Proof of Timely-Retrievability for Storage Systems at the Edge

(extended abstract of the MSc dissertation)

Rita Alexandra Rodrigues Prates
Departamento de Engenharia Informática
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 used to store files such that clients can access data 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 with the required low latency. In this thesis, we study techniques that have been proposed in the literature that can help in detecting this type of misbehavior and design and implement a new proof of storage to detect this rational behavior. Our *Proof of Timely-Retrievability* (PoTR) aims to assess whether a storage node at the edge is able to retrieve a data object with a latency smaller than some specified threshold δ. We leverage the availability of *Trusted Execution Environments* (TEE), more precisely *Intel SGX*, to ensure that the proof is produced by the fog node being audited, and to reduce the communication between the auditor and the audited node. Our proof is issued in response to a challenge: we describe how a challenge may be designed and configured, so that a proof of timely-retrievability is obtained. We have evaluated our design experimentally, and we show that the auditor is able to distinguish a fog node that respects the target limit on data access latency δ from a fog node that does not.

1 Introduction

Today, there are many scenarios where an end-user, or an organization stores data in machines run by third-parties, either to ensure durability and availability, or to provide others with low latency when accessing the data. Relevant examples include cloud storage (such as Dropbox, iClould, Google Drive, and many others), peer-to-peer storage (including the storage services used for decentralized applications [4]), content distribution networks (for instance, Akamai), and, more recently, edge storage [21]. Edge computing is a distributed model that places servers with computing resources physically close to end-users so that applications that require low latency processing and storage resources can be executed. Moreover, servers can only provide low-latency to these applications if they are able to access stored data within a constrained delay.

A user, when using a storage service, has some expectations regarding the quality of service to be provided, and agreed with the storage third-party. Some of those expectations include the guarantee that the third-party will not discard or corrupt the stored data, that the data is stored in distinct machines, in specific geographic locations, and that data clients are served with some bounded delay. Unfortunately, a rational [6, 11, 14] third-party may opt to avoid complying with the agreement, if it can gain some benefit and pass unnoticed. For instance, the third-party can keep the data in fewer locations than agreed with the customer, assuming that it can be impossible for the customer to audit how many replicas are used, or where these replicas are located. This behavior 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 defined quality of service.

In this thesis, 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. We propose a new proof of storage: a *Proof of Timely-Retrievability* (PoTR) that leverages the existence of Trusted Execution Environments (TEE), more precisely *Intel SGX*, to ensure that the challenge is executed by fog node being audited [15], and not by some other fog node, and to avoid revealing the data to be accessed during an audit, prematurely. In this thesis, we equally provide an extensive experimental evaluation to assess the accuracy of the produced proof.

2 Background and Related Work

2.1 Data Storage on Third-parties

There are several scenarios where a user may opt to store data on machines controlled by a third-party. The main three scenarios are peer-to-peer (P2P) 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 requirements agreed.

2.1.1 Edge Storage. One of the main motivation for edge storage is the ability to serve data clients with low-latency. Applications for image processing of faces, or object recognition [9, 20], and just-in-time video indexing [19] require a response time below 5–30 milliseconds [18], something that cannot be guaranteed with cloud storage alone. Edge storage consist on placing computing and storage resources close to end-users to avoid long network delays. Thus, edge storage is able to meet the low latency of these new applications. Edge servers, also known as *fog nodes*, or *cloudlets* complement edge storage services. Nevertheless, these servers are resource constrained, and are not able to keep a copy of all objects maintained in the cloud [13]. Instead, they will typically keep copies of items that are primarily stored in the cloud. Moreover, due to their limited storage capacity, providers of edge storage may be tempted to oversell their storage and hide this behavior by fetching, on-demand, data from the cloud or from other servers, instead of serving it from local storage. At the same time, due to servers geographic distribution, they are more vulnerable to attacks by malicious players than cloud storage datacenters [7].

2.1.2 Addressing Rational Behavior. 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 [6, 8, 11, 14]. Specially, 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, and detect this rational behavior 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. For example, in a P2P system, a faulty node may be prevented from further using services provided by other nodes [10], cloud or edge storage providers may be forced to compensate clients for violations of the defined contract, generically known as *Service Level Agreement* (SLA).

2.2 Auditing Third-Party Storage Services

When a storage provider is subject to an audit, it has to provide a proof that it is applying the data replication and data placement policies specified in the SLA. Such proof is generically called a *proof of storage* [5]. Moreover, as an SLA may cover different aspects of storage implementation, such as the target number of replicas, the location of those replicas, or the latency observed by users when accessing data, it is possible to define different proofs of storage.

2.2.1 Proofs of Storage. The literature is rich in techniques that allow to obtain different proofs of storage, that assess different properties of the provided service. The most relevant ones are *Proofs of Data Possession* (PDP) [3] and *Proofs of Retrievability* [2] that aim to check if the storage provider keeps at least some copies of the stored data; *Proofs of Replication* [5, 14] that assess if the storage provider has n copies of a data item (even though they might all be placed in the same machine); and *Proofs of Geographic Replication* [6, 11] that test if the storage provider keeps n data copies in distinct machines, in distinct geographic locations. Each one of these proofs is issued as a response to a *challenge* [14], that is sent to the storage provider by one, or more auditing entities. Typically, a challenge requires the storage provider to execute a set of reading operations over a subset of the stored data, and return on-time a value that proofs the access of the correct data items. This return value, may be an cryptographic hash of the retreived data items.

2.2.2 Structure of a Challenge. A simple way to verify if a storage provider keeps a given data item would be to request that item, and check the item integrity. Although this method could, in fact, offer a PoRet, this method has several limitations.

In first place, this approach is very expensive, as it requires the auditor to consume a large amount of bandwidth to obtain the proof. Therefore, to save bandwidth most challenges require the storage provider to read a subset of the stored data items, and compute a cryptographic hash of them. Typically, the response has to be issued within a pre-defined deadline, determine by the auditor [5, 6, 8, 14]. Furthermore, when sending the challenge, the list of data items to be access should not be revealed entirely. Instead, the list should be interactively revealed, i.e., the next data item to be accessed, it is just known after accessing the previous one [14]. This interactivity prevents the storage provider to download the missing data items on-demand. If the storage provider cannot guess in advance the data items to be accessed, it has to keep all data items within the constraint access time, to build a correct and timely proof.

