.3 Data Geolocation and Time Bounded Challenge *Benson et al.* [6] introduce a geolocation and time bounded challenge to obtain a *proof of geographic replication*.

As each data block is stored together with an authentication tag, storage nodes are challenged to prove data storage of both elements in less than T, where T is the execution time of some audit protocol plus the network delay. To guarantee proof authenticity, T must be lower enough to ensure that the target storage node does not reach to a neighbor one that will answer the challenge, i.e., the challenged storage node must be the one computing the proof. At the same time, as each challenge requests retrievability of as much random blocks that can be retrieved in T, any extra network delay to reach a neighbor node will exceed the time limit. With blocks being randomly selected, this challenge ensures that the proof that is built is unique.

On the other hand, to assess data geo-replication, the auditor, with a set of landmarks, builds a latency-distance predictive model (Section 4.3.4), that is later applied to storage nodes in unknown locations. *Benson et al.* discuss the three models, previously introduced, with Haversine and Driving Distance, and according to them, and as expected the Best Fit Line and Site Expected

models with Haversine distance outperform the Speed of Light model, when retrieving content from 40 universities hosts in North America. The $\frac{4}{9}$ the speed of light is a to large upper bound for latency, leading to inaccurate distances and geolocations.

However, as these predictive models require a set of landmarks, *Benson et al.* extend their approach by providing an IP analysis of the retrieved content as a way to detect data centers geolocation. If for a set of data requests the storage nodes' IPs are geo located to the same region, then, most likely in that region there is a data center.

4.4.4 Constraint-based Data Geolocation Challenge *Gondree et al.* [5] provide a constraint-based data geolocation (CBDG) challenge that combines a *proof of data possession* with constraint-based geolocation [16] to build a *proof of data geographic replication*.

Documents are broken into n blocks and tagged with a message authentication code (MAC), and later stored at cloud storage provider's storage nodes. Before executing an audit, the auditor to assess data integrity and ensure the computed proof is unique, selects c random indices referring the stored blocks and tags to be retrieved. At the same time, the auditor requests a set of trusted landmarks to build a latency-distance predictive model between them that they will later apply. Due to its simplicity, *Gondree et al.* considered the Best Fit Line model, from Section 4.3.4, has the latency-distance predictive model.

Next, when executing an audit, the auditor forwards a challenge (the c random indices) to the landmarks that will, in turn, challenge the target storage node. For each challenge, each landmark will record the delay response time, and together with the pre-built predictive model estimate its distance d to the target storage node. Then, with distance d, each landmark computes the circular constraint region, with radius r that forwards to the auditor, together with the storage node response to the challenge.

With the radius r from all landmarks, and the responses, the auditor checks responses correctness, and if they are correct, computes the intersection between the different circular regions. This intersection will result in the most feasible region for the storage node geolocation. Therefore, this method allows to geo locate storage nodes in unknown locations.

4.5 Discussion

In this section, we summarize and analyze the different challenges proposed in the literature, to help us to understand how we can design an accurate challenge to obtain our *proof of timely-retrievability*. We begin by providing a table, Table 2, and then we discuss how the literature satisfies the previously introduced challenge components.

Table 2 provides an overview of the proofs of storage provided by each scheme in the literature. As the described challenges are applied to a cloud storage environment, that, at the same time, is unable to satisfy data users requests with low latency, none of the previous works prove data timely retrievability.

Challenges	PDP	PoRet	PoRep	PoGR
Time bounded encoding[15]	✓	✓	✓	–
Parallel and sequential time bounded[4]	✓	–	✓	–
Data geolocation and time bounded[6]	✓	–	✓	✓
Constraint-based data geolocation[5]	✓	–	✓	✓

Table 2. Proofs of storage provided by each each challenge in the literature.

We now analyze how the schemes provided in the literature guarantee each one of the components of a proof of storage challenge, previously introduced in Section 4.3.

