Edge Networked Storage: Enabling Seamless Data Management, Processing and Reuse in Edge Networked Systems

Adrian-Cristian Nicolaescu

A transfer report submitted in partial fulfillment

of the requirements for the degree of

Doctor of Philosophy

of

University College London.

Electronic and Electrical Engineering

University College London

July 2023

I, Adrian-Cristian Nicolaescu, confirm that the work presented in this thesis is my own. Where information has been derived from other sources, I confirm that this has been indicated in the work.

Abstract

My research provides an insight into the management and service provision of storage and processing (computing) services, at the Edge of the Internet.

The research starts with the development of knowledge in the technological background that a system has to be made in, to provide flexible, secure and diverse services to an ever-growing user base, efficiently and fast. We find soon that this user base is already quite diverse, and most of the needed services have to be provided very far from the devices from which the data to be served originates. Thus, current services are generally provided with very high latencies because of the distance that data generally has to travel to the service provider and back to the user.

The study of a system that can provide these services closer to users and still be flexible enough to provide any kind of service to any user across the Internet thus has to be conducted. The research of this system starts with a study on the feedback and control mechanisms needed, for computing nodes to manage themselves and provide feedback within a controlled, stable, secure and diverse computing environment, at the Edge. This study is followed by another work, looking into the development of a new architectural view, specific to this Edge environment, to provide the best quality of service while keeping this environment both flexible and effective in service provision. These are the first steps in developing a system that can provide any storage or processing service to the Internet or its Edges.

All considered, a plan was made, to pursue the research needed to develop parts of this system, or at least the equivalent, modular parts/algorithms, to be able to form a flexible and efficient Edge-based computing service provision system, by the end of the PhD.

Impact Statement

Since the 2000's, the attention given to and utilisation of cloud technologies associated with data centres has increased substantially. As more users (commercial and non-commercial) are using the internet-based services (e.g. Google Maps, Ring Security, the MetaVerse, social networks, traffic monitoring) their potential of collecting, creating, processing and distributing huge amounts of data at the edge of the Internet has correspondingly increased.

The big-data based tasks that the above-mentioned data sources entail have also increased drastically in size and complexity. These tasks currently run mostly on centralised data centres, and their needs have also grown as cloud technologies have been more widely adopted, creating the environment for designing and developing more complex and distributed software tools, which revolutionised virtualisation, task distribution and the possibilities for networked systems configuration. Since around 2015, edge computing and Multi-Access Edge Computing (MEC) have started gaining increasing weight in the networks research community. Recently, edge computing has even started reaching research maturity, and taken large steps in implementation, through more massive implementations of private 5G and O-RAN network domain deployments.

More recently, extensive research has started into implementing intelligent multi-access technologies, towards seamless cross-technology handovers and communications integration. Likewise, for the network layer and up, network design towards task orientation and embedded network intelligence (networks for intelligence) has started. Energy efficiency in networked (cloud and edge) computing and network servicesis also an increasingly important topic of research. All these new research areas need a way to house and distribute all the different(ially) relevant data. The research conducted in this thesis will prove very impactful towards the supporting and essential technologies needed for all of the above-mentioned, bleeding-edge research subjects in today's networks and general IT research fields.

With so much data, of different types and formats, being distributed across today's networks, there is a grave need for classification and distribution knowledge, for all this data to be used and abused in the best way possible, and for it not to go to waste under any circumstance. Information theory has been given very little thought recently, and it is high-time that attention is given to it. The best support for today's need for network intelligence would be provided by a very well organised data distribution system. For this to happen, categorisation, a determination of hierarchy and classification within the different categories and hierarchies needs to be done for all data. There are certain differentiations that need to be made between application-specific data, network monitoring, network policy, network optimisation and network intelligence (training, ML type/algorithm, inference and model) data, SLA data, domain data and higher-policy (e.g. domain poicies, locality, task orientation) data. To do this, the data storage facilities and access policies should be very well defined and known, both by machines and network operators. That is how the domain of network cognition has come to life. It is now a very important field of study towards 6G technologies and needs in-depth research and development.

Acknowledgements

The contributions of a few people were essential to the development of the knowledge-base and needed background for coming up with the ideas presented in this work. Two of the main people who contributed to this work, through technical advice and structure feedback were Dr. Argyrios Tasiopoulos, who received his PhD during the summer of 2019 and Dr. Onur Ascigil, who was also a post-doctoral researcher in our group. Since early on and until today, the contributions and guidance of my most supportive supervisor, associate and friend, Dr. Spyridon Mastorakis, Assistant Professor from the University of Notre Dame, my supervisor and guide, Prof. Miguel Rio, Professor of Computer Networks at UCL and the very accurate and perseverant Dr. David Griffin, Principal Research Associate at UCL were much appreciated, in current and, more importantly, towards further developments. I could have also not done this PhD without Dr. Ioannis Psaras, who provided me with the opportunity to pursuit my ideas to develop systems in and contribute to the field of Computing at the Edge of the Internet.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

Today's Internet, and especially edge-based services, devices and users, like the ones that use Internet-of-Things (IoT) and smart city applications are generating increasing amounts of data [1].

Traditionally, the flow of data consumption on the Internet followed a "core-to-edge" model based on the assumption that the main data sources/servers resided in the core of the network infrastructure (e.g., in a data-center) and was requested/consumed by users at the edge of the Internet. Nowadays, with the explosion of the IoT [1], this model is reversed [2]: massive amounts of data are generated by user devices at the edge and flow towards the core of the network infrastructure, for processing and storage purposes. To alleviate the pressure on the network introduced by this reverse data flow model and to achieve low-latency data processing, edge computing emerged as a prominent paradigm that makes computing and storage resources available at the edge of the Internet [3]. With it, some new network architectures started emerging and gaining research interest as well, like Information-Centric Networks (ICN - explained below), because of the popularity of internet caching mechanisms, and with the context of caching, storing and effectively managing massive amounts of user-device-generated data at the edge became an issue that needs to be addressed [1]. This issue becomes particularly challenging, since the generated data may be of different types (e.g., IoT sensor data, images, video frames) and may also have different needs and purposes (e.g., to be processed, to be simply stored at the edge, or to be eventually offloaded to a cloud).

The concept of Information-Centric Networking (ICN) was introduced in 2006, via Content-Centric Networks, by Van Jacobson (one of the developers of the original TCP) and a few others [4]. The concept of ICN was further pursued and put forward through the NSF-funded ”Future Internet Architectures” project. A few different network architectures were developed within this project and field of study (the attribution of addresses to networked data, rather than networked nodes). From these, the most distinctive and relevant for our study are Named Data Networks and several hash-based routing protocols [5, 6, 7, 8, 9]

Edge nodes offer many advantages for cloud computing through data storage and query processing done closer to users and providing data sources so as to reduce interactions with central clouds. This reduces the quantity of traffic sent over long distances, reducing traffic load and improving energy efficiency while addressing privacy concerns by keeping queries and data local. It reduces the latency of interactions, improving Quality of Experience for delay-sensitive services. However, edge nodes are typically of much smaller scale than central data centres and are prone to the risk of being overloaded by processing or storage demands, especially if demand patterns are not well known in advance when dimensioning deployed resources.

Edge computing and Edge storage solutions have been proposed and have started being deployed in real-world scenarios to address this change [10] throughout the past 5 years. At the same time, many edge applications operate on similar (correlated) data, which is a feature that can be exploited for serving multiple queries with the same results [11, 12].

In this context, the field of **computation reuse** is emerging, which proposes the sharing of processing results among applications and services in cases of similar input data.

Content Delivery Networks (CDNs) have been proposed as a prominent solution to deliver requested content efficiently to users through replicated surrogates [13], which are often deployed at the edge. However, CDNs were designed to deliver specific, identifiable content rather than services that are user- and session-specific.

With the help of all the above-mentioned points to be approached, it can be logically inferred that a stable, secure and efficient storage and processing environment is needed for data produced at the Edge of the Internet. The starting idea for a solution based on these principles was the implementation of an Edge Data Repository (EDR) environment. This environment, its implementations and the proposed resulting system following these developments will be presented in the chapters following the Literature Review.

Chapters 3, 4 and 5 tackle the three most important challenges that this system faces (obtained from the issues also identified and presented in the previous section):

1. EDR computing-centric management and feedback for localised offloading;

2. Data distribution, retention, efficient management and delivery of data, towards system and network efficiency, while maintaining very good service standards.

3. Improvement of Edge networks performance and efficiency through the distributed, multi-purpose, unique and orchestrated (re)use, storage and (re)placement of functions and data;

1.2 Research Question

The currently-implemented Edge-to-Core system uses and re-uses Bandwidth (BW) inefficiently, incurs high latency and creates large numbers of upstream bottlenecks. This can happen due to the constant increase in demand for upload and local connection support, and an increasing need for the processing of Edge data with lower latencies. With these issues in mind, there are a few aspects that, while considered within the research community of Edge networking, they are barely touched. A major topic that is not given too much consideration within the Edge networking community is Edge networked storage. This consideration, thus, poses the following question:

What are the improvements and impact that an efficient and more intelligent edge networked storage system could have on the QoE and performance provided through edge networks in terms of capital investments and operational costs offered and needed by Edge networks by/to providers and/or (their) users?

To answer this question, the main focus of the studies conducted towards the final system and architecture design will be on information validity, reliability, retention, reusability and delivery mechanisms. However, to start with, we must give a better definition for intelligent Edge networked storage:

Edge Networked Storage does not represent networked cache, because it is not session-bound; it is also **not exactly** application-specific, but rather application-agnostic and, at the same time, QoE- and system resource-optimised and time- and information validity-centric, so that information is distributed where and when it is needed, in a secure, traceable way. Furthermore, Edge networked storage retains and/or (re)distributes information towards a better utilisation of network resources (routing, processing and storage) and the better satisfaction of network services (better QoE), as stated above.

Considering this research question, a list of issues and ideas to be scrutinised, for solving the problems that this research question poses is necessary. The topics of interest were identified in different ways: through papers' analysis and determination of missed points within their context; through finding missing subjects identified not to be approached as counterparts to IP; by simply looking at the base IP/ICN-based architectures, some of their derivates, and works that could have approached the identified and considered idea for research and design; or by studying and developing solutions based on the above concerns and making certain assumptions and/or discovering new avenues for research. The above-mentioned issues/resulting ideas are covered by the following bullet points.

- The first identified problems of the research subject matter were first determined. The main problems first approached were user mobility, upload bottlenecks, created by edge-based devices and users accessing cloud-based services, and that of data availability and validity timings. The approach was use-case-led and real-world-trace-led, to provide realistic connectivity, Edge processing and storage to Edge nodes, Edge-based users and Data Providers. Through simulations, a comprehensive study and results of data storage, processing was conducted in the EDR definition paper. These simulations provided grounds for strategies based on different message-specific deadlines, implemented for placement with the ERP algorithm, in EDR environments. A feedback and management algorithm for ”global”/cluster-wide performance and service QoS adaptation was developed based on the above-mentioned study;

- Changing the point of view upwards in hierarchy, inside the EDR environments, and dealing with information-centricity and data storage efficiency, SEND was designed. This approach took into consideration different criteria (e.g., data popularity, proximity of data to processing functions running at the edge) to improve the placement of different categories of data at the edge and solves the problem of data availability and distribution at the Edge (which is needed for function execution). To aid data placement and management decisions, SEND relied, among other attributes, on system-wide identifiers of the data context, called labels.

- From the above studies, we identified and focused on a few other problems, as well: information efficiency, processing and storage efficiency, function and data distribution efficiency and service delivery mechanisms. We approached and solved these problems in ReStorEdge. Information retention, delivery efficiency improvement and environment efficiency improvement were studied, by integrating the concept of Locality-Sensitive Hashing and computation reuse with our system, and implementing a hybrid, Hash-and-MetaData routing optimisation system, to preserve both storage and network resources. The idea for exploring information validity and reliability was to use the readily-available message-specific labels, and most importantly, hashes, to determine the way in which the data could be re-routed, to re-use processing and the stored data, towards making the system more efficient (processing less data and using less storage space), improving QoS in the process, as well, by reducing response times and network paths to a node that has more availability. For the improvement of network organisation and service delivery, we implemented an edge storage and computation environment orchestrator, which used the above-mentioned system and its resulting metrics to its advantage, to coordinate the use, storage, processing and re-use of information within its domain, for the benefit of associated customers (data/application providers and/or their users) and its own (the edge environment provider's performance and efficiency).

1.3 Main Contributions

To provide a simple, yet comprehensive review of the work put into this system's design, during the PhD, and to give a small insight into the design's complexity, the contributions to date and some of the proposed solutions for further development are as follows:

- First, the structure of the underlying system was designed, developed and evaluated. The main contribution of the obtained paper was an algorithm and the practical evaluation. The system was made to store data at EDRs for as long as the data may be useful at the edge (e.g., for processing purposes), offloading the data to the cloud for archiving once it is not useful at the edge anymore. This use-case-led design was implemented for the evaluation of the EDR environment in realistic scenarios and to provide a good insight into the simple, Edge-based deployment feasibility of EDRs, to provide connectivity, Edge processing and storage;

- SEND was designed to take into consideration different criteria (e.g., data popularity, proximity of data to processing functions running at the edge) to improve the placement of different categories of data at the edge. To aid data placement and management decisions, SEND relies, among other attributes, on system-wide identifiers of the data context, called labels. Labels are used to tag the data that enters the SEND system, offering a light-weight mechanism for identifying the data type(s), the data generation sources, and the processing functions supported. SEND also implements and improves on the store-process-send system, highlighted above. The experimental results demonstrated that SEND achieves data insertion times of 0.06ms-0.9ms, data lookup times of 0.5ms-5.3ms, and on-time completion of up to 92% of user requests for the retrieval of raw and processed data. With these results, comparing to the previous solution, of the uncoordniated processing system to which the SEND system is compared, an improvement of QoS performance of approximately 40-60% can be observed;

- Finally, considering the above and developments in locality-sensitive hashing (LSH), we proposed an architecture for processing queries over the same number of Edge Data Repositories (EDRs) and topology as above, called ReStorEdge. ReStorEdge exploits the feature of computation reuse, by storing the results of previous computations and returning stored results when queries are sufficiently similar to earlier ones. We used application-specific similarity searches on query content to determine whether a query can be satisfied with a cached result to avoid additional computational load on the edge node, instead of re-processing the query. We implemented a similarity-based data classification system, which we evaluated with real-world datasets of images and smart-assistant speech measurements, and with realistic processing functions and timings. We further performed a network simulation study to evaluate different orchestration strategies based on real-world datasets, evaluating the performance and trade-offs of the design as a whole.

1.4 Thesis Outline

In the following, we will go through the organisation of the remainder of this thesis. The most basic and important parts and background, essential for understanding of the main contributions that this PhD work brings forth are provided and briefly explained in Chapter 2. The tendency and need for data, services and latencies that are faster and closer to users are shown and explained in Section 2.1. The research and engineering communities' general approach and terminologies for Cloud, Fog and Edge are then approached, in Section 2.2. The main research question that this PhD is concerned with is then provided, explained and motivated in Section 1.2, then with that in mind, the main (EDR) environment's essential parts, main storage implementation developments and needed background are introduced and presented briefly in Section 3.2, and afterwards, the rest of the orchestration and management mechanisms developed are briefly introduced, for background purposes, in Section 2.3. A review of the related work and background set for these implementations will be presented in Chapter 2.

After providing all the needed introductions and background, the contributions made within the PhD (as briefly presented in Chapter 2) will be covered in detail, with all their associated design and evaluation contributions. Thus, in Chapter 3 we will look in detail at the start of the EDR environment development, by showing the importance of storage for Edge-based function execution, storage and cloud-bound routing bottleneck reduction.

Further developments into the architecture, environment management improvements and associated protocols will be covered in Chapter 4 and Chapter 5, followed by the proposed system developments, design and implementations, in **??**, and finally, the progress, impact, system development and final thoughts on the system's/framework's and the PhD achievements, towards the wanted (and needed) network architecture implied by the Edge Data Repository environments and, ultimately, the Internet will be reviewed, discussed and concluded in Chapter 6.