In second place, the single request for data items is unable to verify some of the service requirements. For instance, it cannot assess if the storage provider keeps just one, or several replicas of the data items. Therefore, some auditors may force the storage provider to keep multiple copies of the same file, each one encoded with a distinct key [5], unknown to the storage provider. In practice, each replica has to be treated as a distinct file, that has to be stored by the provider. Unfortunately, this mechanism, does not avoid a provider to keep all replicas in the same machine, which may compromise replicas availability in case of failure. To ensure replicas are kept in distinct machines, the challenge may be designed in such a way that it is impossible to respond on-time if all replicas are kept in the same machine, although it may be feasible if the proof is built in parallel [14].

The above mechanism are not enough to guarantee that the entity producing the proof is, effectively, in a given geographic location. Two techniques have been proposed to overcome this limitation. The first one consists on using the response delay to predict the audited node location, once it may be difficult for the storage provider to issue a timely proof if data is kept in a distant geographic location, due to the extra network latency to access the data. The usage of a set of auditors, in different locations, may increase the accuracy of this mechanism by resorting to triangulation mechanisms [6, 11]. The second approach consists on resorting to a TEE to ensure that operations essential to the proof construction are executed in a given machine [8]. At the same time, this secure environments may be used to interactively reveal the challenge.

2.3 Trusted Execution Environments

A Trusted Execution Environment (TEE) is an isolated environment that can offer stronger security properties for code and data, such as integrity, and confidentiality [15]. A processor with a TEE has two execution modes: the *normal mode*, where the operating system, and applications run, and the *secure mode*, that ensures the integrity and confidentiality of data and code inside the secure mode, even in the presence of an untrusted operating system.

There are a set of different TEE technologies. In this work, we resort to *Intel SGX* technology, as it is the TEE technology available in Intel systems, and we use Intel machines in our experimental setup.

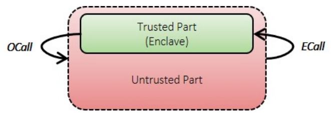

Figure 1. *Intel SGX* components and execution mode switching calls.

Intel SGX splits the computing environment in two parts: the trusted parts, a set of TEEs or *enclaves*, and the untrusted part. Applications in secure mode run inside the enclave, while applications in normal mode run in the untrusted part. Furthermore, as the processor core can only execute one environment at a time, the exchange of environments happens through hardware calls, more precisely *ECalls* and *OCalls*. Figure 1 provides an overview of *Intel SGX* components, and the executed calls for execution mode exchange.

Apart from code, and data security, *Intel SGX* provides security guarantees about the machine identification. In other words, any external machine that communicates with the enclave has guarantees that is communicating with a real enclave, through the *attestation process* [7, 15].

3 Proof of Timely-Retrievability

In this work, we introduce a *Proof of Timely-Retrievability* (PoTR) as a proof of storage that aims to asses whether a storage node can retrieve stored data with a latency smaller than some specific threshold δ. The value of the δ parameter can be selected by the auditor based on the specific requirements of a given application, but it is typically small and, in some cases, can only be satisfied if the audited node keeps the data locally. For instance, many edge services and applications require response times below 5–30 milliseconds [18], thus, a fog node that stores data to serve these applications needs to be able to retrieve data with a latency in the order of the milliseconds.

Our PoTR is obtained as a response to an audit, therefore, it is require to deploy a machine with auditing capabilities, i.e., with the ability to, based on a set of measurements, conclude whether the edge storage provider complies with the expect SLA. However, as fog nodes are in a large number, and geographically distributed [16], our proof is designed in such a way that it can be extracted by any node in the internet, regardless of its location. This property offers a lot of flexibility regarding the deployment of the auditor, and allows for a single auditor to perform audits on a large number of fog nodes.

We could have opted to design a PoTR requiring the auditor to be placed near the audited node. For instance, if the auditor is placed in the same network of the data clients, it can observe directly the actual latency experience by the clients. Unfortunately, this method would be expensive to audit a system with a large number of fog nodes, as it requires to move the auditor, or to deploy a large number of auditors. We could also ask clients to report the latency they observe and use this information to perform the audit. However, this approach raises many challenges as client privacy needs to be preserved. If a PoTR can be extracted independently of the auditor location, we avoid these limitations.

3.1 Obstacles

In the design of our PoTR, we face some obstacles, namely: i) timing information provided by the audited node cannot be trusted [1], thus, the time to produce the proof must be measured by the auditor; ii) the network between the auditor, and the audited node is subject to variance, that may introduce an error when estimating the time the audited node took to produce the proof; iii) fog nodes are heterogeneous [16], and the time they require to perform computations and read data (even if the data is local) is not constant, therefore, the proof should be based on average values from multiple readings; and iv) the audited node may attempt to delegate the generation of the proof to another node, that has faster access to the data than the audited node itself, so it is require to ensure the proof is produced by the audited node, and not delegated.

3.2 Assumptions

The protocol to extract a PoTR is executed between two nodes, an auditor and an audited fog node. We make the following assumptions regarding these two nodes, and the network that connects them: i) the auditor knows exactly which files, and their content, are assigned to a fog node, and files are immutable [1]; ii) a fog node is considered correct if it can retrieve the files assigned to it with a latency smaller than some given threshold δ. Any fog node that cannot satisfy this requirement is denoted to be faulty. The value of δ is application specific,

[1]Mutable files can be supported by creating new versions of a given file, where each version is immutable.

but can be small and, in some cases, can only be satisfied if data is stored in a storage device directly connected to the fog node; iii) the auditor is aware of the latency distribution observed by a correct node, when accessing data items, and when performing the computations to build the proof; iv) the auditor is able to obtain a characterization of the network connecting it to the fog node, and has access to the latency distribution of the messages exchanged between them (this can be achieved through multiple requests, before an audit).

We also assume that each fog node has a processor with *Intel SGX*, as we leverage on the guarantees provided by a TEE, and that the auditor has guarantees that is communicating with the expected enclave [7, 15], due to the attestation process. We assume that the integrity and confidentiality of the data and code inside the enclave are guaranteed [7].