- **Set of Objects:** In [15], as the cloud storage provider must prove storage of the n data replicas, storage nodes are forced to retrieved all data items. However, as in [4, 6, 5] the auditor selects a random number of data blocks to retrieve, a challenge will only address a sample of the remotely stored data. Nevertheless, in the three cases the number of challenged data blocks will determine the probability to detect a rational cloud storage provider: larger the challenge, longer the storage nodes will take to respond. Then, a proof will only be computed on-time if the accessed data items are kept in local storage.
- **Sequence of Objects:** To assess data storage in fewer nodes than the agreed, in [4], data blocks in the same node are retrieved serially, and in parallel to other nodes. If data is kept in fewer nodes, more data items need to be read serially, and therefore the issued proof will exceed the deadline. In this work, the number of data items to be retrieved is also masked, by identifying the next data item to retrieve from the previous one.
- **Cryptographic Hashes:** Because in [15] the auditor requests for the data replicas, and in [6, 5] the storage nodes have to answer the data items, together with their tags, this audits bring extra overhead to the network, and are more expensive. Therefore,the hash computed by the storage nodes in [4] is a more accurate solution, as it is cheaper, and does not overload the network, at the same time it allows to assess data integrity.
- **Deadline:** As [15, 4] address storage nodes in known locations, the auditors in both challenges set the deadline as a correlation between the size of data to be retrieved, and the time to compute the proof, together with an upper bound estimate for the network delay. However, in [6] and [5], because storage nodes may be in unknown locations, and challenges aim to assess data geolocation, the auditor estimate a more accurate network delay from a set of predictive latency models (previously introduced in Section 4.3.4).

5 Architecture

We now address the problem of building proofs of storage for edge computing settings. We assume a system, illustrated in Fig. 1, where a set of fog nodes are

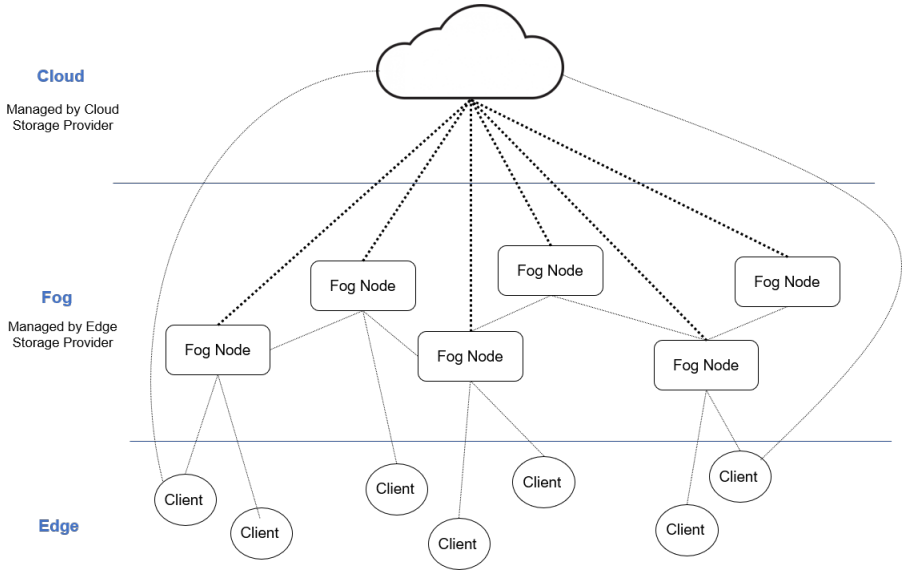

Fig. 1. Communication and auditing (bold) channels.

requested to locally store documents, that are also available in the cloud, such that they can serve local clients with low latency. For instance, a given fog node could be requested to store information regarding maps and points of interest in its geographical region. We assume that the edge nodes are managed by some *edge storage service provider* (ESSP) and that the objects that need to be served by each fog node with low latency are specified in the SLA.

If the ESSP is rational, it may opt not to store all objects in the fog nodes as requested, and fetch documents from the cloud, or from other fog nodes in order to serve clients. In this case, clients will observe latencies larger than those specified by the SLA, defeating the purpose of using edge storage. Our aim is to design a set of mechanisms to assess if a fog node can retrieve the objects it is supposed to store fast enough.