1.5 List of Publications

The research conducted during thie PhD has produced three major contributions, two of which have been published in a workshop within a high-profile ACM conference and another in a top IEEE networks conference. One other resulting paper was submitted to another top networks research conference. The list of publications is given below.

- **Conference Publications:**
  - Adrian-Cristian Nicolaescu, Onur Ascigil, and Ioannis Psaras. Edge data repositories - the design of a store-process-send system at the edge. In Proceedings of the 1st ACM CoNEXT Workshop on Emerging In-Network Computing Paradigms, ENCP '19, page 41–47, New York, NY, USA, 2019. Association for Computing Machinery
  - Adrian-Cristian Nicolaescu, Spyridon Mastorakis, and Ioannis Psaras. Store edge networked data (SEND): A data and performance driven edge storage framework. In IEEE INFOCOM 2021 - IEEE Conference on Computer Communications, pages 1–10, 2021
- **Paper under Review:**
  - Adrian-Cristian Nicolaescu, Md Washik Al Azad, Spyridon Mastorakis, David Griffin, Miguel Rio. ”ReStorEdge: An edge storage system with computation reuse semantics”

Chapter 2

Background and Literature Review

2.1 Towards the Edges

Lately there has been a visible increase in data production and distribution within Edge Networks. This can be attributed to several advances in technology, such as: the increase in popularity of live video streaming, continuous development of social networks and their employed Edge data analytics, increasing usage of messaging applications, the distribution of augmented/virtual reality (AR/VR) devices and applications [14] and, most importantly, the notable increase in interest, development and deployment of IoT and Smart City applications [15] and/or services. These applications and services impose an increasing need for support at the Edge of the Internet, for data distribution, storage, processing and compression/filtering service provision and mobility.

Most of the current problems with data delivery and services are associated with upstream network bottlenecks from Edge networks. As described above, users require constant Internet support to benefit from the required services.

2.2 Cloud, Fog and Edge

The ”Cloud age” is starting to fall short of what it promised and still promises to deliver, since what applications need (especially new ones like AR/VR, IoT that did not exist 20 years ago) has changed since the earlier implementation of the (more centralised) Internet. Clouds are currently falling in popularity, due to their high latencies and lack of support for the increasing amounts of data being generated by end-user devices. [16, 17, 18]

The Cloud is generally based in data centres and it provides certain points of presence. Because of their locations and available resources, Clouds have very flexible resource allocation and large resource pools, both for processing and storage. Even though clouds can be distributed, certain mismanagement, attack actions or system failures can make these networks and/or providers very large single-points of failure[1]. If one part of the Cloud fails, many dependent applications and third-party services hosted on that cloud will most probably fail, which can greatly affect the Quality of Service (QoS) and Quality of Experience (QoE) provided to users/customer companies.

Service provision can also be slow and costly, due to data being connected to or from the end-users through ISP-provided networks and clouds. Data could have to do round-trips through both ISP and Cloud networks. Every hop in this routing- and transaction-chain has to be paid for, accounting for the price of transfer, connection and hosting, even if the destination of all these services is one hop away.

There is a certain differentiation between the Fog and Edge environments, but these are quite flexible and not very well defined. Most researchers and engineers define the Fog as the network environment 3-6 hops from the end-users towards the Core network. Considering today's Internet, most of the network management, efficiency, interconnection, commercial and service provision policy decisions should be taken in Fog networks. [19]

In this case, consider the Edge of the Internet to be defined as the space between Edge data producers/consumers (end-users) and 2-to-3 hops toward the Internet Core. Edge Networks include first-points-of-contact of end-user devices to **a network (not necessarily the Internet)** and sometimes up to the first, or at most second-hop private-network border router that are connected to the rest of the Internet. In today's Internet, Edge networks are becoming increasingly diverse and complex. As a result, these networks could very soon get out of hand, due to their increasing need for higher bandwidths and reliability, lower latencies and faster

[1]Major Facebook DNS outage: link here

trigger-match-to-action reactions.

2.3 Storage, Processing, Routing and Management at the Edge

In the context of the Edge Data Repository (EDR) environment, storing and effectively managing massive amounts of generated user device data at the edge is an issue that needs to be addressed [1]. This issue becomes particularly challenging, since the generated data may be of different types (e.g., IoT sensor data, images, video frames) and may also have different needs and purposes (e.g., to be processed, to be simply stored at the edge, or to be eventually offloaded to a cloud).

To address this issue, we proposed a data storage and management framework at the network edge, called "Store Edge Networked Data" (SEND). SEND aimed to improve the performance of the network edge and the QoS provided to users by keeping raw (generated by user devices) and processed (output of processing services/functions) data in persistent storage close to users for periods of time longer than cache storage. In SEND, EDRs, are deployed at the edge and seamlessly interact with flows of network traffic. EDRs accept different categories of data (e.g., data to be stored at the edge, the data input(s) and output(s) of processing functions running at the edge, the code of the processing functions) that may transit through the edge environment. Through a logically centralized management plane, SEND takes advantage of the characteristics (attributes) of the data generated at the edge to make decisions on where to store and whether/how to replicate the data across different edge locations in order to maximize the offered QoS.

Throughout the PhD, it was noted that quantities of data generated at the edge and research interest into edge computing have been increasing. [1] Edge computing and Edge storage solutions have also been proposed and have started being deployed in real-world scenarios to address this change [10] recently. These developments and the fact that many edge applications operate on similar (correlated) data had potential to be exploited for serving multiple queries with the same results [11, 12].

In this context, the field of **computation reuse** is emerging, which proposes the sharing of processing results among applications and services in cases of similar input data.

Content Delivery Networks (CDNs) have been proposed as a prominent solution to deliver requested content efficiently to users through replicated surrogates [13], which are often deployed at the edge. However, CDNs were designed to deliver specific, identifiable content rather than services that are user- and session-specific, while computation reuse is not. To address these challenges, we proposed an edge framework, called ReStorEdge. ReStorEdge offers data processing, storage and computation reuse capabilities at the edge for a range of services. Examples of such services include media tagging, subtitling, and natural language processing.

2.4 From Edge to Cloud

There are three problems that CDNs currently solve, and each one of them could be improved. These will be highlighted, explained and further discussed in the following paragraphs, with the implementation of ICN architectures in mind.

”By far, the most important performance measure for streaming video is average end-to-end throughput.” [20]. Today’s and the book’s view on this statement is one of fetching pre-recorded content from the core of the currently IP-based network, with the increased efficiency of CDNs. The reverse process is given little consideration within the book and little attention in today’s network environment. With IoT, VANETs, Smart City, Wearable device, AR/VR and other Edge-based applications gaining exponential adoption these years, an Edge-to-Core (also Edge-to-Edge and Edge-to-Fog) data delivery approach is needed. An approach has to be made towards an efficient ”reverse Content Distribution Network” (rCDN) [21]. Considering the last statement and the current and foreseen data heterogeneity and abundance in Edge networks, an Information-Centric addressing approach and an in-network, managed Edge storage system are needed to tackle this gargantuan challenge - to find, classify, aggregate, store, process and finally deliver massive amounts of Edge-generated data (whether video, sensor, location, text or others) to the core or other end-systems/users in the Edge/Fog, or even through the Core network (or Cloud).

The second problem, of popular content being sent many times over (mostly) the same down-stream links and making ISPs/users pay for no new content being introduced into the domain is solved by ISPs subcontracting CDN providers to provide servers within their networks. However, the same cannot be said about the upstream links, since there are no ”reverse-CDN” solutions implemented currently. The implementation of an ICN-based approach to in-network persistent data storage distribution and management within these domains would solve this problem effortlessly.

The third problem, of single-point failure is, again, partially (in the downstream) solved by CDNs, but yet again, an ICN-based implementation of Edge data storage and service delivery would solve this problem by completely ridding the network of single-entity addresses and replacing them with data addresses, also fully eliminating the problem of any kind of data loss. This approach mainly shifts the threat into the field of data management and slightly into security. These issues were further tackled in the latter parts of the PhD, as shown in the next chapters, by touching the topics and developing ideas for future work in the planning and as exposed in the introduction.

Edge Data Repositories represent the idea of implementing storage and processing (or variations of these, with processing/storage capabilities) in/co-located with routers inside the larger network, to either replace CDNs or provide the service that they do (storage), but closer to the Edge, for reverse data flows (towards clouds), in a more distributed manner, in the case of bottlenecks (a compression-decompression process) and on-site (in the data plane). The consumption rate can be important in this case, and could potentially be achieved via a store-process-send mechanism. Albeit, this process does depend on the classic trade-off between quality/quantity and timing of data delivery. Considering current technologies, this approach should be much faster than the use of extra network resources and time, for rerouting, distributing, aggregating and possibly processing the data in other parts of the network. If data is simply stored and not processed in the persistent storage of routers, both the live-streaming case and the network longer-term aggregation/distribution and data fetching performance could be greatly improved.

Xu et al [22] provided an initial assessment and analysis on Edge storage and computing more recently, highlighting problems/developments, addressed in Chapter 5. With this occasion, we introduce the fourth, and most current problem: The introduction of intelligence to the Network Edge. That is systems intelligence, working not only on Edge-based networks, but also **inside** the Edge networks, themselves.

The original concept of Edge Data Repositories (EDRs) was developed taking inspiration from the Mobile Data Repositories position paper [23]. They were originally designed to provide a good solution for storing and processing information at the Edge and conveying information from the Edge towards the core. These systems were developed to be integrated in the larger networks, and develop on some of the shortcomings, assumptions, mechanics and concepts of a few other papers, to provide a solid development and implementation plan. These inspiring ideas, designs and mechanics will be presented in the next chapter.

First, the papers will be reviewed in a specific, structurally-defined order. Afterwards, a closer look will be taken at the specific systems to be implemented, contrasting and comparing, to provide a better view of the Edge Repository Provider (ERP) system's needs. Questions will appear along the way to the end of this chapter. These questions have either been addressed already, by the introductory material or they will be answered later, in the technical content chapters (3, 4 and 5), within this thesis.

2.5 Producer Data Push Mechanism

Since most data originating at the Edge and destined to networked devices has to be at least processed, if not even stored within the Edge networks or on the Internet, for the user's/Application Provider's purposes, this Edge data has to be somehow ”pushed” into the networks. This PUSH mechanism can be needed for different reasons, but among the most important ones are intermittent power and/or network connection, network resource constraints and data availability. Due to this, most networks that rely on Edge-sourced data have to be adapted. Considering that today's networks and data production have been increasing and that there are higher numbers of devices contributing towards these trends, the concentration of the networks research community has also shifted more towards Edge-network adaptations, in accordance with predictions. [24, 25, 26]

2.6 Network-layer Storage

Most of the existing literature that contentrates on edge data storage does it from a service point of view. Different approaches for the optimal placement of data caching and processing units have been proposed, aiming to maximise the QoS offered by the edge [27, 28, 29]. Other papers are more management-driven utilising the logically centralized intelligence of Software Defined Networking (SDN) to install forwarding rules on edge routers, so that generated data can be forwarded to the closest edge server for processing and storage [30]. In none of the papers, however, storage is maintained within the Edge environment (the former strictly considering processing and the latter considering only fog-level, session-specific caching - 4 to 5 hops into the network). ICN (especially NDN and hash-routing) supports in-network, Network-layer storage natively [31, 32, 27]. In every case considered above that employs ICN, however, wherever the data consumer may be and regardless of the amount of data it needs, the data consumer **has to request** the data, at some point in time, and for a duration, or (a) specific (amount of) data. Regardless of where and when this request is issued, it must include some packet identification data and/or MetaInfo/MetaData (filters). Here we argue that the Interest-Data pair is inefficient and slow in handling large amounts of data, which is already split into small chunks. Thus, even though the Interest-Data symmetry and ”control loop” [6] are not very useful in our case, the Interest-type (request) packet could still create a PIT entry in each router receiving it, down to the ERP's domain border. A more detailed explanation of this process and some new concepts will be introduced in

Chapter 3 and Chapter 4

2.7 Edge- and Fog-based Computing

Further to the line-speed-capable storage requirement, it must also be mentioned that payload processing is also considered for implementation, on top of the storage service, which adds an extra layer of complexity to the system (incurring extra costs).

Some of the current research [33, 34], especially into keyword-based systems [27] (which works with a type of **"labels"**, to help address data, similarly to [35]), shows great potential for line-speed Edge computing placement and resource allocation adaptation. However, storage is not considered for the improvement of processing in most Edge computing research to date. To obtain application-specific distributed function execution environments in each of the storage entities, the functions to be executed in different storage environments (around the world) need to be distributed. Some of the functions may be mapped to the necessary areas for execution first, while others might have to be mapped to higher-level networks before distribution and instantiation [36].

The approaches mentioned above did not offer integrated data location awareness along with data storage and distribution of (raw or processed) data or the code of the processing functions. All the partial solutions and concepts needed to develop the new architecture offer such an integrated approach where all data (raw data entering the system or processed data created by the system - for example, the output(s) of a processing function at the edge) is stored at the edge for as long as it may be useful to users and applications.

2.8 Network Naming, Authentication and Management

The management and analysis of data at the Edge of the Internet are very important, for the improvement of resource distribution, architecture design, device configuration and network performance. The way data, devices and users are distributed within Edge networks needs to be taken into account for the research and development of any new ideas and design flows.

As the Edge environment and its associated users and data are becoming increasingly important, the management of services offered, Edge resource allocation, Edge-generated data and their associated QoS is becoming exponentially important, as well. Of the last few variables, the most important is the management of data, as the title of this section suggests, due to the fact that all others are indirectly controlled by the efficiency and performance of data management and manipulation.

There are a few very interesting papers which add value to the development of data management (with and without ICN implementations) schemes for different types of devices and data in Edge networks. Out of these, one stands out, and that is **Mobile Data Repositories at the Edge** [23]. This paper is a position paper, presenting a high-level view of placing storage services in the Edge, for storage, uplink decongestion and connectivity, depending on the purpose, scope and needs of the user machine. It provides an overview of the main architecture implementation and proposes a few different approaches to storing data at the Edge. It is one of the main background papers, needed as a cornerstone concept within the developments and implementations of the undertaken research.

Intra-Domain Routing

Throughput is one of the most important metrics for the EDR system. Considering the potential data production rates at the Edge, and especially originating in Mobile Edge nodes (like cars, busses and probably trains), the input rates that EDRs should support should be very high. [37, 38, 39] The most interesting to note with this kind of network traffic is the fact that the traffic can vary by very high amounts, at different times of day, and even between each second or minute [40]. Thus, with such variations of data production and flow, a buffer-like Edge system would not only be useful, but, with the implementation of persistent storage and processing capabilities, could even benefit the whole network. This system could provide router bottleneck alleviation upstream, fast access to persistent storage, for other Edge devices or users and higher information throughput, through packet compression and possibly application level processing.

Now, consider the above statements and the fact that throughput is mainly limited by the routers' output capacities and MTUs. The relevant metrics to use as feedback for this storage system include, but are not limited to: user/IoT Provider connection numbers, connections' mode of functioning (e.g. periodic or non-periodic, message streams or one-off packet deliveries), computational resources (i.e. processing, caching and storage), throughput (needed intra-domain and inter-domain) and others [27].

2.9 Improvement of Edge Storage Management for In-Network Computing

In this section, we present a brief background on storage frameworks, including storage on the cloud and the network edge, along with previous related work.

2.9.1 Storage Frameworks

Extensive research has been conducted on storage frameworks. The community has explored distributed file systems, such as the Network File System (NFS) [41] and Coda [42], and more recently Hadoop [43] and the Google file system [44]. Alternative database designs (e.g., relational, object-oriented, graph-based) have been proposed for data storage and indexing of the stored data [45]. Lately, the community has focused on key-value stores for the deployment of distributed network storage and middleware applications [46], including mechanisms to increase the performance of such stores through programmable hardware [47]. SEND is orthogonal to these approaches and is able to utilize different file system or key value store designs for its internal data storage structure.