We assume that the enclave in the fog node is unable to read an accurate and reliable system clock [1], as i) the system clock is vulnerable to manipulations; ii) the operation to read trusted times, like the one provided by TPM (*Trusted Platform Module*) has a huge overhead; and iii) the node untrusted part may delay time-messages, to fetch on-demand remote data blocks, and escape detection. Thus, it is impractical to use the enclave for time measures. Nevertheless, the enclave provide us with security guarantees that the proof is effectively built at the audited node [15].

Finally, we assume that is not economically feasible for the edge storage provider to reallocate all objects in less than δ at the beginning of the audit, from a remote storage location that cannot satisfy the δ threshold on access latency to a nearby storage location that can satisfy δ, i.e. the cost to reallocate all objects every time there is an audit would exceed the costs of maintaining data objects in a location with a access time within δ.

3.3 System Architecture

Our PoTR is computed by a fog node to prove that it can access data with a latency that is smaller than a given agreed threshold δ. Thus, the system architecture includes an auditor, a set of fog nodes, and a set of remote storage systems, connected to the fog nodes. Simultaneously, multiple and heterogeneous clients are connected to the fog nodes, that may satisfy the clients requests.

In addition, each fog node has an *Intel SGX* processor, with an enclave, and an untrusted part. The enclave is trustworthy to the auditor and, therefore, it is the fog node component that communicates with the auditor. The enclave is trustworthy to the auditor, as it authenticates towards the auditor during the attestation process. Hence, the auditor has guarantees that the code running inside the enclave is the expected one, and that this code is protected from the remaining operating system [7]. The untrusted part is not trustworthy to the auditor, and is responsible to keep files in local storage, and communicate with remote nodes, if necessary.

3.3.1 Fog Node Storage Organization. The edge storage provider is responsible for storing the files at the fog layer. Therefore, to fulfill the SLA, the provider must ensure that files are stored in such a way that they can be retrieve by the fog node with a latency smaller than δ, the access time threshold.

In each fog node with *Intel SGX*, local documents are kept at the untrusted part, as the enclave storage capability is limited. Thus, if the processor is running inside the enclave, there must be an exchange between execution environments to read a data item (from enclave to untrusted part). Even if the data item is remotely, there is also an exchange of execution environments, as the untrusted part is the one responsible to communicate with remote machines [7].

The set of files that needs to be stored by a fog node and that, therefore, can be subject to the auditing procedure, is common knowledge to both the auditor and the audited node. Both parties have a copy of the set of files sorted in a deterministic manner. Thus, both parties can use the index of a file in the sorted list as a mutually agreed short unique identifier for that file. We denote this index as the *set index*. Note that this organization is only possible if the auditor and the audited node agree on the set of files. The mechanisms required to negotiate which set of files can be subject to audit is orthogonal to the contributions of this work.

3.4 Design of the Challenge

The challenge requires the fog node untrusted part to access a given number of data objects, with a given sequence and return, at the end, a value related to the retrieved data objects. *The delay the fog node takes to read the expected data blocks, and to compute the final value, is used by the auditor to estimate the reading delay δ observed at the audited node.*

Each challenge (implicitly) specifies a sequence of files that needs to be accessed by the audited node, and each file is uniquely identified by a *set index*. For a matter of efficiency, the fog node for each file reads a data block with size s_b,[2] instead of the whole file.

In each challenge c, the fog node has to read a pseudo-random and unpredictable sequence of N data blocks (each with size s_b), and return a cryptographic hash of the concatenation of all accessed data blocks. The number N of data blocks is a configuration parameter, that influences the challenge accuracy and efficiency: higher the value N, more accurate, but less efficient, will be the proof.

The pseudo-random sequence of data blocks is determined by a nonce η^c (unique per challenge, and generated by the auditor), and the data blocks content. For each challenge, the auditor sends the nonce, encrypted with a symmetric key), the number N of data blocks, and the size s_b of a block to the enclave. To ensure the nonce η^c is unknown to the untrusted part, the enclave computes its cryptographic hash. With the result hash, the untrusted part is able to determine the set index of the first file to be accessed. The set index is determine by applying a module function to the hash with the total size of the set of files.

Simultaneously, as the fog node has to read a data block with size s_b, and not the whole file, the auditor sends, to the

[2]Usually, a multiple of the block size used by the fog node file system.

enclave, a second nonce η_b^c that will determine a data block inside the file, with the computed set index. The enclave, in turn, computes the cryptography hash of this second nonce, and forwards the result to the untrusted part. Both nonces are hashed by the enclave, to ensure the untrusted part does not know them in plain text. Otherwise, the untrusted part could maliciously pre-determine the N data blocks. Now, with the hashed η_b^c, and the total size of the file to be accessed, the untrusted part can determine the data block, with size s_b, inside the file to be retrieved, by applying a module function.

After reading the first data block, the untrusted part computes the cryptographic hash of its content, together with the file set index, and returns the result to the enclave. The enclave, in turn, computes a new hash with the response and the nonce η^c, and forwards the result to the untrusted part, as this new result will identify the next file. Only this new hash, enclave dependent, will determine the next file set index. To increase the data blocks index randomness, the enclave computes a second hash, with the hash of the retrieved data block.

With the new hashes, the untrusted part repeats the module functions to obtain the next set index, and the data block index (inside the identified file), reads the data block, and computes its cryptographic hash with the file set index, returning to the enclave. The enclave will compute two new hashes and so on, until the N data blocks are read. At the end, the enclave computes a final hash with η^c, and sends the result back to the auditor, as the final proof. The auditor, after receiving the final hash, repeats all the above computations, to build a verifiable version, to check the proof correctness. If both computations match, the issued proof is correct.

Security Guarantees. Each challenge requires two unique nonces and the sequence of N data blocks is pseudo-randomly chosen. This ensures the fog node is unable to retrieve the N data blocks in parallel from neighbor nodes. Simultaneously, with the dependency between data blocks, as the next block index depends on the content of the previous one, the fog node can not reuse outputs from previous challenges. Finally, the constant exchange between execution environments (from enclave to untrusted part and vice versa) ensures the fog node can not resort to a remote machine (at the cloud or in the edge) to compute the whole proof. The proof is effectively computed at the expected machine [15]. Note that with verification of proof correctness, the auditor assesses the storage documents integrity. Any modified accessed data file will result in an incorrect proof.