5.1 Proof of Timely-Retrievability

To address the problem above, we define a *proof of timely-retrievability*, a proof that a fog node produces to demonstrate that it is able to serve objects with low latency. More precisely:

> ***Proof of Timely-Retrievability (PoTR)*** A *proof of timely-retrievability* proves documents retrieval within some (typically small) latency δ.

As δ reflects the maximum time for a fog node read a data object from disk to answer a user data request with low latency, and as edge services and applications require response times between 5–30 milliseconds, in our work, we define δ in the magnitude of milliseconds.

5.2 Assumptions

In our work, we assume that the cloud storage provider is trustworthy to the client, and that an edge storage provider may misbehave to save resources costs.

As the data outsourced to the edge storage provider is managed, in the first place, by the cloud storage provider, we assume that the cloud storage provider knows the data expected to be stored locally in each fog node, but it is the edge storage provider the entity responsible to spread the data items to the diverse fog nodes. We also assume that each fog node has a processor with the Intel SGX extension: we will use guarantees provided by a trusted execution environment to support the execution of the challenge on the fog node. A trusted execution environment, also referred as *enclave*, is an isolated environment that guarantees code, and data integrity, and confidentiality [14, 17]. However, in this report, we did not consider the use of Intel SGX extension in detail, as we focused on how to obtain our propose proof of timely-retrievability.

Finally, we assume that is not economically for the ESSP to download all objects to a fog node at the beginning of the audit, i.e., that the cost of downloading all objects every time there is an audit would exceed the costs of maintaining these objects locally.

5.3 Placement of the Auditor

In our work, we have opted to run the audit from a node located in the cloud. This has advantages and disadvantages. The main advantage of this option is that it simplifies the deployment. A single auditor can be placed in a central location and be used to extract proofs of timely-retrievability from multiple fog nodes. The main disadvantage is that is that the communication between the auditor and the fog node being audited is subject to variable delays, and the challenge must take this variance into account when selecting the number of objects to be sampled and the deadline for producing the proof. Fig. 1 represents in bold the auditing channels, between the cloud storage provider and the fog nodes, together with the remaining communication channels existing in the network (fog node—fog node; cloud—client; fog node—client).

As an alternative, the auditor could be deployed in specific clients, deployed on purpose for the effect. A significant advantage of this option is that the auditor could use exactly the same network as the data clients, and obtain a more accurate estimate of the delays observed by other clients. However, this would require to deploy at least one auditor node for each Fog client, which could be impractical.

A third alternative we could delegate auditing capabilities to the clients owned by end users. In this case, the data users would collect information regarding the observed latencies when accessing data. This data could be collected later on for processing in order to assess the fog node compliance with the SLA. We have disregarded this option due to the significant privacy issues associated with collecting data access and location information from end users.

5.4 Challenge

We now describe the challenge that we have designed, to be used by an auditor placed in the cloud. The challenge requires the fog node to access a number of objects in sequence and return within a specified deadline. In the following we describe how we setup the different parameters of the challenge.

Set of Objects The auditor selects the number of documents subsets N to be retrieved from each fog node. Then, with a pseudo-random function selects a sequence of N indexes, each one referring to a document subset (a *data block*) expected to be locally stored at the target fog node.

As N reflects the number of data blocks to be retrieved, and higher the value N, longer the fog node will take to read those data blocks from disk. The auditor must choose an N value that allows to detect the fog node access time to disk, and reduce error margins, and time variances from network communications.

Sequence of Objects The auditor before sending the challenge to the fog node enclave arbitrarily orders the list of selected N indexes. This method will increase the probability to detect if the target fog node keeps data blocks in local storage, as the node is forced to do random jumps when accessing the disk. Case a part of the data is kept remotely, this random jumps most likely will fall into an unavailable data block, requiring it to be downloaded on-demand.

To ensure the proof is executed in the desired node, the auditor forwards the list of randomly ordered indexes to the trusted execution environment of the fog node. However, because data items are kept outside the enclave, due to its limited storage capacity, each data block to be read requires the fog node processor to change its execution mode (from secure to normal, and vice-versa)[14] repeatedly. The enclave (secure mode) requests the fog node (normal mode) to read and compute a hash of a data block, at a time. The index of the next data block to be retrieved is only provided to the fog node, after it has returned to the enclave the previous hashed block. This process is repeated until all required data blocks have been retrieved, and hashed, i.e., until the N data blocks have been serially read from disk. The fog node must retrieve data blocks serially from disk, and not in parallel, to signal to the the auditor the delay the target fog node takes to retrieve a data block.