2.9.2 Cloud Storage

Data storage has been a popular cloud-based service offered by the majority (if not all) of cloud providers, such as Microsoft, Amazon, and Google. At the same time, there are companies such as Dropbox and Box that specialize on providing storage as a service on the cloud. Extensive research on cloud storage has been conducted by the community. Performance and scalability were among the first properties to be explored [48, 49], while solutions to improve the availability of cloud storage services have also been investigated [50].

Data deduplication designs have been studied to ensure that a single copy of redundant data is stored on the cloud [51, 52], reducing the space and bandwidth requirements of data storage services. Several approaches have also been proposed to safeguard the privacy of the stored user data [53, 54] and perform operations, such as data searches, in a privacy-preserving manner [55]. Finally, issues, such as multi-tenancy [56], data replication [57], and data management [58], as well as their impact on cloud storage have been studied.

Storage services residing on remote clouds typically result in high response delays for applications. Such delays may not be tolerated by applications that require low-latency data access. Moreover, IoT devices generate massive amounts of data, which put significant pressure on the core network for their transmission to remote clouds. To this end, SEND focuses on providing storage as close to users as possible (at the edge of the network) for both raw and processed data.

2.9.3 Storage and Data Placement at the Edge

Psaras *et al.* performed an initial exploration of the benefits and challenges of data storage at the edge to alleviate the stress that the massive amounts of data generated by IoT devices put on the core network for the transmission of these data to cloud storage [2]. Nicolaescu *et al.* built on top of this concept by developing and assessing a preliminary design for the management of popular data at the edge [59]. Liu *et al.* investigated the impact of the size and number of data storage units on the data availability and operational cost [60].

The majority of existing literature tackles the problem of edge data storage from a service point of view. Strategies for the optimal placement of data storage units have been proposed, aiming to maximize the QoS offered by the edge [27, 28, 61]. Other approaches follow a management-driven direction by utilizing the logically centralized intelligence of Software Defined Networking (SDN) to install forwarding rules on edge routers, so that generated data can be forwarded to the closest edge server for processing and storage [30].

Service/function and data placement at the edge has been further explored in the context of specific application domains. Scientific workflows may require taking into account the dependencies between different data types to improve the workflow execution efficiency [62]. Social virtual reality applications may require considering the data of users and the processing logic on these data as a bundle [63]. In this context, optimizing the data placement at the edge is performed to enable user interactions. Finally, for the deployment of connected vehicles, strategies to prefetch and process data at Road Side Units (RSUs) have been designed and evaluated [64].

The approaches mentioned above did not offer integrated data location awareness along with data storage and distribution of (raw or processed) data or the code of the processing functions. SEND offers such an integrated approach where all data (raw data entering the system or processed data created by the system–for example, the output(s) of a processing function at the edge) are stored at the edge for as long as it may be useful to users and applications. SEND utilizes different placement strategies/algorithms based on the context (labels) of each data piece, the processing function(s) that each data piece may be associated with, and the period of time that each data piece may be useful to users and applications at the edge. At the same time, SEND aims to enhance the provided QoS for all applications, users, and devices, without being a solution limited only to certain application domains.

2.10 The considerations of Edge Computing Management with the help of Storage and Reuse

2.10.1 Edge-based Data Storage and Offloading

In [59, 65] and [66], the authors provide insights into Edge networked storage systems and how these could be managed. However, the proposed management strategies consider how data storage could be managed, and not how both processing functions and data storage resources could be orchestrated. At the same time, prior work has explored different strategies for computation offloading from users devices to the edge of the network [67, 68]. Together, Edge-based data storage and offloading [59] can form the basis for frameworks that facilitate the operation of applications at the edge of the network.

In addition, prior work has proposed the processing of data at the edge based on associations [67] or possibly even network flow management and association ([68]). However, there is little research into session-less data flows, processing and storage associations, towards better network resource utilisation and the improvement of system-wide QoS.

Data classification can be very important for certain applications and for choosing specific data for specific applications. A good example of early collaborative data and service classification is presented in [69], through the creation of cooperation groups and associated channels, for information dissemination and cooperation. In most, more contemporary works, the data will probably be (for a very short time) stored in application-specific ”enclaves” or VMs, for security, and then the data could be processed appropriately, through certain data associations ([67]) or possibly even network flow management and association ([68]). However, there is little research into session-less data flows, processing and storage associations, towards better network resource utilisation and the improvement of system-wide QoS.

2.10.2 Computation Deduplication and Reuse

In cloud computing, the storage of several data duplicates becomes an important issue in terms of information distribution, storage reliability and security [48]. Data deduplication has been previously researched in parallel with CDN systems ([52], [70]) to increase the efficiency of data distribution and availability within clouds ([50]). Thus, research on data deduplication at the edge can benefit from prior research in cloud computing data deduplication.

The direction of data deduplication at the Edge of the Internet has lately gained traction and has been discussed in different research publications [66], [71] and [11]. Most techniques for data reuse assume that specific content is addressed by name [72], [73] or by a hash of the name [11]. While [11] considers data storage and reuse across applications it does not investigate the distribution of cached information across a network. We take a different approach and access cached results based on the similarity of the query rather than by using a specific name as an index to a specific piece of content.

Similarity has been previously approached from different points of view (mostly IoT and VR), to better implement and make use of computing at the edge [74] and [75]. On the other hand, these approaches only consider edge-based, end-user, heterogeneous devices, where the environment could be hard to control and secure. These processes can add complexity to the system, create fragmentation in resource allocation and imply a large overhead in processing. This can be an especially hard to solve problem when considering end-user devices' computing capabilities.

2.10.3 Locality Sensitive Hashing

Locality Sensitive Hashing (LSH) [76], [77], [78] and [79] is a technique that exploits hash collisions in such a way that similar queries are mapped to the same bucket. In [80, 81] and [82], the authors use the capabilities of LSH to improve the way in which networks are managed, by **improving routing tables and SDN placement through hashing entries** and using LSH for similarity-based routing, further **improving network security** in the process, by implementing hash-based identity anonymisation.

Some works have previously used LSH with the network itself [83] or processing data/code/results [84] and [85], in order to **improve processing resource usage and timings**. These applications for LSH are very common as of recent, in the research of computation systems. However, the context of these improvements has varied and was never considered to be a network- and/or storage-based system.

One of the most relevant and useful domains to use LSH is within storage environments, due to the ability of data (or certain features of it) to be hashed [86, 87], [8]. Further, data has the ability to be split into smaller "chunks", aggregated and/or deduplicated, in similar and/or purpose-specific conglomerates [74], forming more useful "pools", from which (generally network-based) applications and tasks can draw any needed data as "prime materials". Some studies into **localised storage and computing systems** include [88]

The advantage of local storage optimisation via LSH (LSH being applied **externally or within the storage/computing system**), was previously used for in-network, general storage [75], and computing ([84]), towards achieving better system performance. What is not approached as a topic in the above works is session-less, network-based storage, and its implications. We take advantage of LSH in our work to realize the semantics of computation deduplication and reuse in the context of our framework.

Chapter 3

Investigation of Design Space and Development for Edge Data Repositories

3.1 Introduction

According to several studies [89, 90, 38] and predictions for the following years (2022-24) [17, 91], the problem of upstream bottlenecks for Edge ISPs will also be exacerbated by the need for extra services closer to the Edge[21]. This requirement will be a result of increased network traffic created by connected cars, homes, other Internet of Things (IoT) implementations (e.g. smart cities)[15] and/or augmented/virtual reality (AR/VR) devices[14]. Networks are dispersing, hence resource redistribution is required for a smooth transition towards decentralised or even distributed (private) networks, which are still governed by certain (more general) network policies.

Important service provision requirements are set by Edge data generated through peak-hour commuting and Smart City applications. Existing work [89, 90, 38] suggests that user, IoT and vehicular data will overload the network with information in the next years. These designs consider the reverse data flow, but imply very small data quantities. In that regard, another paper [38] considers a few blue-sky approaches to the grave need of data (re)distribution in the upstream flow.

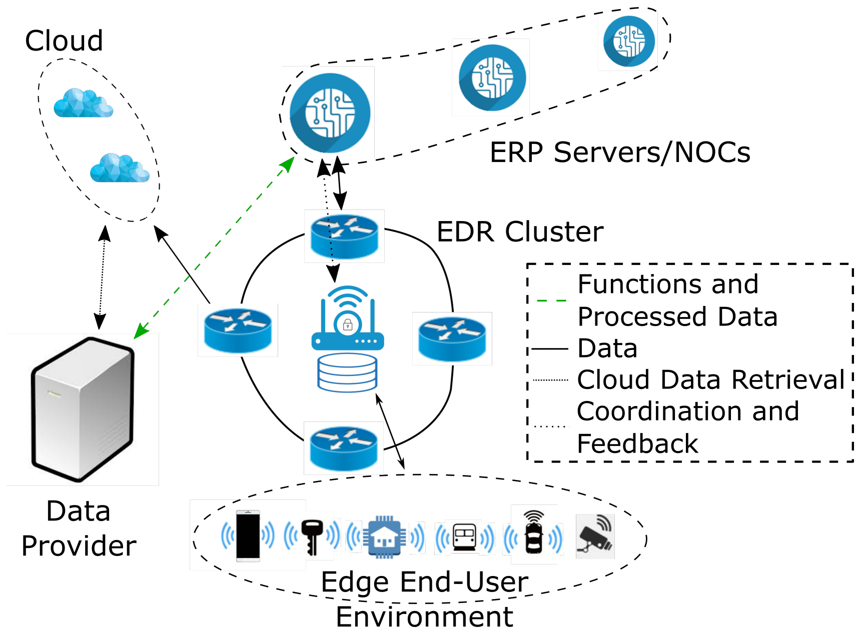
Data from the domains mentioned above could, thus, be more accessible and usable at points closer to Edge networks, for efficiency and latency purposes.

”Mobile Data Repositories at the Edge” [23] presents a new approach to in-network data distribution, to improve connectivity and reduce upstream stress, by using persistent, Network-layer storage [92]. The *Proactive Strategy*, proposed by this system, relies on data packets to tell the Data Repository systems how they should be treated, through MetaData (MetaInfo) tags. Adding to this, we introduce the store-process-send concept, which was developed to integrate processing with storage, to increase data delivery efficiency of data originating at the Edge. Thus, we now define two main tags, useful throughout the rest of this chapter. The *shelf-life* is the minimum storage period for storage service provision and the maximum processing window for processing service provision with lower constraints. The *freshness period* represents a deadline for a time-sensitive data processing service. These help in Data management decisions such as storage time, processing needs or storage/deletion/archival offloading after processing. Considering these concepts, we introduce nodes specialised in data storage and processing, called Edge Data Repositories (EDRs), managed by Edge Repository Provider (ERP) servers (briefly introduced in this paper). This work represents a first step in solving an open question of managing EDR environments [92].

3.2 The Start of Developing a Solution

The Edge Repository Provider systems are meant to provide processing, storage and stable network connections, as services within a secure environment. These goals can be achieved by further identifying an approach and system components, to appropriately satisfy the current commercial and network-based service provision needs. The four connected system parts that affect each other’s service provision parameters and functionality, depending on policies, demands and locality are, thus, described and defined below. (Figure 3.1)

With this first step of the design, we introduced the store-process-send concept, which was developed to integrate processing with storage, to increase the delivery


Cloud
ERP Servers/NOCs
EDR Cluster
Functions and Processed Data
Data
Cloud Data Retrieval
Coordination and Feedback
Data Provider
Edge End-User Environment
Figure 3.1: ERP System Diagram

efficiency of data originating at the Edge. Considering these concepts, we introduce nodes specialised in data storage and processing, called Edge Data Repositories (EDRs), managed by Edge Repository Provider (ERP) entities. This was the first step towards answering an open question, for managing EDR environments [92] and towards developing a realistic and feasible solution for the storage and timely satisfaction of highly demanding Edge-based tasks [59].

From a data perspective, the primary part of the system is the class of devices/systems producing data. The Edge Data Producers (EDPs) are end-devices or end-systems, which produce data with specific topics, mobility, frequencies, types, scopes and destinations. These devices/systems and their data parameters are associated with specific Application/Data/IoT Providers and their users.

Technically, the most important part of this system are the EDR environments, which provide a secure, stable and service-rich environment for Edge Data storage and processing. Security can be ensured through service-specific resource allocation, in containers or enclaves. EDR environments mainly provide stability by storing and processing (mobile) data locally, and providing persistent network services for the respective data. EDR environments are also the hubs that connect this entire system, efficiently and adaptively. An EDR environment's management, security and service provision policies are monitored and managed by its local (ERP) orchestrator, through its service provision and resource allocation feedback and management algorithms.

ERP management entities (Network Operations Centre [NOC] and/or servers and/or EDRs even, in a distributed system) represent key commercial entities. These may be of a central (ERP NOC), or local (edge-based system servers or even EDRs) management type. ERP management entities represent the commercial and policing part of the system, which is mostly concerned with ERP system security and management policies. These entities govern security bootstrapping, function and data coordination, service provision policing, system trade-off monitoring, performance monitoring and management of commercial policies between their systems and other entities.

The other commercial part of the system (outside the system's control) is the Data Provider. This commercial and network entity provides its service and security policies to its own set of users and negotiates commercial, network and security policies with the ERP(s) for its (and it's users') own benefit. Associated EDPs and the functions provided for data processing are also managed by these providers. The main concerns of these providers lie in bootstrapping EDPs with data generation certificates and providing secure functions to the network. Security bootstrapping assures EDRs that producers provide secure data, with appropriate parameters.

Before jumping into system design, a good background of realistic scenarios has to be provided and some Edge data generation use cases will be presented below. The data generated varies across a few dimensions; and all of these dimensions have to be accounted for in the Edge Data Repository Systems. The data variety dimensions to be considered are: size, urgency, device spread across one Edge domain, frequency of data updates, device (and Edge data) mobility, accuracy of data/measurement, unaltered delivery/storage preference, different data type aggregation and processing and more complicated combinations of the different dimensions as well.

A few cases considered in these studies will be shown, through the work presented in this thesis, and these will demonstrate the utility of Edge Data Repositories within the Edge environment, thus helping the development of their associated systems and architecture. These use cases are presented below.

3.3 Data Repository Design

Now we shall examine each part of the system in more detail. EDRs were developed to be modular, configurable and scalable. The functionalities illustrated below demonstrate these features.

Storage: All repositories benefit from network persistent storage capability, being able to buffer and retain data. Depending on service requirements, this functionality may suffice for certain services.

Processing: All repositories have processing capabilities for application-specific service provision. Processing is done independently of line-speed operations, in the application layer.

Storage association and management - After a message is received by a repository, it is added to storage management and the newest processing message is then processed (Data ”Path” of Figure 3.2).

Message processing and update handling - Each update consists of a processing phase and a depletion phase, on each simulation cycle. The update process is presented in the Control ”Path” of Figure 3.2.

Main Question - Considering the system design, the main issue will be approached heuristically. The following questions provide the main reasoning for our evaluation and are inspired from existing Edge networks research [38, 93]: What is the algorithm that determines the most efficient resource configuration, based on available resources, services provided, traffic analysis and EDP types? Could this algorithm be provided with simple yet effective service quality feedback, towards local adaptivity, for specific services, under certain conditions?