3.5 Estimating the Reading Delay δ at the Audited Node

Apart from checking the proof correctness, the auditor tests the proof timeliness. The auditor, when sending a challenge to the enclave, starts a timer that is stop when the proof is received. A proof is valid if it is correct and the reading delay estimate, for a single data block, is acceptable to the auditor.

Being T_i the time elapsed between the auditor sending the challenge i and receiving the response from the fog node, T_i may be decomposed in different factors, namely:

$$T_i = rtt_i + \alpha_i^1 + \ldots + \alpha_i^N + \delta_i^1 + \ldots + \delta_i^N \tag{1}$$

where rtt_i is the network round-trip time for the challenge-response messages, N the number of data blocks to be accessed, α_i^j the delay observed at the fog node to compute the cryptographic hash of a given data block j and compute the index of the next block and δ_i^j the delay observed to read a data block j. Note that for sufficiently large values of N, we will have:

$$\frac{\alpha_i^1 + \ldots + \alpha_i^N}{N} = \overline{\alpha}_i \approx \overline{\alpha} \qquad \frac{\delta_i^1 + \ldots + \delta_i^N}{N} = \overline{\delta}_i \approx \overline{\delta} \tag{2}$$

However, the auditor is unable to measure accurate values for rtt_i, $\overline{\alpha}_i$ and $\overline{\delta}_i$, once they are different and variable at each challenge i and, at the same time, the auditor is only able to measure the total delay T_i. Therefore, to estimate $\overline{\delta}$, i.e., the actual mean delay a fog node takes to read a data object, in our work, we resort to mean values:

$$\overline{\delta}_{\text{estimate}} = \frac{T^i - \overline{rtt} - N\overline{\alpha}}{N} \tag{3}$$

Thus, the estimate error, for a given challenge i, will depend on how distant the observed values are to the mean values. Experimentally, we verified that the α values are subject to a negligibly small variance, whereby we assumed $\overline{\alpha}_i = \overline{\alpha}$, i.e, we discard the error introduced by α sampling. Hence, the error estimating $\overline{\delta}_{\text{estimate}}$ depends on two factors: i) the reading error ε^{δ} when estimating $\overline{\delta}$ (that depends on the sample size, i.e., the number of accessed data blocks) and ii) the variance between the observed network round-trip time (rtt_i) and the expected mean value ($\overline{rtt}$), divided by the number N of data blocks, as Equation 4 describes:

$$\varepsilon^{\text{rtt}} = \frac{\left|rtt_i - \overline{rtt}\right|}{N} = \frac{\Delta^{rtt}}{N} \tag{4}$$

3.6 Configuring the Challenge

As introduced above, larger the number N of data blocks, lower the estimate error. However, there will always be an error, even if small, when estimating $\overline{\delta}$. Therefore, when configuring a challenge, the user must provide two parameters: the maximum error $\varepsilon_{\text{max}}^{\overline{\delta}}$ that he is willing to tolerate, when estimating $\overline{\delta}$, and the challenge reliability ϕ, i.e., the probability to estimate $\overline{\delta}$ with an error lower than $\varepsilon_{\text{max}}^{\overline{\delta}}$.

With the maximum error $\varepsilon_{\text{max}}^{\overline{\delta}}$ and the reliability value ϕ, the auditor is able to determine the number N of data blocks, to ensure a low estimate error. At the same time, the auditor when determining N takes into account the network distribution between the auditor and the target node, and the reading delay distribution of the audited node. Given the reliability value ϕ and with the inverse cumulative distribution function (ICDF) of the network distribution, we compute the worst case difference between the observed value rtt_i and the expected mean $\overline{rtt}$. With this difference (Δ^{rtt}), we are able to determine a value N that places the estimate error of rtt_i above a given value $\varepsilon^{\text{rtt}}_{\text{max}}$.

On the other hand, from the reading delay distribution, at the fog node, and with the a learning curve, we are able to determine the maximum value $\varepsilon^{\delta_i}_{\text{max}}$ of the difference between the mean expected value $\overline{\delta}_i$, that comes from sampling measurements, and the real observed $\overline{\delta}$. Thus, the number N of data blocks may be chosen so that $\varepsilon^{\text{rtt}}_{\text{max}} + \varepsilon^{\delta_i}_{\text{max}} < \varepsilon^{\overline{\delta}}_{\text{max}}$.

4 Evaluation

4.1 Experimental Setup

In our experiments, we use three machines. One of the machines is used to execute the auditor. Another machine is used to execute the audited node. Finally, the remaining machine is used as a remote storage node, that is used by the audited node to access data in configurations where it does not store the files locally. In our experiments, the fog node stores all files at the same location, i.e., all files are stored locally, or all files are stored in the remote node.

The three machines are physically placed at our lab and are connected via a switch, with a mean network delay, between any two machines, of $0.1ms$. We extended our evaluation to emulate deployments where the different machines are placed at distinct geographic locations. To emulate wide-area links, we artificially add delays to the messages exchanged by any pair of nodes. These delays are drawn from a network latency distribution that has been observed experimentally with machines geographically apart.

4.1.1 Access to Remote Storage. The fog node is set up to have access to two file systems: a local and a remote one, so that we can test our proof with both an honest and a dishonest provider. The remote file system is placed at the remote storage node. We have experimented two different techniques for the fog node to access the remote file system:

Secure Shell FileSystem. We set up the fog node to access the remote file system via the *Secure Shell FileSystem* (SSHFS) protocol. With SSHFS the fog node reads files from remote storage as if they were in local storage. At the same time, we resorted to *symbolic links*, to allow code implementation to be storage location independent. For each file system (local, and remote), there is a set of symbolic links. According to the file system it aims to access, the fog node uses the respective set of symbolic links.

One advantage of this configuration is that the audited node uses the same file system interface to access the files, regardless of their location.[3] A disadvantage of this approach is that the SSHFS protocol has a significant overhead. In particular, to access a remote file using the SSHFS protocol the audited node may need to perform several remote calls, which amplifies the effect of the network latency, and makes a dishonest node easier to detect.

[3]The different protocols used when accessing local or remote files are hidden by the Unix's virtual file system.