At the same time, because the fog node needs to resort to the enclave to known the next data block, we guarantee that our proof of timely-retrievability is authentic, as it can not be computed by a neighbor fog node, or in the cloud. If we considered the scheme introduced in Section 4.3.2, where the identity of the following data block cloud be inferred from the hash of the previous one, we would not guarantee that the proof was authentic. The fog node could request a neighbor node, or even the cloud to compute the proof.

Cryptographic Hashes With each retrieved data block, the fog node computes an hash, together with a nonce η (previously chosen by the auditor and provided to the fog node through the enclave). The nonce, unique per fog node, not only

guarantees that the issued proof is unique, as it guarantees it is computed on-demand (a valid proof can not be computed without η).

In turn, the enclave as it receives the hashed data blocks with the nonce keeps them in memory. At the end, after all required blocks have been read and hashed, the enclave computes a new hash of all blocks that will forward, as a response, to the auditor.

Before checking proof correctness, for each challenged fog node, the auditor computes the same sequence of data blocks hashes, with the respective nonce, building a verifiable version. When receives the response from the enclave, compares the obtained proof with the computed verifiable version, if both computations match, the issued proof is correct.

Deadline The auditor defines a deadline T that each challenged fog node has to comply to build a on-time proof of timely-retrievability. As each fog node can only satisfy users data requests with low latency, if the time it takes to retrieve a data object from disk is at maximum δ milliseconds, the auditor must guarantee that the deadline T is only met if the fog node retrieves each one of the requested data blocks, in the challenge, at maximum δ milliseconds. If the fog node does not keep data blocks in local storage, and needs to fetch them from a neighbor node, or from the cloud, this maximum retrieval time δ will not be respected, leading to an invalid proof of timely-retrievability (the proof is issued over the time limit).

At the same time, as for each one of the N requested data blocks to be retrieved the fog node needs to compute an hash, together with the previously provided nonce η, this computation delay will influence the fog node response time to the challenge. So, the auditor, when defining the deadline T, needs also to account with the delay to compute the hash of the N data blocks. We define α as an estimate value of the computation time of a data block hash.

Because an audit requires the auditor to send the challenge to the fog node, and the fog node to answer back the computed proof, when defining the deadline T, the auditor needs to account with the delay the messages take in the network. Therefore, being d an estimate value for the network delay in one direction communication, the auditor accounts for the deadline the network delay of both directions communications ($2d$).

As this being said, the auditor defines T as the sum of the communication network delay ($2d$) plus the delay to retrieve and compute the hash of the selected N data blocks ($N(\delta+\alpha)$). Once the auditor aims to prove the data timely retrievability, the delay to retrieve a data block is set as δ, the maximum delay accepted to provide data to users with low latency. Nevertheless, as communication delays, accesses to disk, and hashes computation may be subjected to variances and error margins, we add estimate margins to the deadline T: ε^d, ε^δ, and ε^α that correspond to error margins of network delay, access time to disk, and hash computation time, respectively. Equation 1 provides an overview of the described deadline T.

$$T = (2(d \pm \varepsilon^d)) + N \times ((\alpha \pm \varepsilon^\alpha) + (\delta \pm \varepsilon^\delta)) \tag{1}$$

As ε^{α} and ε^{δ} may be negligibly small, the same does not happen with ε^{d}. As the network is frequently unstable, and the auditor may be distant to the target fog nodes, the network delay and respective error margin (ε^{d}) may be over-estimated, providing a slack for the fog nodes to misbehave without being detected. Therefore, as a countermeasure, the auditor must guarantee that the number N of requested data blocks is sufficiently high to attenuate the network delay variances. The auditor must guarantee that the time the fog node takes to compute a proof is mainly dependent on the time to retrieve, and hash the N requested data blocks. Nevertheless, as ε^{α}, and ε^{δ} are present when processing a data block, these errors do not attenuate with the increase of N.