3.4 Evaluation

Hypothesis - Consider that storage is used within limits, and while the packets' shelf-lives have not expired, the packets are stored in EDRs. If the EDRs start

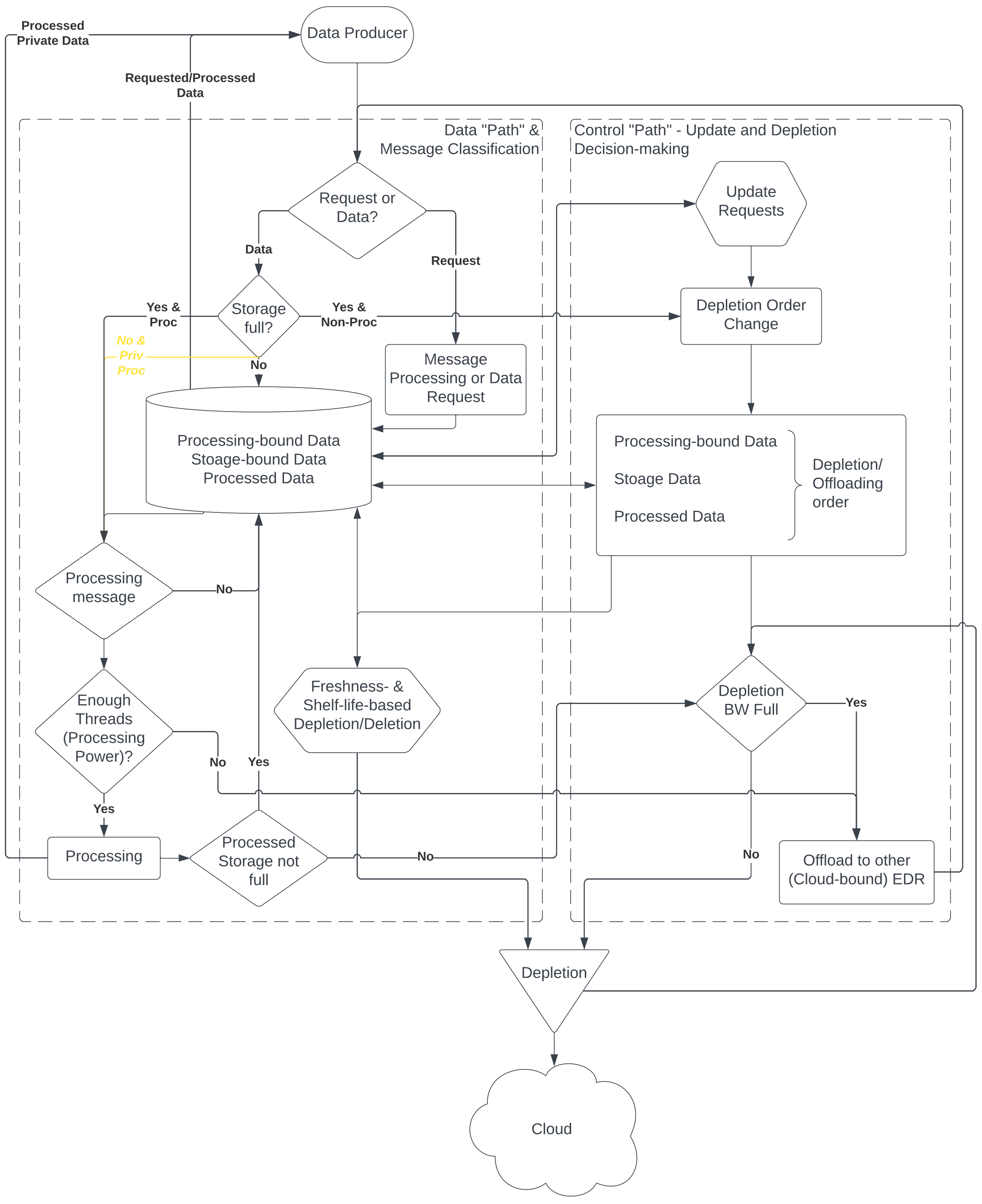


Figure 3.2: Flow chart of internal repository processes - yellow arrow represents a new addition (potential to explore with the later-introduced mechanisms of SEND

overflowing or the packets' shelf-life expires, before they are processed, they will be considered unsatisfied. Thus, it is important to assess the percentage of packets processed within the freshness period or shelf-life or deleted/offloaded after shelf-life expiry. These shall be briefly covered in sub-section 3.4.6, and later in more detail, in sub-section 3.4.5.

3.4.1 Use Cases

A simple, home sensor temperature reading (and possibly action) implies very low traffic, and possibly the need for very little processing power. However, as these types of devices drastically increase in numbers and update frequency (i.e. in home automation and Smart Cities), their implied needs increase proportionally, or at an even higher rate. As their update frequency and/or numbers increase, the needed aggregation (storage) and processing for these types of messages and their associated/aggregated counterparts increase, as well.

A RING Doorbell asynchronous video feed can impose a high traffic flow problem on Internet links; especially if the connections are made via Cloud connections (as most IoT Providers, like RING, have Cloud-based servers). More importantly, this is a problem that one video feed generates. Under the conditions of enormous amounts of upstream background traffic, the bottleneck issue presented above would likely be magnified tenfold.

Smart Home ”continuous” device suite monitoring - considering the suite of devices made by the same manufacturer or supported by the same EDRs - a service that many IoT Providers either want to have or already provide for their users.

TfL Bus location updates can be very tedious in today’s IP networks. On the other hand, busses in London have specialised connections, to have secured access to the Internet and send updates to their TfL servers (independent or in the Cloud). Furthermore, the bus stops are also connected to the TfL servers and could potentially aid in these transmissions and updates (maybe not only for busses).

Zipcar fuel/energy monitoring can be even more complicated than stated above, considering that continuous access to today’s Internet via IP networks is very hard to obtain; especially considering the establishment of TCP (Transmission Control Protocol) sessions. Even with asynchronous, non-critical updates (as this is not a very time-sensitive measurement), it can be very hard for a moving vehicle and an Edge IP network to coordinate data, location and ideal aggregation placement, for function execution in the Edge environment.

Tesla sonar feed and collision detection could represent both urgent and non-urgent data, in one. This is because such data may need some processing, and, in conjunction with some other combination of IoT measurements or recently processed data from the area, could indicate an emergency. This problem is also special due to its implied placement, timing and freshness period restrictions.

TfL Bus and Traffic CCTV ”continuous” video feed represents non-critical (non-urgent) data, with a large size and great potential for specific feature extraction. The main problem with this type of data is that, if sent further into the network, it could incur very high traffic. Furthermore, some of this data could be sensitive, thus needing high compression and security within storage, processing and communication media.

These use cases will be taken under consideration and provide the realistic background to consider throughout the thesis. Some use cases may be more relevant for specific parts of the design, however the modularity of the work and design helps the system be more flexible and impactful in such situations.

All entities interested in the data:

- Local authorities;
- Traffic systems;
- News broadcasters;
- Car insurance companies;
- Car manufacturing companies.

Corner cases and associated data needs

- Traffic systems may, at the point of the accident, be interested in one of the processing responses
- Local authorities may be interested in more detailed data, maybe wanting the raw data (most detailed AND - note - most recent RAW data)
- Local news broadcasters may be interested in a general filming angle of the accident, but at a later point

- The drivers' insurers may be interested in the **first** (**earliest possible notification**) accident message and the on-board footage, at a later time (footage does not even need to be stored on the EDRs, but the general message could, for proof)

- Car manufacturers may be interested in specific, unaltered car data form (either of) the cars, but after a longer period of time; so that data could be archival data, retrievable through the cloud (even though some of that data may be processed on the spot, relevant for current events, but also for later diagnostics and statistics).

3.4.2 Solutions for corner cases

Application Providers/Function Developers and Beneficiaries associations - and their purposes:

Local traffic (and public transport, most often) authorities - they need up-to-date, simple and fast information from their systems, thus using a simple signalling function on the cameras, also using object-detection, to identify the accident (in this example), to output a simple accident message, to either lock the intersection or the relevant part of the traffic ASAP and establish diversions quickly

Local authorities taking any messages and feeds whenever they are available, apart from what the insurance companies and manufacturing companies are interested in - otherwise said, they are interested in all the information (raw and/or processed, as it becomes available; however, not all information also needs to be delivered with priority. Only the first message, that the local traffic and public transport authorities need (maybe with an even higher priority, if possible). These guys need the first processed information ASAP.

Insurance companies will be interested in more long-term information, but they would still care about when it happened and be sure to have the most recent and up-to-date information. They may want more details about the accident - for example share information between the two insurers, based on cloud-maintained car registration databases. They may also want information processed at the Edge, that is very volatile, for example driver safety, other involved entities, eye witnesses and any other variables that could have influenced the investigation. These entities will most probably be interested in information that is processed and uncompressed. (while they may also want to store the volatile information for longer, to execute any kind of relevant functions on it, like, for example, corroborating the person driving with the number plates, car colour, any pre-existing conditions on the car that were or were not declared to the insurer before the accident.

Manufacturers will want the fastest and most compressed data, while keeping their most valuable volatile values for only as long as they determine that another, related failure could occur on the car. If that does not appear within the time limit, functions could also just send the data obtained to that point to the cloud and be finished.

Data evolution within the system due to the above associations and their services' restrictions:

The most important part is to get the first function output through LSH (the traffic authorities' processed alert message, indicating the location and time)

Afterwards, this can be part of different functions - for example, the local news broadcasters will want LSH done on some of the data, but not all of it. Thus, they could impose no LSH on specific processed information outputs, or, for example, if that alert message is also processed again, with data from another, local CCTV camera's view (that the new broadcaster has access to) or catches data through other functions on the repositories, that include these features (cars and accidents), it may want the EDRs to not corroborate these and get the data - after which it would either just decode the data (being the data/application provider of the recording app), or ask for decoding permission from the data originator (if the data/application provider for the recording app is different or the recording application is just the phone's - thus, the user is the actual owner)

Manufacturers will want to apply LSH to most of the data and within very specific time limits.

Insurers will want a combination between what the news and the manufacturers are interested in, leaving the more general data out, of course.

An equivalent system to the one presented in the Introduction of this thesis (Chapter 1) is an SDN-controlled system. Here, data names, labels and hashes, services and resources are manipulated instead of control tables in the network control plane. Similarly, ERP servers are controlled by policies dictated by and provide updates to ERP NOC(s), which set the main operating policies, analogously to SDN controllers, guided by a (set of) network-control application(s).

Within this thesis, we are mainly going to consider a few specific use cases, as follows.

1. The first case comprises of **sensing equipment** as producers (e.g. temperature and pressure sensors). These devices can produce small-sized data messages, at specific (often frequent) intervals of time and are generally static. The sensor data is normally ready to be processed and volatile. This device type would be most prevalent in **office buildings**.

2. The use case considered for the purposes of representing a realistic data load within a smart city is **video/audio streaming and larger, smart home, command-based data, from immobile Edge devices**. **Home** surveillance and doorbell cameras create large data feeds, for example. While this data may be critical, it is not necessarily provided continuously. Although it can be stored for longer periods of time after processing, it may be offloaded to clouds, as long as it is not solicited/required within a certain amount of time.

3. Another example of mobile Edge Data Providers (EDPs) generating small-sized data can be **measurement sensors within a car**. Such a sensor could be measuring battery charge, GPS location etc. [91]. This data can have shorter freshness periods, but may have longer shelf lives, for further network processing and availability. Each data point may be less important and more volatile than larger messages. Thus, this kind of data (or better yet, their processed results) could be offloaded to the cloud more often.

4. The most complex example of large-sized data could be **video or LIDAR**

data [91], created by a **public transport vehicle (e.g. Buses**, Underground). These kinds of data can normally have larger freshness periods, if they are to be processed. The shelf lives of these Data packets could be longer, due to their longer relevance period for Edge applications.

5. The last, and one of the most processing-intensive use cases considered within this thesis will be **NLP-driven voice command interpretation and execution**. With such (mostly smart-speaker implemented and smart-assistant-based, in either **home OR mobile environments**) applications spreading more and more these days, soon more resource- and energy-efficient processing solutions will be needed.

Consider that some IoT/Data/Application Providers and users will still have services established in the cloud, for non-time-sensitive requests and data processing. Thus, considering the large and relatively flexible reach of clouds nowadays, it is reasonable to think that there is at least one cloud provider within each area covered by WANs. Even if that is not the case, the closest cloud server to the ERP should be mapped (via NDNS, respecting cloud-ERP policies), as a constant resource to provide large processing pools, remote and persistent storage for remote and fast access to Users/Data Providers which are either located within that cloud provider's domain, or outside the Edge Repositories' domain.

Given the background reviewed in chapter 2, and the outstanding issues imposed partially by the above use cases and analogies, the new concept, of Edge Data Repositories (EDRs) was developed.

The algorithm that EDRs use employs structures normally used in SDN (tables and organisation) for systems control and a communication protocol that uses synchronous system updates and asynchronous requests and responses (as orchestration for table updates) to convey the outputs of the algorithm (and later inputs from the ERP servers). The EDR resembles a SNMP Agent and its associated resources and metrics resembling MIB objects, which would hierarchically be managed by a Network Management System (NMS), corresponding to the ERP servers, and according to certain hierarchical structures, according to the previously envisioned and explained NDNS mapping structure. The organisational structures providing the services in each EDR resemble ”match-plus-action” (control) tables. Thus, the control structures of EDRs should be SDN-based, while the communication protocol would be similar to SNMP (as explained above). This will be presented through the commands table and its associated algorithms, for processing and storage services, in Sub-section 3.4.5.

The results necessary in algorithm development mainly depend on the type of EDP. The producers are diverse in their movement patterns and service needs. We use the *example use case* to establish 6 different service models for each of the first 4 use cases presented in Section 3.4.1 for Edge Data Repositories. One (6th) is a common, ”golden standard” model, used for QoS comparisons. All repositories in each scenario provide one service, throughout each simulation scenario and model configuration. Note that storage capacity was reduced to 5GB/EDR in these assessments.

3.4.3 Setup and Assumptions

An EDR domain was implemented and evaluated using an extension[1] of the Opportunistic Network Environment (ONE) simulator[2] [94]. The following assumptions were used for testing and evaluation.

• processing power was measured in threads, assuming equal power in all threads [95]

• one thread uses a specific amount of time for a service type

• function execution takes a finite amount of time per content type, assuming execution sessions established

• for simulation efficiency, the smallest step of processing time is 0.1s (realistically it could be smaller)

• EDPs, EDR cluster, Data Providers and Cloud Providers are securely bootstrapped, for data and function verification

• all EDPs and EDRs use their private keys and pre-fetched certificates/trust anchors

[1]The ONE Repos: https://github.com/Chrisys93/the-one-repos

[2]The ONE: http://akeranen.github.io/the-one/

to produce secure data

• freshness period accounts for only one processing cycle

• under service strain, the retrieving EDR will compress and send processing messages to the cloud before processing

• compression is executed on individual messages, and not on the aggregated messages, as a file, for ease of simulation

The EDP generation rates used in this paper were based on CISCO predictions [17, 91] for 2022-24, several case studies ([96, 97, 98]) and reasonable approximations. Average data production is 85GB/node/day and varies with type of EDP.

Example use case - An EDR cluster with the specifications:

• a total upstream bandwidth limitation of 10Gbps

• 500 cars, 40 pedestrians and 15 buses served

• three hours accounting for peak traffic hours of the day

• all data required on mobility patterns, generation rates and served file types known

• the cluster deployment area considered to be Helsinki city

The example case domain/scenario is comprised of 80 repositories, distributed in a cell-like structure covering central Helsinki. The mobile nodes generate messages as follows (non-processing:processing): 2:0 per pedestrian per 5s, 0:2 per car per 2s with a processing delay of 0.1s and 1:2 per bus per 2s [99] with a processing delay of 0.2s. Messages are 1MB in size. Each EDR serves 10GB storage capacity, 10 parallel processing threads and a 10MB/s maximum output bandwidth. Although the cars are the main type of producers studied in this case, all nodes generate realistic network traffic.

3.4.4 Main Design Parameters

Assessing the processing and storage capabilities and associated QoS, we can ascertain three different ratios that can be used as system metrics: fresh processing rates, shelf-life processing rates and ingress traffic.

Fresh processing rates refer to the capability of the EDR to process all ingress messages within their freshness periods. This is obtained by scaling the approximated per-message processing delay with the freshness period. **Shelf-life process-**

ing rates are used for determining the total processing capability of EDRs. It is assessed in relation to the shelf-life of processing messages and the associated *fresh processing rate*, in the context of the type and expected processing delay of the ingress serviceable messages. The **message influx**, as the name indicates, is a measurement of the ingress processing message traffic targeted at the respective service.