Client-Server Plugin. Due to the limitations of the SSHFS-based configuration, we decided to implement an alternative technique for supporting remote file access by the audited node, that avoids the latency amplification phenomena. It works by placing a server on the remote storage machine that serves as an helper to the audited node to respond to the challenge (the audited node works as a client to this server). When the audited node is requested to compute the hash of a block it does not store locally, instead of attempting to read the block itself, it sends to the server all the information needed to compute the hash, i.e., the hashes corresponding to the set, and block indexes. The remote node reads the data block, computes the file cryptographic hash, and sends the result back to the fog node, which forwards back to the enclave. In this approach, the only visible overhead is the network between the nodes, avoiding the extra complexity caused by the SSHFS protocol. This strategy requires a single round-trip in the network to execute a step of the challenge, and avoids the transfer of the actual block in the network. As far as we can see, this is one of the most effective strategies for a dishonest node to hide its misbehavior.

4.1.2 Network Emulation. In our experimental setting all machines are connected to the same local area network. To assess the performance of the system when the auditor is at different locations, and when the audited node stores the data in remote nodes, we have resorted to network emulation.

We have used the *ping* tool to collect the distribution of the observed round-trip time between two endpoints of the network we were trying to emulate. For our experiments, we have collected these values for the network between the IST campus in Taguspark (Oeiras) and our lab at INESC-ID in Lisbon, and between the Google datacenter in London and also our lab at INESC-ID in Lisbon.

For each of these networks we have collected enough samples to obtain a good approximation of the distribution of the round-trip latency. We collected 600 pings for both endpoints. Then, to emulate a network, we have artificially added a delay in the communication between the two endpoints. We have used two different techniques to introduce this delay, each with its own advantages and disadvantage:

The first technique, that we follow describe as the *naive adversary*, combines the SSHFS protocol to access remote files, with the emulator *NetEm* (*Network Emulator*), available in Ubuntu operating system. *NetEm* as the advantage of allowing to control *all* the communications between two nodes. Therefore, it allows to delay all calls performed by SSHFS without any additional instrumentation. In this case, *NetEm* was configured to follow a normal distribution and to use the mean and standard deviation values from the samples that we have captured. One limitation of using *NetEm* is that it may introduce some bias when approximating the real distribution, for instance the resulting average round-trip time can be larger than the target value [12].

	Scenarios			
	TI	TII	TIT	TIL
Auditor	Taguspark	Taguspark	Taguspark	Taguspark
Fog Node	INESC-ID	INESC-ID	INESC-ID	INESC-ID
Remote Node	-	INESC-ID	Taguspark	London

Table 1. Nodes location for evaluation scenarios.

The second technique, that we describe as the *ingenious adversary*, uses the client-server plugin to request the remote computation of the block hash, and the following technique to add delays: every time a remote call was performed, we have instrumented the code to include a sleep time with a duration randomly selected from the samples collected from that network. Albeit simple, we observed that this method was able to offer a better approximation of the real network round-trip time distribution. We have used this technique for the client server plugin and also to the communication between the auditor and the audited node, given that, in these cases, it was easy to identify the points in the code where the instrumentation needed to be inserted.

4.2 Evaluation Scenarios

We now introduce the different evaluation scenarios, where for each scenario, we interchanged the auditor and the remote node between three locations: Lisbon and Oeiras, in Portugal, and London, United Kingdom. We kept the fog node fixed in Lisbon. When two machines were placed in Lisbon, we have used the real network. When two machines were placed in different locations, we emulated the wide-area link using the techniques describe above.

With *ping* tool we measured the network distribution between INESC-ID and Taguspark, and between INESC-ID and London, and for both networks we computed the mean (μ) and standard deviation (σ) values. The network INESC-ID–Taguspark is characterized by $\mu = 7.4ms$, and $\sigma = 12.3ms$. In turn, the network INESC-ID–London is characterized by $\mu = 34.5ms$, and $\sigma = 1.7ms$. Surprisingly, the network INESC-ID–Taguspark is subject to a larger variation, i.e., more unstable, than the network INESC-ID–London (INESC-ID–Taguspark network has a larger standard deviation value than INESC-ID–London network).

Given the local-area network, between the three machines, and the above wide-area links, we were able to create a set of 12 evaluation scenarios. In these 12 scenarios, the auditor, and the remote storage locations are interchanged between INESC-ID, Taguspark, and London. In Table 1, for space reasons, we only provide the set of evaluation scenarios where the auditor is placed in Taguspark, as it is the set of most critical scenarios for the auditor, due to the network variability between the auditor, and the audited node. The TI scenario represents a case where the fog node keeps the set of documents in local storage.

4.3 Configuring the Challenge

When setting a challenge, the main factor to take into account is the number N of data blocks to be retrieved, as it affects the error $\varepsilon^{\overline{\delta}}_{\max}$ and the challenge reliability ϕ.

The number N of data blocks depends on two factors: (1) the network between the auditor and the fog node, and (2) the reading delay distribution. Hence, $\varepsilon^{\overline{\delta}}_{\max}$ and ϕ depend on them too: more unstable is the network between the auditor and the fog node and, more variable is the estimated reading delay distribution, greater is the estimate error and lower the challenge reliability. Yet, as the error that comes from the network between the auditor and the fog node is dispersed through the number N of data blocks (Equation 4), the variance of the reading delay distribution is, at the end, the main factor of error.

Simultaneously, as the auditor is aware of the reading delay distribution at a fog node that complies with δ, the auditor is able to detect an honest provider with higher accuracy than a dishonest provider. Note that a dishonest provider may have an infinite set of misbehaviors. For instance, it may serve the files with different access times, i.e., some within δ and some beyond δ, to change frequently the files location and, thus, access time, which makes it more difficult to the auditor to detect it. As a consequence, our proof is able to guarantee a false positive rate (FPR) but, in the general case, cannot offer guarantees on the false negative rate (FNR).

The FPR describes the number of challenges, out of the executed ones, that identify a fog node as dishonest, when the node is actually honest. The FNR describes the number of challenges, out of the executed one, where a node is signed as honest, but it is actually dishonest. We select a number N that provides acceptable FPR and FNR.

In our proof, we want to meet a FPR of 0.01% and a FNR of 0.05%. Through experiments and error analysis, we noticed that with $N = 1000$, we can meet the desired rates. Note that the configuration of $N = 1000$ depends on the δ threshold and, on how distant the estimated access times are to δ. We follow describe the observed estimate error $\varepsilon^{\overline{\delta}}_{\max}$ for $N = 1000$, as it is hardware dependent, and we used different machines in different moments of our experiments.