Being t_c the time a fog node takes to answer a challenge; d_c the network delay verified in the challenge; α_c the hash computation delay during a challenge; and δ_c the time a fog node takes to retrieve a data block, when subjected to a challenge; t_c, together with the considered error margins, may be described as in Equation 2.

$$t_c = (2(d_c \pm \varepsilon^{d})) + N \times ((\alpha_c \pm \varepsilon^{\alpha}) + (\delta_c \pm \varepsilon^{\delta})) \tag{2}$$

From Equation 2, we can obtain the delay (δ_c) a fog node takes to read a data block (Equations 3 and 4).

$$\delta_c = \frac{t_c - 2d_c \pm 2\varepsilon^{d} - N\alpha_c \pm N\varepsilon^{\alpha} \pm N\varepsilon^{\delta}}{N} \tag{3}$$

$$\delta_c = \frac{t_c - 2d_c - N\alpha_c}{N} \pm (\frac{2\varepsilon^{d}}{N} + \varepsilon^{\alpha} + \varepsilon^{\delta}) \tag{4}$$

Where, in Equation 4, $(\frac{2\varepsilon^{d}}{N} + \varepsilon^{\alpha} + \varepsilon^{\delta})$ captures the error ε^{c} in estimating δ_c the challenge is subject to (Equation 5).

$$\varepsilon^{c} = \frac{2\varepsilon^{d}}{N} + \varepsilon^{\alpha} + \varepsilon^{\delta} \tag{5}$$

As the number N of data blocks to be retrieve increases, the estimation error ε^{c} of the reading delay δ_c is subject to decrease. So, the auditor must select a value N to ensure the reading delay has the lower possible error margin (Equation 6). Thus, the minimum value for N may be determine by the auditor from Equation 7.

$$N = \frac{2\varepsilon^{d}}{\varepsilon^{c} - \varepsilon^{\alpha} - \varepsilon^{\delta}} \tag{6}$$

$$N > \frac{2\varepsilon^{d}}{\varepsilon^{c}} \tag{7}$$

Then, with a minimum number N of data blocks, the auditor guarantees that the time to retrieve a data block is larger than the error margins, it is subject to (Equation 8), that will allow a auditor to determine whether a fog node is able to retrieve a data block in $\delta_c \leq \delta$, or not.

$$\varepsilon^{c} << \delta_c \tag{8}$$

We now describe a more concrete example, on how the auditor may determine the number of minimum data blocks to be read, to assess the delay to retrieve each one of them.

For an maximum expected δ of 30 milliseconds, and a desired estimation error of less than 3 milliseconds ($\varepsilon^c < 3ms$), and a network delay variance of 100 ms ($\varepsilon^d = 100ms$) [18], the auditor must guarantee that:

$$\frac{2 \times 100}{N} ms < 3ms \tag{9}$$

$$\frac{2 \times 100}{3} \approx 70 < N \tag{10}$$

So, the auditor must select at minimum 70 data blocks to be retrieved.

In summary, a fog node to prove that it is able to satisfy users data requests with low latency, must build a valid proof of timely-retrievability. A proof of timely-retrievability is valid if the computed final hash representing N retrieved data blocks, together with the nonce, is correct, and the proof is issued in $t_c \leq T$.

As the fog node is challenged to retrieve each one of the N data blocks at maximum in δ milliseconds. If the delay to retrieve each data block (δ_c) is longer than δ milliseconds, the proof will be over-time, and so invalid. An invalid proof will, in turn, alert the auditor that the agreed service is not being complied by the ESSP. At the same time, a fog node that reads each challenged data block in $\delta_c \leq \delta$ milliseconds, will provide a valid proof of timely-retrievability, proving compliance with the desired QoS aspect: data timely retrievability, which means that is able to respond to users data requests with low latency.

5.5 Optimization

As the network delay brings highly variance to the deadline, and may provide a space for a fog node to misbehave without being detected, the removal of this parameter could improve audit confidence. Therefore, a possible way to achieve this is by having the enclave, which is a trusted part to the cloud storage provider, measuring the delay the fog node takes to build a proof. This will not just reduce the estimation error in time measurements, as it will allow to reduce the number N of data blocks to be retrieved (saving resources), as network variances are no longer taken into account. Nevertheless, this approach requires the Intel SGX extension to read the system clock, which is still an open issue in the literature.