The storage mode setting is one of the parameters under EDR control that can directly affect QoS and resource efficiency. However, more importantly, **the shelf-life** associated with **the size and type** of any ingress messages being accepted as part of a service model are the determining factors of storage usage and storage QoS. Most importantly, the common parameter which affects both storage and processing, **the message influx** has one of the highest impact factors to be considered in our algorithm. This was seen in previous results, which were considered too trivial and simplistic for representation.

Note that **upload limits** put a limit on the amount of messages serviceable in a specific amount of time. However, upstream bandwidth can be adjusted by changing the compression rate(s) and compression-to-deletion ratio(s) associated with the service(s).

This assessment demonstrates that some service virtualisation, through VMs or ”enclaves” can be offered with no costs to QoS, for a variety of devices and their users and/or providers. This, however, would either come at the cost of higher latencies and/or computing resource usage in the former case or at the cost of less flexibility in service variety and device support in the later case. This represents another, important trade-off with regards to Edge computing efficiency, which has been extensively studied. [100, 101, 102, 103]

The main commands to potentially be used within this system are essential for the management communications in question, and can be very similar in nature to certain protocols like SNMP; mainly because of the architectural similarities envisioned at a higher level, and the way the EDR control plane is set up, however they are less related to routing and more related to the distribution on monitoring metrics and the management of both the associated data and the computational/storage configuration. Thus, I took the freedom to (re)create such commands for our system's requirements. Table 3.1 is provided for reference.

Command Name	Actors	Description
Performance Update	server to EDR	Periodic request for QoS update
Heartbeat	EDR to server	Periodic update on system performance, configuration and resource usage
Configure	server to EDR	Change MIB object values
Trap	EDR to server	Asynchronous Warning or Notification (& suggestion) message
Response	EDR to server	Response to server

Table 3.1: Table of algorithm generated commands

3.4.5 Algorithm

An algorithm for EDR management will now be presented, providing insight to the process that requests and data undergo once entered into the data lifecycle process ensued by the EDRs' store-process-send mechanics.

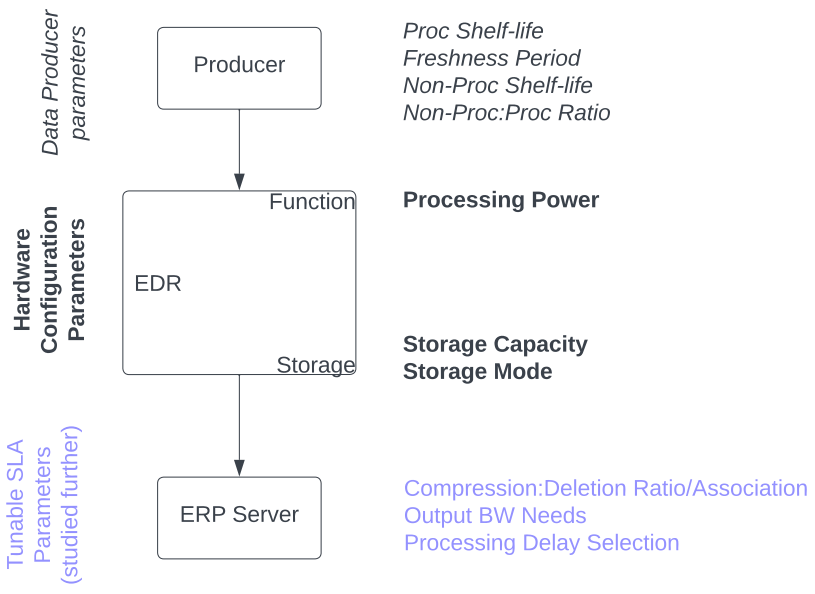


Figure 3.3: Main structure of algorithm

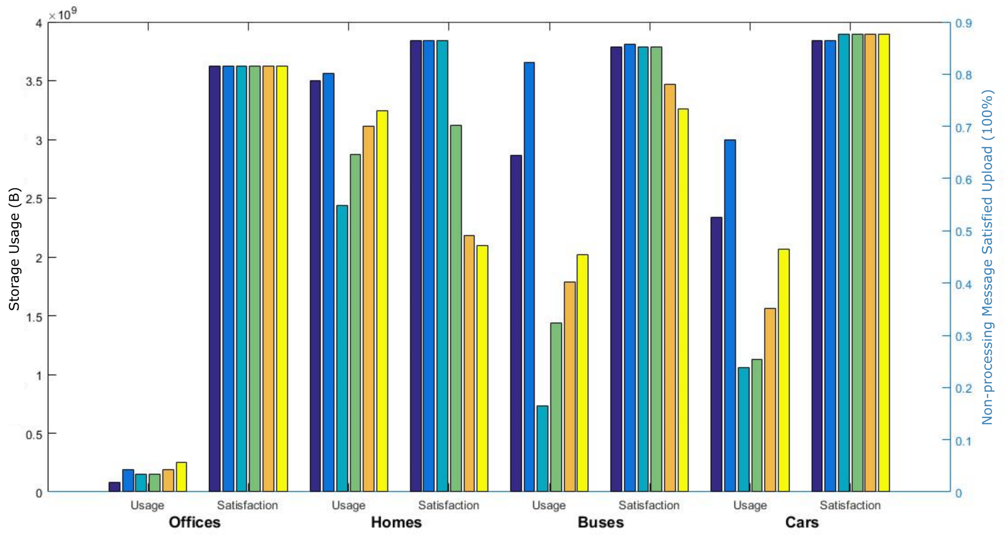
The algorithm outputs **(warnings, notifications, performance updates and heartbeats)** can be used to determine needed changes, in accordance with the existing data types and service model(s). The algorithm is meant to adapt resources and give the feedback needed by the ERP server to dynamically apply different service- and resource-based optimisations within the cluster. A representation of this communication process is also provided through Fig. 3.3.

The point of this algorithm is to provide an SDN-like management structure, to operate just above the network layer, depending on individual EDR performances (diagnosed through application-layer messages like the ones in Table 3.1). **With the server having its own view of the EDR cluster, it can optimise manageable object values in MIB records and service provision tables, for adapting the system and services for the network environment.** In other words, EDRs send heartbeat and ”trap”-like update messages to ERP servers. Depending on their cluster-wide view of the EDRs. This algorithmic and architectural structure creates the opportunity for ”ERP Servers” to (potentially automatically/autonomically) decide on what actions to take, to optimise service placement, for performance and QoS. Through configuration messages, ERP servers could provide routing and resource configuration updates to any EDR as a consequence of its cluster-wide decisions thereafter. Following these actions, the feedback process could restart.

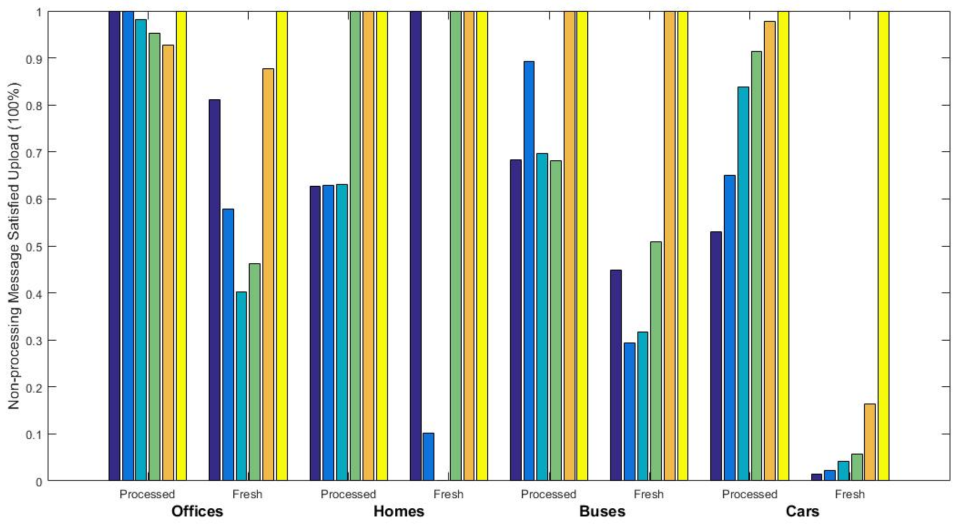
3.4.6 Evaluation of EDR Performance

As sub-figure 3.4b shows, the processing success rates depend on the scale of the freshness period and shelf-life, compared to the function execution times (processing delay), which are generally very small. The ratio between the first two has to be close to 1 to have any effect on the success rates. Thus, we consider the traffic, delay, freshness period and shelf life, as the most relevant parameters for dynamic adaptation and processing effectiveness. Thus, these two metrics are fed back to ERP servers, in order for the system to act on and/or influence the above four parameters, towards better feedback on the next loop through the algorithm.

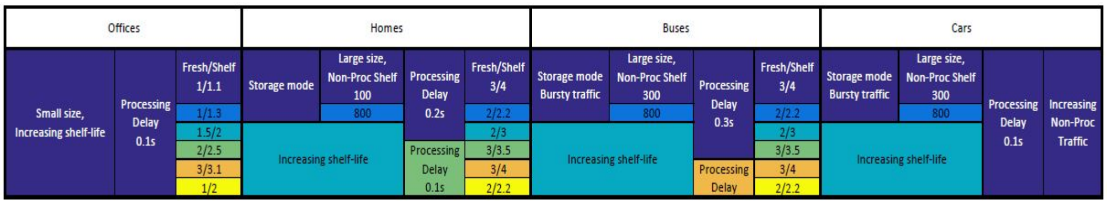
Moving on to the parameterisation of processed message freshness and keeping the focus on sub-figure 3.4b, we can infer that the main decision parameter



(a) For all use cases, the bars represent: EDR Storage Usage Expressed in Bytes on the left; and QoS Expressed in Percentage of Messages Uploaded to the Cloud, that were stored and kept for their Shelf-life on the right



(b) QoS Expressed in Percentage of Messages Processed within associated Freshness Periods - Y-axes indicating percentages for all use cases, for both processed messages (within the shelf-life, excluding ”fresh” messages) and fresh messages (processed within the freshness period)



Offices			Homes				Buses				Cars			
Small size, Increasing shelf-life	Processing Delay 0.1s	Fresh/Shelf 1/1.1	Storage mode	Large size, Non-Proc Shelf 100	Processing Delay 0.2s	Fresh/Shelf 3/4	Storage mode Bursty traffic	Large size, Non-Proc Shelf 300	Processing Delay 0.3s	Fresh/Shelf 3/4	Storage mode Bursty traffic	Large size, Non-Proc Shelf 300	Processing Delay 0.1s	Increasing Non-Proc Traffic
		1/1.3		800		2/2.2		800		2/2.2		800		
		1.5/2	Increasing shelf-life			2/3	Increasing shelf-life			2/3	Increasing shelf-life			
		2/2.5			Processing Delay 0.1s	3/3.5				3/3.5				
		3/3.1				3/4			Processing Delay	3/4				
		1/2				2/2.2				2/2.2				

Figure 3.4: Bar charts for different simulation use cases with general node configuration

changes to the processing delay. This inference is explained by the trends shown in the ”Fresh” measurements of all use cases, in parallel to the legend-based table of values below it. However, this ratio is a potent modifier of the impact that the

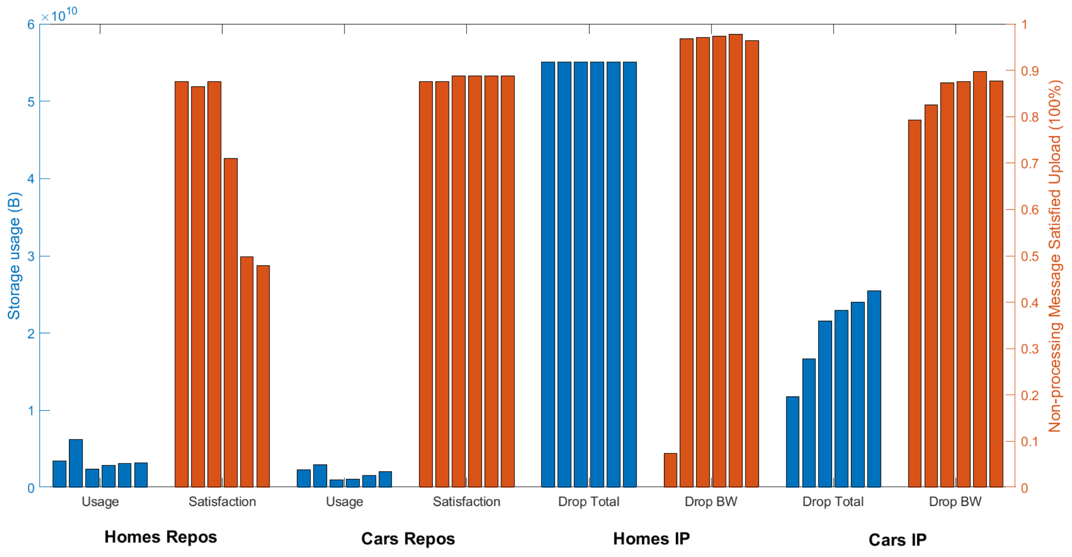


Figure 3.5: EDR Storage usage and satisfaction rates (left two sets of bar plots) vs IP total package drop sizes and Drop BW/Useful BW percentages (right two sets of bar plots)

ratio between the delay and the difference between shelf-life and freshness period has. We can also note that the most important modifier of both processing success and freshness rates is the ratio of input non-processing to processing messages, representing ingress traffic.

Clearly, when more storage is used, messages' shelf-lives limit satisfaction rates. Considering this, the shelf-life associated with a specific service determines its satisfaction rates and whether the receiving EDR storage overflows. This naturally also depends on the size of messages and traffic. More exact values for this tipping point of one of the models, as shown in figure 3.4a, are: between 500 and 600s non-processing shelf-lives, processing shelf-life under 50s, a rate of 1 processing message for each 4 non-processing messages generated each second, a size of 1MB/message and static EDPs.

In the last three models of the cars and buses use cases we can see that, as shelf-life increases, the importance of the ”non-proc:proc” traffic ratio decreases with respect to storage occupation. On the other hand, we can see that when sizes of messages are bigger and EDPs more stable/static, the importance of the ”non-proc:proc” ratio is much higher. In this case, the more positive the ratio, the higher the chance for messages to be stored less than their shelf-life values (above a specific threshold - see Fig. 3.4a).

From this figure it can be inferred that shelf lives determine the maximum storage capacity required for a service. Thus, *Storage Mode* is used to keep messages within storage in underused repositories, for both efficiency and better storage service provision. The number of messages stored longer than their shelf-life (overtime) is normally low, unless repositories are configured in storage mode. If messages are stored overtime, however, storage is used inefficiently and storage satisfaction may decrease, if used inappropriately. Thus, storage mode is a valid and useful option only for services with reduced or infrequent bursty traffic.

As also demonstrated in figure 3.5, the implementation of storage drastically outweighs the advantages of just using processing within the IP environment. This is demonstrated by the amounts of data (potentially) being dropped at the repositories' routers, without the implementation of storage and only the implementation of Edge processing within DTN (with the same sizes of buffers).

3.4.7 Better Storage Management

Upon further consideration of the above-presented work, the low-level feedback algorithms, that were not fully evaluated in the above work's evaluation and a few other related works, more work was pursued, towards a better, more complete storage management solution. This paper covered the aspect of data distribution efficiency within the EDR storage environment, towards the system's improvement in performance, with regards to QoS and system resource allocation efficiency.

This paper's concept was mainly based on the EDR environment's design, evaluated above. Some utility of this work will be demonstrated in this design and evaluation, by directly borrowing the main systems' functionalities. These functionalities were used in the next system's implementation to develop a more comprehensive servicing and storage system. The new design was named SEND, and its main purpose was to provide Storage at the Edge, for Networked Data (StorEdge Networked Data).

A realistic evaluation study of SEND was conducted. First, by implementing an EDR prototype on top of the Google file system [44], evaluated based on real-world datasets of images and IoT device measurements/readings [104, 105].