4.4 Results

We divided our experiments in two adversaries: a naive adversary and an ingenious adversary. In each one we resorted to different techniques for both remote storage access, and network emulation. We also, for each adversary, used different machines [4]. Hence, each configuration will have a distinct estimate error $\varepsilon^{\overline{\delta}}_{\max}$, as it depends mainly on the reading delay distribution, that is hardware dependent.

Although we divided this section based on the two adversaries, there are some challenge configuration parameters that are hardware independent or, even if hardware dependent,

[4]This is due to the fact that we have upgraded the machines in the middle of the evaluation process.

are the same in both setups. We now describe the common parameters and configurations:

In the first place, we opted for data blocks of size $s_b = 64KB$,[5] as it is provides a low variance [14] ($1.0ms$ and $0.3ms$ variances for the first and second adversaries, respectively).

Second, we filled the file system with a dataset of 520184 images [17], with sizes between $846B$ and $180KB$. When the audited node has to read a file with a size smaller than $64KB$, the file data is padded with 0 until the data block has $64KB$. The index of the next file to be access is determined by the cryptographic hash of the $64KB$ data block with the hash with $32B$ from the previous iteration. We used *SHA256* function for the cryptographic hash and *Rijndael AES-GCM 128bits* algorithm to cipher the nonces.

In each adversary, we ran a set of challenges, each one configured with $N = 1000$. Simultaneously, we ran the challenges in the scenarios where the auditor is placed in Taguspark (Table 1). We opted for the scenarios where the auditor is in Taguspark, as it is the more unstable network and, thus, a more critical and challenging scenario for the auditor. To this end, we have emulated the INESC–ID-Taguspark network between the auditor and the fog node.

4.4.1 Naive Adversary. The implementation of the naive adversary uses a combination of SSHFS and *NetEm*. We configured *NetEm* to follow a normal distribution based on the network samples captured with *ping*. For the network INESC-ID–Taguspark, we set up *NetEm* to follow a normal distribution with $\mu = 8.0ms$ and $\sigma = 2.0ms$. The μ and σ values are different to the ones captured directly with *ping*, as we did an adjustment to the samples in order to follow a normal distribution.

For the naive adversary, we used a Intel NUC7i7DNKE machine, with a Intel i7-8650U CPU, that supports SGX, 8 GB RAM, 250GB M.2 SSD and, runs Ubuntu 20.04 LTS. Experimentally, we measured the delay this machine takes to compute the cryptographic hash ($\overline{\alpha}$) and we obtained $\overline{\alpha} \approx 0.8ms$.

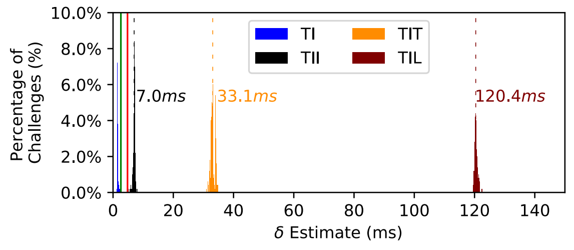

Figure 2. Estimate of the reading delay δ, in milliseconds, at the naive adversary, by an auditor emulated in Taguspark and the remote storage node emulated between INESC-ID, Taguspark and London. The green line is the $DT \approx 2.6ms$, and the red line is the $ULT \approx 4.8ms$.

[5]A multiple of $4KB$, the block size used by the file systems.

Scenarios	$\overline{\delta}_{\text{obs}}$	$\overline{\delta}_{\text{est}}$	$\lvert\overline{\delta}_{\text{obs}} - \overline{\delta}_{\text{est}}\rvert$
TI	$1.6ms$	$1.5ms$	$0.1ms$
TII	$7.1ms$	$7.0ms$	$0.1ms$
TIT	$32.6ms$	$33.1ms$	$0.5ms$
TIL	$120.3ms$	$120.4ms$	$0.1ms$

Table 2. Observed ($\overline{\delta}_{\text{obs}}$) and estimated ($\overline{\delta}_{\text{est}}$) mean access time at the naive adversary, and auditor respectively.

Figure 2 shows the estimated reading delay δ verified at a naive adversary, with the auditor emulated to Taguspark and the remote storage node emulated between INESC-ID (TII), Taguspark (TIT) and London (TIL). The first scenario (TI), describes the case where the audited node keeps the whole dataset in local storage and, does not resort to a remote storage node. In each scenario, we ran 500 challenges.

Note that for the scenarios where the audited node resorts to a remote storage node for storage and, the remote node is emulated to Taguspark (TIT) or, London (TIL), the estimated access times are 4x the mean of the emulated network. This overhead is due to SSHFS remote calls, as *open*, *lseek*, *read*, etc, that are also subject to the delay added by *NetEm*. The two first scenarios TI and TII are equally subject to the overhead introduced by SSHFS remote calls. However, in the scenario TI, the SSHFS overhead is less clear as the remote calls happen inside the same machine. Yet, we show that with our challenge the auditor is able to estimate access times closer to the access time observed at the audited node.

Table 2 describes the observed and estimated averages values by the audited node and by the auditor, respectively. The auditor can estimate the observed reading delays with a maximum deviation of $0.5ms$ for the TIT scenario. The TIT scenario registers the larger deviation, as it is the case with the more unstable network between the audited node and the remote storage node.

In this experiments, we considered a $\overline{\delta}$ value equivalent to a local storage access, i.e., $\overline{\delta} \approx 1.6ms$, as it is a critical scenario, due to TI and TII estimates proximity. The TI and TII scenarios have the lowest access time difference, between any two scenarios (a difference of $5.5ms$). Nevertheless, with $N = 1000$ and a FPR=0.01%, for the TI estimates, we can ensure $\varepsilon^{\overline{\delta}}_{\max} < 1.0ms$.