6 Evaluation

Our main goal is to evaluate the accuracy of our proposed challenged, and respective proof of timely-retrievability to detect rational edge storage service providers. More precisely, we will evaluate the true-positive rate (how often the challenge successfully flags as faulty a misbehaving fog node) and the false-positive rate (how often the challenge unfairly marks as faulty a correct fog node) for different system scenarios. We will build the scenarios by varying different

parameters, namely: i) the actual average reading delay δ_c a fog node takes to retrieve a data block; ii) the variance of the network delay (between the auditor and the fog node); and iii) the selected number N of data blocks to be retrieved during a challenge.

6.1 Reading Delay

Let δ be the reading delay that needs to be observed by a fog node that complies with the SLA. A faulty fog node will suffer an average delay $\delta^{faulty} > \delta$. Naturally, the larger the difference between the desired delay δ and the actual delay δ^{faulty}, the easier should be to detect the misbehavior. We plan to experiment with different values of δ^{faulty}, to assess the sensibility of our challenge to this parameter.

Note that if the fog not is faulty, the value of δ^{faulty} will depend on the fraction of objects that are not stored on the fog node and also on the location used to fetch these object on demand. Thus, different behaviors of a rational fog node can lead to the same value of δ^{faulty}. We also plan to assess if these factors have an impact on the accuracy of our challenge.

6.2 Network Delay Variance

Because the network is frequently unstable, and the auditor may be distant to the challenged fog nodes, the auditor will verify a higher network variance when measuring the time a fog node takes to answer to a challenge. This high variance will, in turn, interfere with an accurate measurement of the time a node takes to read a data block. As the number N of requested data blocks may overcome this situation, we will evaluate how the network variance influences readings measurements, and whether according to the auditor/fog nodes geolocation, the number N of requested data blocks needs to be changed (increased, or decreased).

6.3 Number of Challenged Data Blocks

Equation 7 provides a method to derive the number of objects N that should be included in the challenge, as a function of the target estimation error and the known errors in the network, disk access and computing delays. We plan to run challenges using different values of N, including values smaller and larger than the value suggested by Equation 7, to understand if this formula is the most appropriate.

7 Scheduling of Future Work

Future work is scheduled as follows:

– January 9 - March 29: Detailed design and implementation of the proposed architecture, including preliminary tests.

– March 30 - May 3: Perform the complete experimental evaluation of the results.
– May 4 - May 23: Write a paper describing the project.
– May 24 - June 15: Finish the writing of the dissertation.
– June 15 Deliver the MSc dissertation.

8 Conclusions

As third-party storage systems may be affected by faults that compromise data availability, data must be replicated across different storage nodes. However, as this requires extra costs to deploy and manage data replicas, storage providers may rationally misbehave by keeping data items in fewer storage nodes than the required by the client. Therefore, storage providers, and their storage nodes must be subjected to audits to detect misbehavior, and non compliance with an agreed service. Apart from data replication, third-party storage providers may not comply with data integrity, data retrievability, geo-replication and timely retrievability.

One of the characteristics a cloud storage provider can not comply is data timely retrievability (documents are retrieved within a typically small latency), due mainly to network distance of data centers to users. Therefore, a cloud storage provider may resort to a edge storage provider to place copies of users' data in storage nodes (fog nodes) closer to them. But as edge storage providers have a limited storage capacity, they may be tempted to rationally oversell their capacity and to hide this behavior by fetching, on-demand, data from the cloud or from other servers instead of serving it from local storage, at the cost of serving users with high latencies.

In this work, we provide a challenge as an auditing mechanism from which a proof of timely-retrievability can be obtained. Our proof of timely-retrievability allows a trustworthy cloud storage provider to assess service compliance by a hired edge storage service provider (ESSP). The auditor challenges fog nodes, managed by the ESSP, to prove they are able to retrieve a set of data blocks within milliseconds, as a way to guarantee data clients are satisfied with low latency.