Afterwards, to scale up the system, an evaluation based on network simulations was developed and implemented, based on synthetic and real-world datasets, to evaluate the performance and trade-offs of the SEND framework as a whole. Our experimental results demonstrated that SEND achieves data insertion times of 0.06ms-0.9ms, data lookup times of 0.5ms-5.3ms, and on-time completion of up to 92% of user requests for the retrieval of raw and processed data.

3.5 Concluding the first step in defining the EDR environment

The assessment completed using the ONE simulator and its results produced a local management and feedback algorithm. This algorithm provides a first insight into the dynamic management potential of ERP systems. Through the feedback provided, it was demonstrated that the development of an EDR cluster management algorithm for the ERP system can be advantageous for different network conditions and certain service offerings. These concepts shall be developed, to form a more complete system, accounting for more aspects of networking not covered in this paper, like content-driven networking, security, commercial and management policies.

These evaluations are important as they evaluate which trade-offs should be considered more important for different service models. We also provide the feedback algorithm, for design and adaptation of systems that integrate EDR clusters. This has potential for further development, accounting for further complex parameters and metrics (e.g. data aggregation and locality). There is a wide range of developments which can be based on this system, depending on their final purpose of deployment.

Chapter 4

Store-Edge Networked Data (SEND) - Storage Management at the Edge

4.1 Introduction

IoT Providers can rarely have their own established networks, especially if their nodes are mobile and unless they want to invest into a whole storage, processing and routing infrastructure. IoT Providers generally use third party Cloud storage and computing resource services, instead of their own servers. However, there are some providers which have their own servers or even data centers and prefer just paying for ISP links and the associated traffic.

Consider that content/application providers like Google, Netflix, Facebook etc. provide their services strictly via DNS and CDN based private networks. The performance of such a large network would be greatly improved by partially moving these application layer network mechanisms to the network layer, in a content-based network model. A further improvement to this would also be to now account more for the reverse flow of information (from users/Edge devices towards the core/data centres), at both the application and network layers (creating a more interwoven interaction between the two layers). Lastly, for even more intelligent, efficient and effective networks and systems, computation-reuse- and storage-based orchestration strategies could also be implemented. Such implementations would also come with the great potential of both financial gain and economic innovation in the networking domain.

With the explosion of the Internet-of-Things (IoT) [1], this model is reversed [2]: massive amounts of data are generated by user devices at the edge and flow towards the core of the network infrastructure, for processing and storage purposes. To alleviate the pressure on the network introduced by this reverse data flow model and to achieve low-latency data processing, edge computing emerged as a prominent paradigm that makes computing and storage resources available at the edge of the network [3]. In this context, storing and effectively managing massive amounts of generated user device data at the edge is an issue that needs to be addressed [1]. This issue becomes particularly challenging, since the generated data may be of different types (e.g., IoT sensor data, images, video frames) and may also have different needs and purposes (e.g., to be processed, to be simply stored at the edge, or to be eventually offloaded to a cloud). To address this issue, this chapter proposes a data storage and management framework at the network edge, called "Store Edge Networked Data" (SEND).

Persistent networked storage is different from cache storage, in that in persistent networked storage data is not only session-less (the destination/purpose of the data is not determined by only one application/user connection session or stream), but it is also network-agnostic. In persistent network storage, data can be used in a publish-subscribe manner, and it can be specific to more than one application provider/application user/service collection and/or (network or data) utilisation policy.

SEND aims to improve the performance of the network edge and the Quality of Service (QoS) provided to users by keeping raw (generated by user devices) and processed (output of processing services/functions) data in persistent storage, close to users, for periods of time longer than cache storage. In SEND, storage servers, called Edge Data Repositories (EDRs), are deployed at the edge and seamlessly interact with flows of network traffic. EDRs accept different categories of data (e.g., data to be stored at the edge, the data input(s) and output(s) of processing functions running at the edge, the code of the processing functions) that may transit through the edge environment. Through a logically centralized management plane, SEND takes advantage of the characteristics (attributes) of the data generated at the edge to make decisions on where to store and whether/how to replicate the data across different edge locations in order to maximize the offered QoS.

This chapter makes two main contributions:

- The design of SEND is presented, which takes into consideration different criteria (e.g., data popularity, proximity of data to processing functions running at the edge) to improve the placement of different categories of data at the edge. To aid data placement and management decisions, SEND relies, among other attributes, on system-wide identifiers of the data context, called labels. Labels are used to tag the data that enter the SEND system, offering a light-weight mechanism for identifying the data type(s), the data generation sources, and the processing functions supported. SEND also stores data at EDRs for as long as the data may be useful at the edge (e.g., for processing purposes), offloading the data to the cloud for archiving once they are not useful at the edge anymore.
- A thorough evaluation study of SEND is performed. First, an EDR prototype is implemented on top of the Google file system [44], which is evaluated based on real-world datatets of images and measurements/readings of IoT devices. To scale up the experiments, a network simulation study is performed, based on synthetic and real-world datasets evaluating the performance and trade-offs of the SEND framework as a whole. Our experimental results demonstrate that SEND achieves data insertion times of 0.06ms-0.9ms, data lookup times of 0.5ms-5.3ms, and on-time completion of up to 92% of user requests for the retrieval of raw and processed data.

The rest of this chapter is organized as follows. In Section 2.10, a brief background on storage frameworks is presented and prior related work is discussed. In Section 4.2, the SEND system model and assumptions are presented, while in Section 4.4 presents the design of SEND. In Section 4.5, the experimental evaluation is done, and finally, in Section 5.7, we discuss various considerations and extensions of the SEND design, with the help of which the work presented in this chapter is then concluded.

4.2 System Model and Assumptions

As illustrated in Figure 4.1, it is assumed that an edge computing environment, with a number of EDRs (EDR_1, EDR_2,..., EDR_N) have been deployed one-hop away from user devices. It is further assumed that the devices generate data based on their nature or the applications that may condition their data generation patterns and formatting (e.g., periodic temperature readings for an IoT sensor, video frames captured by a mobile device running an augmented reality application that requires real-time object recognition). Once this data is offloaded by a device and is received by an EDR, it can be stored at the edge for future processing or low-latency access, or it can be processed right away and the processing results can be stored in the EDR environment. As Section 4.4.2 shows, each data piece may be useful (e.g., as an input of a function at the edge) for a certain period of time. After this period of time, the data can be archived from an EDR to the cloud.

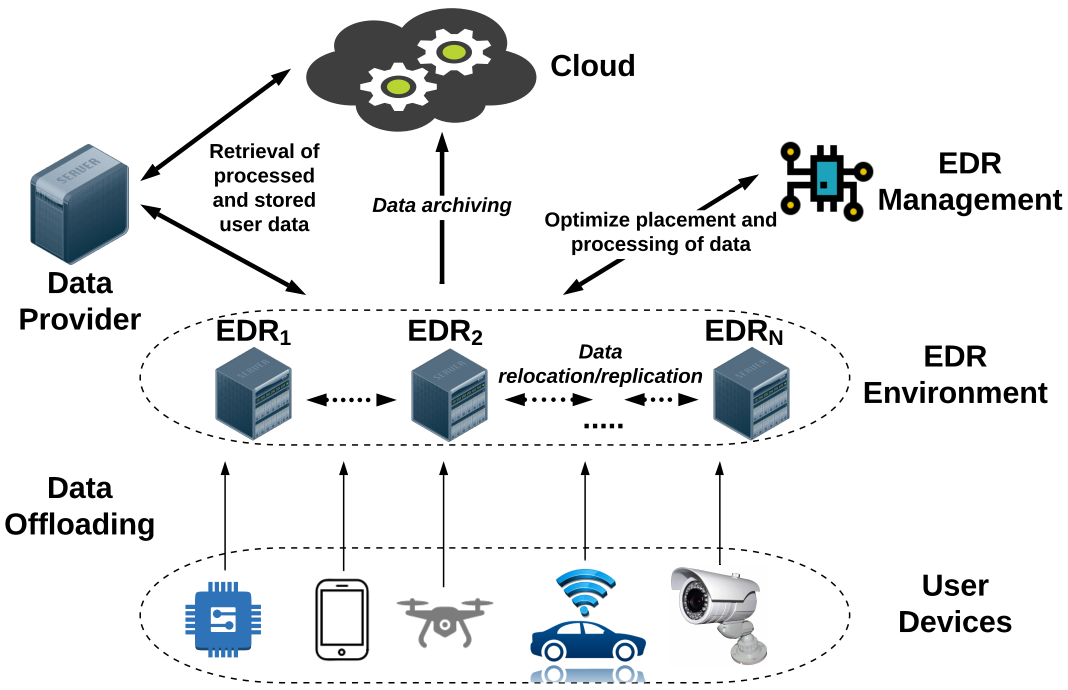

Cloud
Retrieval of processed and stored user data
Data archiving
Optimize placement and processing of data
EDR Management
Data Provider
EDR1
EDR2
EDRN
Data relocation/replication
EDR Environment
Data Offloading
User Devices

Figure 4.1: SEND System Model. Data offloaded by user devices is stored in the EDR environment, which is managed by a logically centralized EDR management entity. Data providers can retrieve raw and processed data from EDRs or the cloud once the data is archived from the edge to the cloud.

It is assumed that a number of processing functions (e.g., object recognition, data analytics) may be offered by an EDR and that all requested functions will exist in the EDR environment. Functions may be pre-installed by an EDR or a data/application provider. Alternatively, the function code can be offloaded directly by user devices if their hardware capabilities allow. The instantiation of functions will take place through the use of Virtual Machines (VMs) within an EDR, while the VM hardware allocation in "cloudlets" of EDRs is elastic. A function consists of application-level code, applied to input data, towards obtaining a result (output data) requested by either a data provider or a user registered with a data provider. Data providers may be corporations, such as ZipCar, Uber, Nest, Ring etc., which utilize edge-provided (EDR-provided) services and network data storage for the data needed by them or their users.

In the context of the SEND system, as in the previous case, the existence of a logically centralized EDR management module (ERP server or NOC) at each edge network is assumed. Each EDR periodically sends statistics about the stored, processed, and served data and their attributes to the management module, which is responsible for making data management and placement decisions. Based on these decisions, EDRs may relocate or replicate data within their environment. Finally, data providers may interact with the cloud and the EDRs to retrieve raw or processed data generated by their users for further processing and insights.

4.3 Architecture key points

Why Hashes?

The view of ICN is adopted within this architecture, and more specifically, hash routing is mainly used here, for its advantage of being able to associate and encode data with message-specific hashes, making tracking of the data possible and having the advantage of being able, through the use of mathematical techniques, to encode other interesting parts into these hashes and associated packets (e.g. Meta-Data and validity identification mechanisms). Furthermore, due to not being limited to specific names (like NDN), the data can also be context-sensitive, as the hash-encoding could even mean different things for different applications interpreting the data-and-context-based hashes.

Labels: Their purpose and use

As hashes themselves can only identify one data packet/message, they can be inefficient from a context point of view. Thus, the system architecture needs a way to find all relevant and ”interesting” data, from a context and/or hierarchical point of view, regardless of the type or origin of the request/data/service instance.

Data production and Function provision should both be either provided or approved and known by the data producers’ (mobile or IoT - mainly - end-systems) commercial managers (Data/Application Providers). Edge Data production and Function provision should be independent of each other, at the same time. Cloud data can be of either type, or of the result type (combining the previous two), so it has no impact on hashing. The main ”glue” between producer data and the rest of the Internet (and functions) are the repository environments, which process and distribute the data according to resource efficiency, commercial and provider needs. The hierarchical and commercial part are mapped and used (”glued” together), in an analogous way, mainly by the ERPs’ management servers. All of this is achieved through the introduction of labels. As discussed in the previous chapter, ”Labels are a data characteristic flexible enough to be attributed to all types of data entering this system.” This means that labels can be used for the general purpose of tracking any kind of data within the EDR environment and, at the same time, provide insights to automated management entities, to provide fast and effective feedback to EDRs, effectively increasing system performance with regards to both QoS and resource allocation efficiency.

4.3.1 Secure data-function-request generation and deployment processes

The following processes (each flow progress from left to right) are supposed to have three (possibly four) main parts:

- the packet-/data-generating device
- the data generated
- the hash function
- (the key and/or label provider)

Note that the labels are part of the process **within** the generating device and do not come directly from the Data/Application Provider; they are bootstrapped/provided in the same way as the hashing keys

Data packet generation process design:

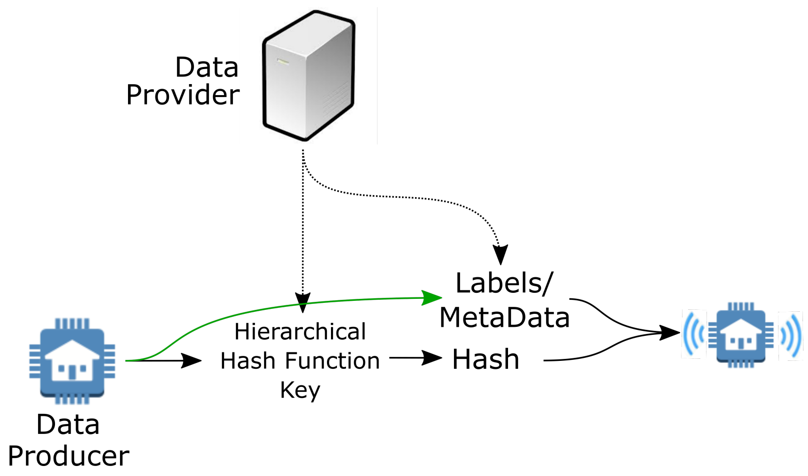


Figure 4.2: SEND Data Production

Data packets are produced as soon as the (collection of) measurements are taken and have to be transferred for processing within the network. The hashes are quickly and easily computed; attaching the appropriate (readily-available) labels to the packet as MetaData. As the green-lined arrow indicates, if the Data Producer produces more types of/function-associated data, or with different restrictions, its Labels/MetaData the data packets will be deployed so that they have different attributes.

Function (function code to be executed, being used within EDR environment, bootstrapped and present in all EDRs) process design:

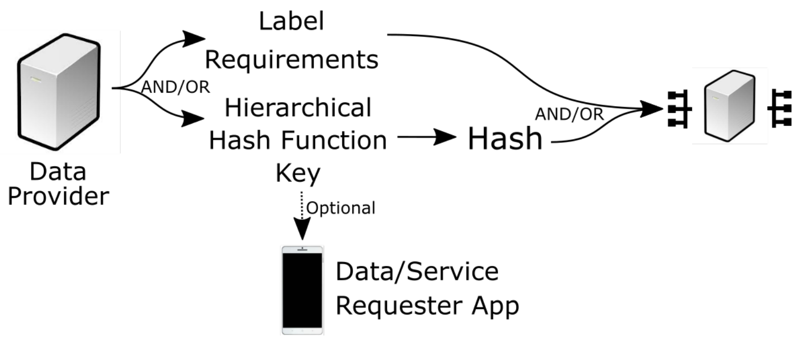


Figure 4.3: SEND Function Production

Function generation is done similarly to data production. However, in this case, the code has to be compiled to be executed on specific (collections of) data and, thus, can include any combination of requirements, like a combination of labels, data hash(es) and/or only one of these. Request packet generation process:

Request generation process design:

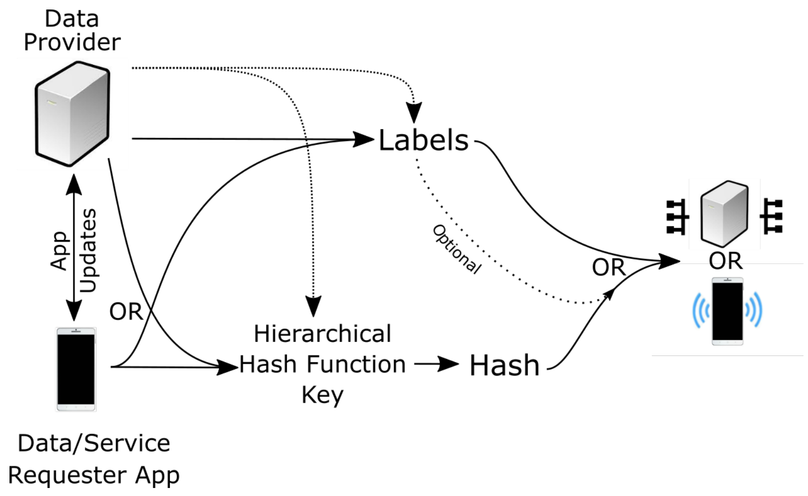


Figure 4.4: SEND Request Production

Processed Data can be requested via the request of specific labels and/or data combinations, by either the Data/Application Provider or the provider's users, via its API; or, indeed, any device capable of running the Provider's API (and boot-strapped/entitled/authorised to do so). For all the above, the densely dotted line arrows indicate that the labels and keys are bootstrapped/provided (periodically) by Data/Application Providers For all the above, the continuous, thick lines indicate data transfer/necessary data processing steps

4.3.2 Data and label processing within the EDR Environment

The schematic in Figure 4.5 shows how the labels and data generated as presented above are processed and used once in the EDR environment, towards providing better QoS and improving the EDR environment efficiency, at the same time, using the data it is provided with and the strategies presented in the second proposed paper. The design of this process is presented in the following section.