With this being said, a fog node that complies with δ is correctly assigned as honest, if the estimated access times, by the auditor, are below $2.6ms$. We describe this threshold as the detection threshold (DT), as it corresponds to the maximum access time acceptable to the auditor, that ensures the required FPR=0.01%. Even though the scenarios TI and TII have the lowest estimate difference, the TII estimates are clearly above DT, which means that even with an difference of just $5.5ms$, the auditor is able to accurately distinguish a fog node that keeps files in local storage from a fog node that resorts to a remote storage node in the same local-area network. Note that the DT depends on the value δ agreed in the SLA, and the error $\varepsilon^{\overline{\delta}}_{\max}$ at an honest node.

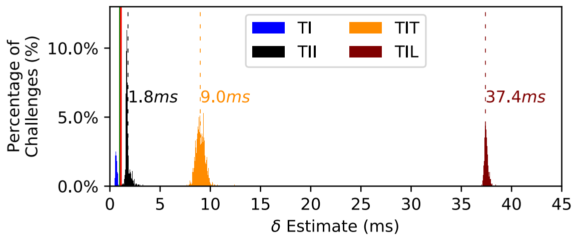

Figure 3. Estimate of the reading delay δ, in milliseconds, at the ingenious adversary, by an auditor emulated in Taguspark and the remote storage node emulated between INESC-ID, Taguspark and London. The green line is the $DT \approx 1.0ms$, and the red line is the $ULT \approx 1.1ms$.

In addition, we defined an unacceptable latency threshold (ULT). This threshold lower bounds the access time estimates for a FNR of 0.05%. As there are a set of possible scenarios, we defined the ULT based on the TII scenario, as it is the one closer to the DT. Given the difference between the average estimate and the value that bounds 0.05% of TII estimates, we obtain the false negative margin. With $N = 1000$ and a false negative margin of $2.2ms$, we define an ULT at $4.8ms$. Any fog node with access time estimates above $4.8ms$ is clearly assigned as dishonest (does not comply with the δ).

The scenarios were the audited node resorts to a remote storage node (TII, TIT and TIL) have estimated access times are above $4.8ms$, thus, above the ULT, by what their misbehavior was clearly detected by the auditor. In a matter of fact, for the ran 500 challenges, we did not verify any false positives neither false negatives.

4.4.2 Ingenious Adversary. The ingenious adversary combines the client-server plugin with the sleep approach. In the experiments for the ingenious adversary, we used a Intel NUC10i7FNB machine, with a Intel i7-10710U CPU, that supports SGX, 16GB RAM, 250GB M.2 SSD and, runs Ubuntu 20.04 LTS. The remote storage machine is similar to the adversary machine. Experimentally, we measured the cryptographic hashing delay of a $64KB$ data block with a $32B$ hash and we obtained $\overline{\alpha} \approx 0.1ms$.

Scenarios	$\overline{\delta}_{\text{obs}}$	$\overline{\delta}_{\text{est}}$	$\lvert\overline{\delta}_{\text{obs}} - \overline{\delta}_{\text{est}}\rvert$
TI	$0.5ms$	$0.6ms$	$0.1ms$
TII	$1.8ms$	$1.7ms$	$0.1ms$
TIT	$9.5ms$	$9.0ms$	$0.5ms$
TIL	$37.5ms$	$37.4ms$	$0.1ms$

Table 3. Observed ($\overline{\delta}_{\text{obs}}$) and estimated ($\overline{\delta}_{\text{est}}$) mean access time at the ingenous adversary, and auditor respectively.

Figure 3 shows the estimated reading delay δ by the auditor, for the ingenious adversary. The auditor is emulated to Taguspark and the remote storage node is emulated between the considered three locations: INESC-ID (TII), Taguspark (TIT) and London (TIL). The auditor also estimated the reading delay for an audited node that does not resort to a remote storage node (scenario TI). For each scenario, we ran 1000 challenges. Table 3 describes the observed and estimated average values by the ingenious audited node and the auditor, respectively.

The estimated values in the scenarios with remote storage (TII, TIT, TIL) have no longer an extra overhead introduced by the remote storage access approach. Instead, the estimated access times are closer to the mean of the network between the audited node and the remote storage node. At the same time, the estimated values for the first scenario (TI) are lower than the estimated at the naive adversary, for the same scenario, as we used more advanced hardware.

Simultaneously, note that the variance between the estimated values for the TIT scenario is larger than the verified for the naive adversary. This is due to the configuration of the sleep time approach. As we select randomly a value from the captured samples, the sleep time, for a set of challenges, will follow the network variance, that is in this case $151.3ms$ ($\sigma^2 = 12.3ms$). In the naive adversary the variance is lower, as we set up *NetEm* with $\sigma = 2.0ms$ and, thus, there is a variance of $4.0ms$.

In this experiments, we considered, $\overline{\delta}$ as the access time to access files in local storage, i.e., $\overline{\delta} \approx 0.5ms$. We select the observed mean value at the TI scenario, as it is, once again, a critical scenario for the auditor. The difference of the estimated values between the TI scenario and the TII scenario is just $1.3ms$ (lower than the same difference in the naive adversary). Hence, the auditor has a smaller gap to place the DT. Thus, it becomes more difficult to distinguish the TI and the TII scenarios.

For the local storage scenario (TI), with $N = 1000$ and for FPR=0.01%, there is an estimate error $\varepsilon^{\overline{\delta}}_{\text{max}} < 0.4ms$. Hence, we are able to define the DT at $1.0ms$. Thus, the auditor is able to detect the audited node misbehavior in scenario TII, as the estimated values are above the DT. In the ran 1000 challenges, we did not verify a false positive.

As for the false negatives, we define the ULT at the $1.1ms$. Once again, we define this threshold based on the estimated values for the TII scenario, as it is the one closer to the DT. From the difference between the mean and the delay that ensure 0.05% estimates are above it, we obtained a false negative margin of $0.1ms$. Remember that the ULT lower bonds the estimates from misbehavior nodes, for a FNR of 0.05%. In the 1000 ingenious adversary experiments, we did not verify any false negative, as all estimates from misbehavior nodes were above $1.1ms$.