Acknowledgments We are grateful to Cláudio Correia for the fruitful discussions and comments during the preparation of this report. This work was partially supported by the FCT via project COSMOS (via the OE with ref. PTDC/EEI-COM/29271/2017 and via the "Programa Operacional Regional de Lisboa na sua componente FEDER" with ref. Lisboa-01-0145-FEDER-029271) and project UIDB/ 50021/ 2020.

References

1. Benet, J.: Ipfs: Content addressed, versioned, P2P file system. arXiv preprint arXiv:1407.3561 (2014)

2. Swarm: Swarm: Storage and communication for a sovereign digital society (2019)
3. Taleb, T., Samdanis, K., Mada, B., Flinck, H., Dutta, S., Sabella, D.: On multi-access edge computing: A survey of the emerging 5g network edge cloud architecture and orchestration. IEEE Communications Surveys Tutorials **19**(3) (2017)
4. Li, L., Lazos, L.: Proofs of physical reliability for cloud storage systems. IEEE Transactions on Parallel and Distributed Systems **31**(5) (2020)
5. Gondree, M., Peterson, Z.N.: Geolocation of data in the cloud. In: Proceedings of the Third ACM Conference on Data and Application Security and Privacy, San Antonio, Texas, USA (2013)
6. Benson, K., Dowsley, R., Shacham, H.: Do you know where your cloud files are? In: Proceedings of the 3rd ACM Workshop on Cloud Computing Security Workshop, Chicago, Illinois, USA (2011)
7. Ramaswamy, L., Liu, L.: Free riding: a new challenge to peer-to-peer file sharing systems. In: 36th Annual Hawaii International Conference on System Sciences, 2003. Proceedings of the. (2003)
8. Feldman, M., Chuang, J.: Overcoming free-riding behavior in peer-to-peer systems. SIGecom Exch. **5**(4) (jul 2005) 41–50
9. Andrade, N., Brasileiro, F., Cirne, W., Mowbray, M.: Discouraging free riding in a peer-to-peer cpu-sharing grid. In: Proceedings. 13th IEEE International Symposium on High performance Distributed Computing, 2004. (2004)
10. Streiffer, C., Srivastava, A., Orlikowski, V., Velasco, Y., Martin, V., Raval, N., Machanavajjhala, A., Cox, L.: eprivateeye: To the edge and beyond! In: Proceedings of the Second ACM/IEEE Symposium on Edge Computing (SEC), San Jose, California, ACM (2017)
11. Drolia, U., Guo, K., Tan, J., Gandhi, R., Narasimhan, P.: Cachier: Edge-caching for recognition applications. In: Proc. of the 37th Inter'l Conf. on Distributed Computing Systems (ICDCS), Atlanta (GA), USA (June 2017) 276–286
12. Satyanarayanan, M., Gibbons, P., Mummert, L., Pillai, P., Simoens, P., Sukthankar, R.: Cloudlet-based just-in-time indexing of iot video. In: Proceedings of the Global Internet of Things Summit (GIoTS), Geneva, Switzerland, IEEE (2017)
13. Ricart, G.: A city edge cloud with its economic and technical considerations. In: Proc. of the 1st Inter'l Workshop on Smart Edge Computing and Networking, Kona (HI), USA, IEEE (2017)
14. Correia, C.: Omega: a secure event ordering service for the edge. Master's thesis, Instituto Superior Técnico, Universidade de Lisboa (nov 2019)
15. Benet, J., Dalrymple, D., Greco, N.: Proof of replication. Protocol Labs, July **27** (2017) 20
16. Gueye, B., Ziviani, A., Crovella, M., Fdida, S.: Constraint-based geolocation of internet hosts. IEEE/ACM Transactions on Networking **14**(6) (2006)
17. Ekberg, J., Kostiainen, K., Asokan, N.: The untapped potential of trusted execution environments on mobile devices. IEEE Security Privacy (2014)
18. Aikat, J., Kaur, J., Smith, F., Jeffay, K.: Variability in tcp round-trip times. In: Proceedings of the 3rd ACM SIGCOMM Conference on Internet Measurement. IMC '03, Miami Beach, FL, USA, Association for Computing Machinery (2003)