4.4 SEND Design

In the way that EDR domains are managed, provide their services and considering the nature of the services they provide, the EDR environment represents the ”glue” between most of the identifiers, applications/providers, data and functions, securely executing approved functions on data provided from the Edge and providing a final, linked Data object, which can not only be verified and cached throughout the network, it can also be reconstructed, with the right encoding, decoding and security applied. Furthermore, these actions and data processing show how the commercial and organisational/ISP/network provision hierarchies, both have a ”home” in EDR environments. EDR domains provide a type of information hub at the Edge, where Edge Data is produced and stored, different functions are securely executed and stored and where companies can collaborate and produce different service policies, in conjunction with the ERPs.

An overview of the SEND design is first presented here and, subsequently, the design components are described in detail.

4.4.1 SEND Design Overview

In Figure 4.5, an overview of the SEND design is presented, with its operational workflow. As illustrated on the left of Figure 4.5, SEND includes a management component that considers stored data and available functions. This component makes function instantiation decisions based on the location of the stored data and the execution locations of available functions. All of the above are taken into account when new or previously stored/processed (e.g., associated with known labels) data is introduced to the system. As a result, data is relocated (or replicated) to EDRs (edge locations) where it is likely to be needed in the future in order to satisfy user requests for the execution of functions or for the retrieval of the data itself.

As illustrated on the right side of Figure 4.5, SEND interacts with network flows, accepting data that transits through the EDR environment. SEND separates data transiting edge networks, at least upstream (i.e., from the user devices to the network core), into: (i) function-related data that may represent the actual code of processing functions running at the edge or input(s) and requests for the execution of such functions; and (ii) storage-related data that is simply stored at the edge for future use. The SEND design is driven by attributes that can be common among different categories of data (e.g., size, generation timestamp, hash). Each piece of data is stored at the edge for as long as it may be useful to users and applications.

Note that all data attributes are considered important, however, the most important attribute for the SEND design are the labels. Labels can significantly contribute to optimizing: (i) data placement and replication among different storage locations at the edge; and (ii) the execution location of available functions.

In the following subsections, the (other) components of the SEND design (that were not presented) will be presented in more detail. Different **data attributes and categories of data as defined in the context of SEND** willbe presented first (Section 4.4.2). Then, the improvement on data storage and servicing with **the most important data attributes (labels and hashes)** will be explained (Section 4.4.3) and a description porovided for **the SEND data management operation and different strategies for the placement of data at the edge** (Section 4.4.4). Finally, all the components of the SEND design will be put together, to elaborate on **the overall operation algorithm of EDRs** (Section 4.4.5).

4.4.2 Data Attributes and Categories

Data attributes: In SEND, EDRs consider a number of attributes for the data, such as the data size, generation timestamp, hash, freshness period, shelf life, and labels. The freshness period and the shelf life are presented in the above evaluation, of the EDR environment. In the case of the SEND system, the data shelf life is more important for the decision needed to (ex)change data storage locations. As above, when the shelf life expires, the data is archived to the cloud. These attributes also help with data management decisions such as storage time, processing needs, and storage after processing.

Labels are a data characteristic flexible enough to be attributed to all types of data entering this system. This attribute is flexible, while lightweight, so that it does not impose high overhead on the network on the system. It can, however,

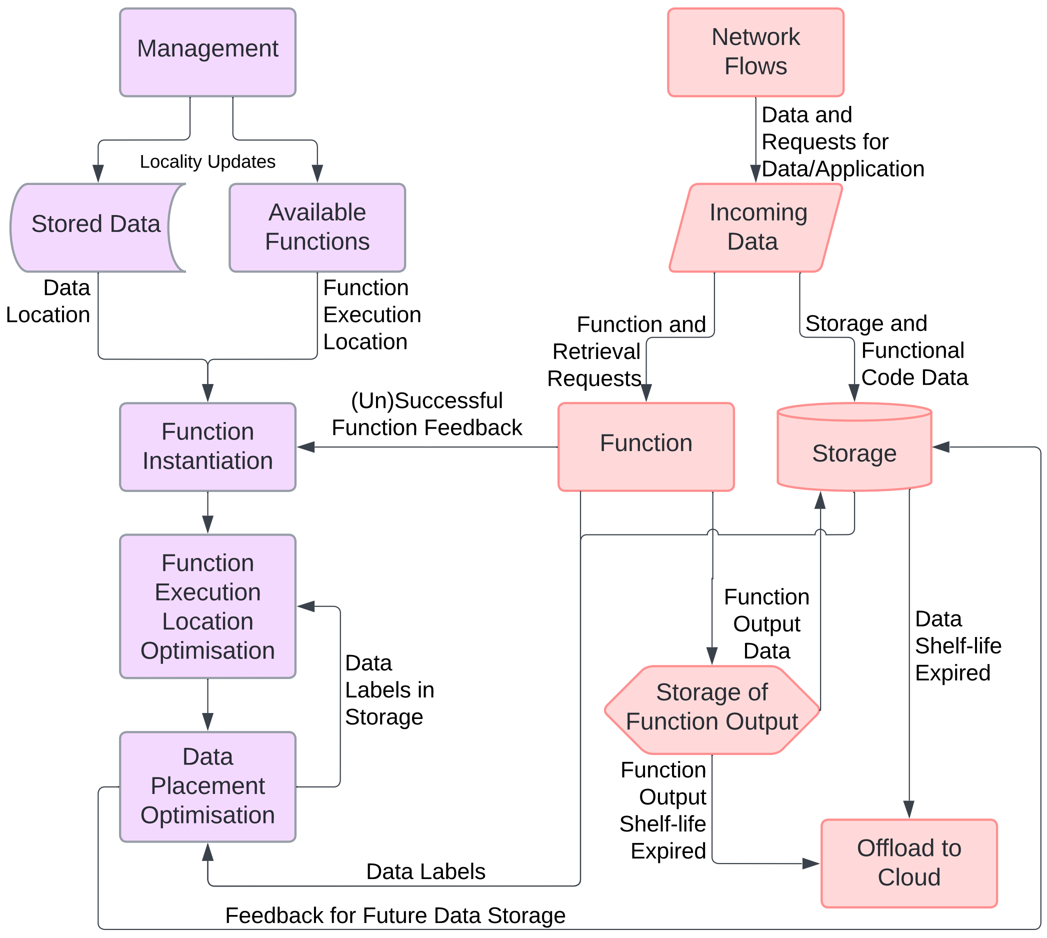

Management
Locality Updates
Stored Data
Available Functions
Data Location
Function Execution Location
Function Instantiation
Function Execution Location Optimisation
Data Labels in Storage
Data Placement Optimisation
Data Labels
Feedback for Future Data Storage
Network Flows
Data and Requests for Data/Application
Incoming Data
Function and Retrieval Requests
Storage and Functional Code Data
(Un)Successful Function Feedback
Function
Storage
Function Output Data
Data Shelf-life Expired
Storage of Function Output
Function Output Shelf-life Expired
Offload to Cloud

Figure 4.5: SEND Design Overview. SEND seamlessly interacts with network flows to accept function- and storage-related data. Raw and processed data is stored at the edge until its shelf life expires. After shelf life expiration, the data is archived on the cloud. The SEND management plane receives feedback from EDRs on labels of received data and successful/unsuccessful on-time completion of function requests to optimize function and data placement at the edge.

also have a more sophisticated form that provides deeper context into the data. For example, temperature data generated by an IoT sensor can be labelled based on its type (e.g., “temperature”), the application that generated the data combined with the type (e.g., “home-IoT-app/temperature-data”), or its generation location and context (e.g., “beautiful-city/beautiful-neighborhood/house123/temperature-data”). Each data piece can also carry multiple labels if it is relevant to multiple functions. For example, let us assume that a driver uses a mobile navigation application while driving to find the fastest route to his/her destination. Data about the speed of the driver's vehicle reported by this application may be useful to multiple functions at the edge; for example, a function that reports accidents and traffic conditions to local authorities and a function that optimizes the routes for users of this application. To this end, the vehicle speed data may carry two labels (e.g., "speed-accident-reporting" and "speed-route-calculation") to associate the data with both relevant functions.

SEND offers the flexibility to application developers to define their own labels and select the values of attributes, such as the freshness period and the shelf life. **Note that these are application-specific labels and MetaData, that is interpretable in different ways, depending on the application processing it and the data owner's different associations with it.** Alternatively, SEND itself can set the values of these attributes based on statistics collected over time about applications, the data these applications generate, as well as the usage of functions.

Main data categories: SEND considers three main categories of data for its operation:

• Storage-related data: This data has the sole purpose of being stored at the edge. Its future use may vary. For example, this data can be used as input for the future execution of functions or be stored for distributed and time-constrained access.

• Function-related data: This data may be the actual function code, requests for the respective function's execution or data to be directly used as input for the execution of a function (typically associated with a short freshness period).

• Data to be offloaded to the cloud: This data has already been used for as long as it was valid (passed its shelf life), therefore, it is not useful for any further operations at the edge. To this end, such data can be offloaded to the cloud for archival purposes. Both storage- and function-related data (in addition to processed data, such as function outputs) will fall into this data category once their shelf life expires.

4.4.3 Labels and Hashes to Improve Data Storage and Servicing

The goal of the SEND design is to improve system performance by keeping data in persistent storage for periods longer than cache storage. These improvements come with the help of the data management algorithm and the data placement strategies (Section 4.4.4) with the goal of storing data closer: (i) to functions that may need the data for processing in the future; and/or (ii) to users and devices that may request the data in the future. The management algorithm and the strategies make use of the data attributes to make informed decisions. Among the most important data attributes are data hashes and labels, which benefit SEND as described below.

Data hashes: Data hashes offer the ability to track data from its source to its destination. The hash of each data piece is immutable, that is each data piece retains its hash after replication or after it is used as input to a function. Hashes can be used to, for example, find the origin of certain data pieces or verify the validity of certain data pieces.

Data labels: The labels offer the ability to track performance, the popularity of data, and data placement statistics. For example, if SEND determines that data labelled as "video-data" is popular in a certain edge region, it may decide to place more data characterized by the same labels in that region, or, for instance, diversify the data in the region, while keeping the "video-data" label as the predominant label. This is the reason why the labels are the **most essential** ***data attribute*** in SEND. In other words, labels make the system aware of the data context that it handles and improves its performance by enabling accurate data placement and storage decisions.

4.4.4 Data Management and Placement Strategies

Management entity: Data management decisions in each edge network are made by a logically centralized management entity (e.g., an SDN controller), which hosts the management intelligence of the EDR environment. The management entity periodically receives up-to-date information about the labels of the data recently received by EDRs, the recently executed functions, and the recent EDR performance. Based on this information, it makes decisions about data and function placement and provides feedback to EDRs about these decisions. Each EDR takes into account the feedback provided by the management entity to relocate/replicate incoming and stored data, available functions, and function requests in order to maximize the system performance and QoS.

More specifically, the decisions of the management entity of each EDR environment can determine the following:

1. Function placement: where (on which EDRs) to instantiate functions in an edge network, so that the offloaded user data can be processed before the expiration of its freshness period. (determined by a predetermined algorithm, defined by [36], subject to the appropriate data being present in the EDR)
2. Data placement: where (on which EDRs) to relocate/replicate stored data based on a data placement strategy (explained below) to satisfy different requirements. For example, data deemed as highly likely to be used for the execution of a function in the future may be pre-fetched by an EDR to ensure that it is co-located with the function code.
3. Routing process: how to route requests for data towards the closest copy of the requested data or requests for function execution along with the needed input data for processing towards the closest instance of the requested function.

The statistics, metric, control and monitoring-type data needed for the above can be implementation-specific, however, some common ways to transfer such data can be SNMP, normal, small TCP/IP packets or even up to REST API messages (potentially on dedicated, management network lines).

Data placement strategies: The management entity executes a data placement strategy, which determines the locations where data should be placed at the edge, whether data needs to be replicated across multiple locations, and for how long data should be stored at the edge. Subsequently, the strategy instance running at the management entity provides feedback to an instance of the strategy running at each EDR, which helps EDRs decide how to handle incoming data. Examples of data placement strategies offered by SEND are the following:

General-purpose storage strategy: Priority is given to storing and replicating the data introduced to the system that has certain values for a set of attributes as data already present in the EDR environment. For example, if data with a label "video-data" and a generation timestamp within the last hour has been already stored, data with the same label and generated within the last hour will also be given storage priority on the respective EDR.

Popularity-based strategy: The data is placed at edge locations where their associated label(s) is considered to be popular (i.e., the rate of user requests for data associated with such label(s) exceeds a certain threshold). For example, if higher numbers of requests for one or a number of labels are present in a part p of the EDR environment, data that carry one or more of these labels will be replicated or relocated towards EDRs available within p.

Function-based strategy: Data is placed close in terms of network hops (if not on the entity) to where functions that expect input data with the same label(s) have been already instantiated. The data is collected in common, pool-like storage, so that several functions can make use of it. For example, let us consider functions that expect input data with labels "video-data", "IoT-measurements", and "highway-traffic". When data with these labels is present in the vicinity of the functions at the edge, the strategy replicates data associated with one or more of these labels into storage in the functions' vicinity or at the EDRs that offer these functions.

Hybrid strategy: This strategy combines both function proximity and label-based data popularity to make placement decisions. It first places data close to functions that expect input data with the same label(s). If such functions cannot be found, the data is placed at EDRs based on its (and its requested labels') popularity.

4.4.5 EDR Operation

Having discussed the different components of SEND, in this subsection, the operation of EDRs (Algorithm 1) is presented. Once a user application/device offloads data or requests the execution of a function, this data or request will reach the EDR closest to the user - for example, let us assume that EDR_1 of Figure 4.1 is the EDR closest to the user. In the case of data, the data placement strategy instance running on EDR_1 will determine whether the data should be stored locally or should be replicated/relocated to another EDR. In the case of a function request, EDR_1 will determine whether to execute this request locally or route (distribute) it to another EDR for execution. The input data can be carried by the request itself or have already been stored at one or more EDRs. For example, if the majority of the input data is available at EDR_N instead of EDR_1 in Figure 4.1 or the requested function is unavailable at EDR_1, but available at EDR_N, EDR_1 will re-route the request towards EDR_N for execution. Once a function is executed, the output will be returned to the user that requested its execution. The output will also be stored by the EDR that executed the function or be relocated/replicated to other EDRs according to the data placement strategy.

Finally, statistics about the stored and relocated/replicated data (e.g., based on the label(s) of the data) or the function requests may be maintained by EDRs. Periodically, each EDR will send its latest statistics to the management entity of the edge network. This entity will provide feedback to EDRs on how to handle future incoming data and function requests.