4.5 Discussion

In both naive and ingenious adversary, we define δ as the delay to access files in local storage, i.e., in memory space in the audited node machine. Thus, the DT is placed next to the TI estimates, as for the given δ, we aim to distinguish a fog node that keeps files in local storage from one that resorts to a remote node. Moreover, the defined thresholds allow to detect a fog node that resorts to a remote node in the same local-area network (scenario TII). Yet, both δ values and the DT are not fixed values for all evaluation contexts. Given a set of requirements, the client may define a lower or larger δ value than the one considered in our experiences. One example, would be to test whether the audited node keeps files stored in Portugal or not. For this example, and taking into consideration Figure 3, the δ value may be defined as $12ms$, and the DT could be place at about $15ms$. Note that we are able to determine some example values, as it is clear the difference between remote storage in Portugal (TIT scenario) and outside of Portugal (TIL scenario). We may even not observe any false positive, or false negative, as the estimates are clearly distant to the DT. In other words, even if the network between the fog node and the remote node is unstable and, thus, the estimate error is high, due to estimates distance, the auditor can detect storage within Portugal versus outside with accuracy.

Equally, if any two distinct scenarios have estimates close to each other, it is more difficult for the auditor to place the DT to ensure a given FPR. Thus, in this case, the auditor may increase the number N of data blocks to increase the challenge accuracy or, may accept a larger FPR. For a scenario where we we accept a latency equivalent to files storage in Portugal border with Spain and there is a remote storage in the border of Spain with Portugal, it may be hard for the auditor to distinguish if a the audited node keeps files in Portugal or resorts to a remote node, outside, but closer to the border.

5 Conclusions

In this thesis we describe a novel 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. Our auditing mechanism is based on a challenge that requires the fog node to access, in sequence, a pseudo-random set of data items, and to respond to the challenge in a timely manner. The challenge is designed in a such a way that, if the fog node does not store locally a significant fraction of the objects, it will be unable to respond in time. We leverage on the existence of TEE, more precisely Intel Software Guard Extension (SGX), to ensure the proof is executed at the audited node, and reduce the communications between the auditor and the fog node. The enclave is responsible to iteratevely reveal the next data block to be read to the untrusted part.

We have implemented and evaluated experimentally our auditing mechanism. Our results show that our proof can accurately detect a node that is not able to satisfy the target latency constraint δ.

Acknowledgments

This work has been developed in the context of the project Cosmos, that aims at developing a storage system to support edge-computing. In the development of the work I benefited from the discussion and collaboration with other members of the Cosmos team, in particular from working with Cláudio Correia, that is addressing on the security aspects of the project. This work was supported by national funds through Fundação para a Ciência e a Tecnologia (FCT) as part of the projects with references UID/CEC/50021/2019 and COSMOS (financed by the OE with ref. PTDC/EEICOM/29271/2017 and by Programa Operacional Regional de Lisboa in its FEDER component with ref. Lisbon-01-0145-FEDER-029271).

References

[1] F. Anwar et al. 2019. Securing Time in Untrusted Operating Systems with TimeSeal. In *2019 IEEE Real-Time Systems Symposium (RTSS)*.

[2] F. Armknecht et al. 2016. Mirror: Enabling proofs of data replication and retrievability in the cloud. In *Proceedings of the 25th USENIX Security Symposium*.

[3] G. Ateniese et al. 2007. Provable Data Possession at Untrusted Stores. In *Proceedings of the 14th ACM Conference on Computer and Communications Security*. Alexandria, Virginia, USA.

[4] J. Benet. 2014. IPFS: Content Addressed, Versioned, P2P File System. *arXiv preprint arXiv:1407.3561* (2014).

[5] J. Benet, D. Dalrymple, and N. Greco. 2017. Proof of replication. *Protocol Labs, July* 27 (2017).

[6] K. Benson et al. 2011. Do You Know Where Your Cloud Files Are?. In *Proceedings of the 3rd ACM Workshop on Cloud Computing Security Workshop*. Chicago, Illinois, USA.

[7] C. Correia et al. 2020. Omega: a Secure Event Ordering Service for the Edge. In *2020 50th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*.

[8] H. Dang et al. 2017. Proofs of Data Residency: Checking Whether Your Cloud Files Have Been Relocated. In *Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security*. Abu Dhabi, United Arab Emirates.

[9] U. Drolia et al. 2017. Cachier: Edge-Caching for Recognition Applications. In *2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS)*. Atlanta (GA), USA.

[10] M. Feldman and J. Chuang. 2005. Overcoming Free-Riding Behavior in Peer-to-Peer Systems. *SIGecom Exch.* 5, 4 (2005).

[11] M. Gondree and Z. Peterson. 2013. Geolocation of Data in the Cloud. In *Proceedings of the Third ACM Conference on Data and Application Security and Privacy*. San Antonio, Texas, USA.

[12] A. Jurgelionis et al. 2011. An Empirical Study of NetEm Network Emulation Functionalities. In *2011 Proceedings of 20th International Conference on Computer Communications and Networks (ICCCN)*.

[13] J. Leitão et al. 2018. Towards enabling novel edge-enabled applications. *arXiv preprint arXiv:1805.06989* (2018).

[14] L. Li and L. Lazos. 2020. Proofs of Physical Reliability for Cloud Storage Systems. *IEEE Transactions on Parallel and Distributed Systems* 31, 5 (2020).

[15] S. Matetic et al. 2017. ROTE: Rollback Protection for Trusted Execution. In *26th USENIX Security Symposium (USENIX Security 17)*. Vancouver, BC.

[16] M. Mukherjee et al. 2017. Security and Privacy in Fog Computing: Challenges. *IEEE Access* 5 (2017).

[17] Nari. 2017. tiles_256x49. https://www.kaggle.com/narimatsu/tiles-256x49?select=0005f7aaab2800f6170c399693a96917_14.png. [Accessed: 2021-03-12].

[18] G. Ricart. 2017. A city edge cloud with its economic and technical considerations. In *2017 IEEE International Conference on Pervasive Computing and Communications Workshops (PerCom Workshops)*.

[19] M. Satyanarayanan et al. 2017. Cloudlet-based Just-in-Time Indexing of IoT Video. In *Proceedings of the Global Internet of Things Summit (GIoTS)*. Geneva, Switzerland.

[20] C. Streiffer et al. 2017. ePrivateEye: To the Edge and Beyond!. In *Proceedings of the Second ACM/IEEE Symposium on Edge Computing (SEC)*.

[21] T. Taleb et al. 2017. On Multi-Access Edge Computing: A Survey of the Emerging 5G Network Edge Cloud Architecture and Orchestration. *IEEE Communications Surveys Tutorials* 19, 3 (2017).