ALGORITHM 1
EDR Operation

```
1:  Inputs: Input $i$ received from a user device
2:  if (type($i$) == data) then
3:      data_destination_EDR = data_placement_strategy(labels($i$));
4:      if (data_destination_EDR == local_EDR) then
5:          store($i$);
6:      else
7:          replicate($i$, data_destination_EDR);
8:      end if
9:      label_statistics $ls$ = update_label_population(labels($i$));
10: else if (type($i$) == function_request) then
11:     function_execution_EDR = routing_process($i$);
12:     if (function_execution_EDR == local_EDR) then
13:         input = get_input_data($i$);
14:         output $o$ = execute(function_to_execute($i$), input);
15:         output_EDR = data_placement_strategy($o$);
16:         replicate($o$, output_EDR);
17:     else
18:         send_to_closest_process($i$, function_execution_EDR);
19:     end if
20:     function_statistics $fs$ = update_function_statistics($i$);
21: end if
22: if (time_to_send_update()) then
23:     send_update_to_management_entity($ls$, $fs$)
24: end if
```

The functions of Algorithm 1 are defined in Table 4.1

Table 4.1: Algorithm function definitions

Notation	Definition
i	Data item/function request
update_label_population	Update the counts for the labels of the replicated data piece
routing_process	Processing the route for the function needed by the respective processing request
data_placement_strategy()	Apply chosen data placement strategy for the appropriate labels (can also be a collection, associated with content
send_to_closest_process()	Send data in brackets to wards the closest EDR that can support the execution of the needed function, towards a final function execution EDR that is known to have the function regardless of transitory state of on-path EDRs
time_to_send_update()	Time-based epoch (synchronous epochs) passed check for ERP update

4.5 Evaluation

In this section the evaluation of SEND uses two different setups. First, an EDR prototype is implemented, which is evaluated based on two real-world datasets. To scale up the experiments, a network simulation study is conducted to evaluate the SEND framework as a whole, within realistic environments.

4.5.1 EDR Prototype Experiments

The EDR prototype was implemented on top of the Google file system [44] (GFS). The node of Google's file system was approached for the EDR prototype implementation as GFS was the most advanced distributed, industry-accepted collaborative and high-availability interactive storage platform that was available at that point in time, and it also was, conceptually-speaking, closest. The prototype is evaluated under two scenarios: (i) storage of images containing complex scenes; and (ii) storage of measurements captured by IoT devices.

Datasets, implementation, and experimental setup: For the first scenario, 100,000 images from the COCO dataset [106] were used. For the second scenario, a dataset captured and publicized in the context of the IoT Inspector project [105] was used. The dataset comprised of data points from 3,000 users and 30,000 IoT devices. For both scenarios, the following metrics were evaluated: (i) the time to insert a data (non-processing) message (image or IoT measurement) to the EDR as the number of data messages inserted was increased (increasing load on the EDR); and (ii) the time to search for stored data based on labels and other attributes such as generation timestamps. These experiments were performed on a server equipped with an Intel i5-9600K processor, 32GB of RAM, and an SSD SATA III drive. Each experiment was run ten times and average results are reported.

Data insertion time: We refer to ”insertion” here as the data writing process to a specific EDR’s storage, with the appropriate (embedded) mapping (freshness period, shelf-life, label and hash). In Figure 4.6, the insertion time results are presented for every 20,000 data image and IoT measurements insertions. For the image use-case, the total number of labels is varied. The total insertion time is composed of the GFS mapping time, added to the system-imposed data access and write time. Results demonstrate that as the amount of stored data increases, the data insertion times increase as well. This is due to the fact that data access time is needed to find the proper place to insert each incoming data piece within an increasingly larger pool of stored data. However, the overall data insertion time stays below 0.9ms.

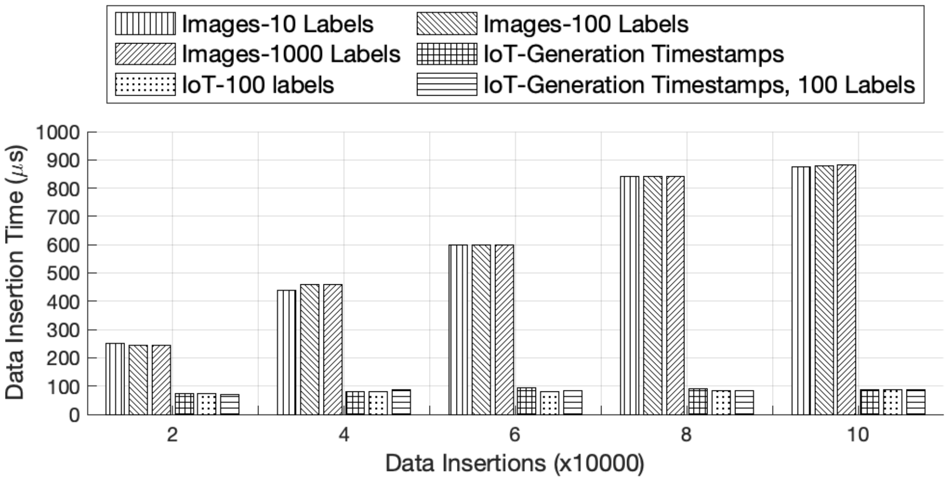


Figure 4.6: Data insertion times. Results reported for every 20,000 insertions for both use-cases (images and IoT measurements).

For the use-case of IoT measurements, varying numbers of labels as well as data generation timestamps are considered. Results indicate that the different attributes do not significantly affect the insertion time. As the total amount of inserted data is increased, the insertion time stays between 60μs and 100μs due to the fact that IoT measurements are of considerably smaller sizes (about 40 bytes each) compared to images (up to 2MB per image). As a result, not only the individual data pieces inserted are of small sizes, but the total size of inserted data is also smaller than the image use-case.

Data lookup time: In Figure 4.7, the data lookup time results for every 20,000 insertions for both images and IoT measurements are shown. Results demonstrate that the lookup time increases as the total amount of stored data is increased. The lookup time also increases with the complexity of lookup operations - for example, by increasing the total number of labels or performing lookups based on a combination of data attributes (e.g., labels and generation timestamps). The lookup times for images range from 0.5ms to 5.3ms, while the lookup times for IoT measurements range from 0.8ms to 4ms. The IoT measurement lookup times are comparable to the lookup times for images due to the characteristics of the IoT dataset, which results in larger numbers of stored data messages (IoT measurements) matching lookup queries.

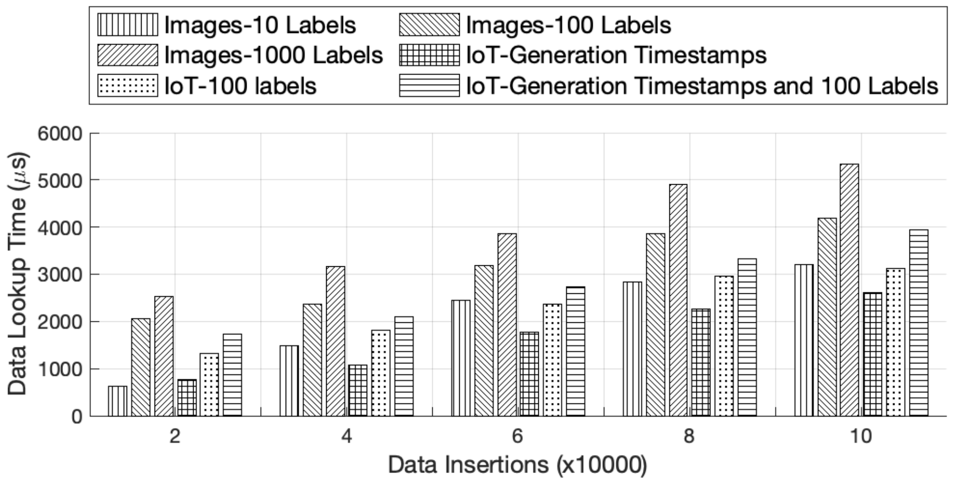


Figure 4.7: Data lookup times. Results reported for every 20,000 insertions for both use-cases (images and IoT measurements).

4.5.2 SEND Simulation Experiments

Simulation setup, parameters, and evaluation metrics: For the SEND simulation environment, the Icarus network simulator [87], a python-based, discrete-event simulator was used. We extended Icarus with an EDR implementation and the concept of data labels. The data placement strategies mentioned in Section 4.4.4 were also implemented for the management of data in storage.

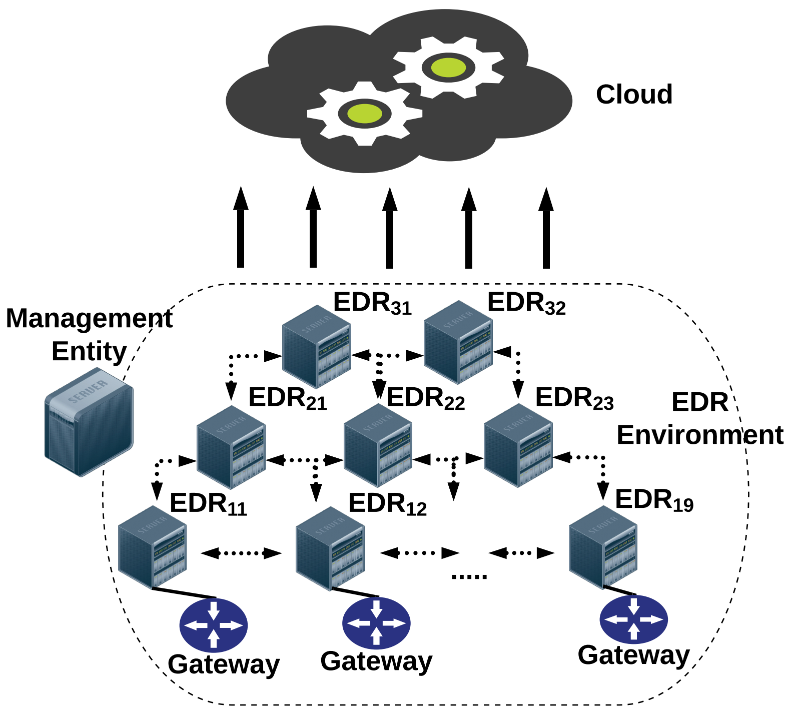


Figure 4.8: Simulation topology.

In Figure 4.8, the interconnected topology used is shown, which consists of EDRs, gateways that receive user requests and data (attached to EDRs at the lowest tree level), a management entity, and a cloud provider. However note that the presented topology can be dynamically configured through SDN and (through SDN) can be very different from the overlay, which is implemented via the management entity. The simulation parameters and datasets are presented in Table 4.2. At the beginning of each simulation, there is a data generation and placement round where EDRs are populated with data. After this round, users send requests for data (raw or after being processed by a function). Each user request is associated with a latency deadline by which the user needs to retrieve the requested data. As a baseline for comparison, a strategy that optimizes function placement based on the number and deadlines of requests is implemented, but without the use of labels (or storage) [36], which we shall call "hybrid-no labels" strategy. To perform a fair comparison with this baseline strategy, the same method to generate request deadlines that Ascigil *et al.* used [36] is applied. Specifically, a request deadline is a random time value (in ms) between the following lower- and upper-bounds: Round Trip Time (RTT) from the gateway that receives the request to its adjacent EDR at the lowest tree level (lower-bound) and RTT to reach the cloud from the gateway that receives the request (upper-bound). Each experiment was run ten times and the average results were reported. Note that we experimented with various different link delays across the entire topology to verify that the presented results still hold.

At the edge, data is replicated among EDRs according to the selected placement strategy, while once the data shelf life expires, the data is archived on the cloud. In the experiments, we use the following datasets: (i) a synthetic dataset; and (ii) a Microsoft Azure dataset [107] and a Google cloud dataset [108], which we utilize to extract the populations of raw data, functions, and labels, the assignment of labels to data, as well as the rates and the actual requests for both raw data and functions. We focus on the following metrics:

• **Request satisfaction rate**: The percent of user requests that are satisfied by their associated deadlines.
• **Normalized overhead**: The volume (in bytes) of overhead traffic per (raw or processed) data piece normalized by the average size (in bytes) of each data piece. The overhead traffic includes the traffic generated for the replication of data among EDRs and the archival of data from the edge to the cloud.

Each EDR exchanges feedback updates with the management entity for every 1,000 received requests. We selected this frequency, so that each EDR is able to gather a reasonable population of data label and function usage statistics, avoid overloading the edge with overly frequent feedback updates, and, at the same time, maintain up-to-date feedback from the management entity. We experimented with lower and higher frequencies. More frequent updates do not substantially improve the request satisfaction rates, but increase the overhead. Less frequent updates can degrade the request satisfaction rates with no substantial improvement of the overhead.

Table 4.2: Simulation parameters and used datasets.

Parameter	**Value(s)**
Number of EDRs	14
Distance and link delay between EDRs at the highest edge level and cloud	3-5 hops and 40ms following values reported by recent studies [109, 110] and cloud computing trends [111]
Used Datasets	Synthetic dataset: 10,000 unique raw data pieces, 10,000 unique functions, 1,000 unique labels (pseudo-randomly generated), 150 user requests per second, 600,000 requests for raw data, 400,000 requests for functions
	Microsoft Azure dataset: 50,000 unique raw data pieces, 50,000 unique functions, 5,000 unique labels, 100 user requests per second, 320,000 requests for raw data, 680,000 requests for functions
	Google cloud dataset: 100,000 unique raw data pieces, 100,000 unique functions, 100 unique labels, 100 user requests per second, 480,000 requests for raw data, 520,000 requests for functions
EDR-Management feedback frequency	Per 1,000 received user requests
Data sizes	Ranging from 40 bytes that represent IoT measurements to 5MB that represent high-quality images
Link delay between EDRs	10ms
Link delay between a gateway and its adjacent EDR	2ms

Simulation results: In Figure 4.9, we present the request satisfaction rate results for different datasets. For our synthetic dataset (Figure 4.9a), the general-purpose placement strategy performs the best, reaching 92% of satisfaction rates. This is due to the following reasons: (i) this strategy does not concentrate on specific labels (e.g., the most popular data labels), treating equally data of different labels that is stored in the system; and (ii) none of the labels in this dataset is far more popular than others. This is also evident by the fact that specialized strategies (e.g., focusing on data popularity, proximity of data from processing functions) result, in general, in lower overall satisfaction rates than the general-purpose strategy, being able to satisfy on-time 61-72% of the requests.

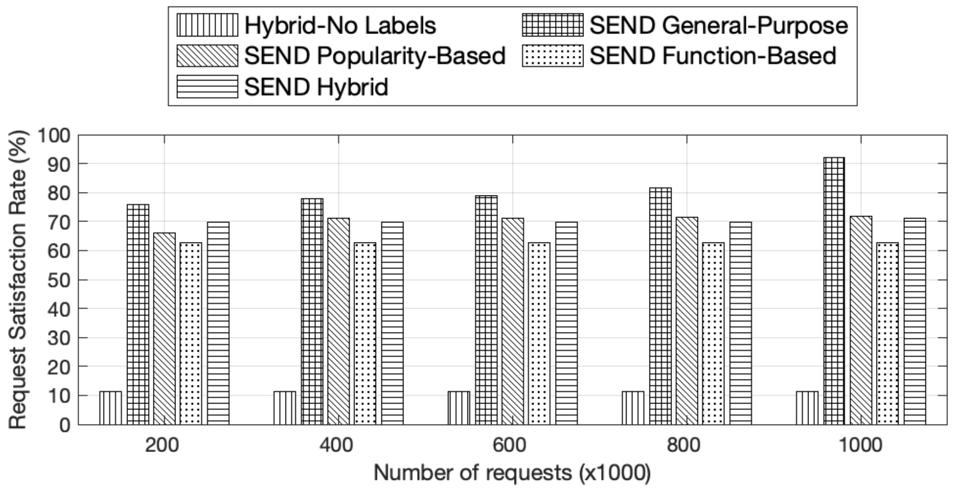
For the Azure dataset (Figure 4.9b), the function-based placement strategy achieves the highest request satisfaction rates (up to 90%). The SEND hybrid and the general-purpose strategies perform slightly worse than the function-based strategy reaching about 85% of satisfaction rates as the number of requests increases. This is due to the fact that the majority of the requests in this dataset are for processed data. Therefore, placing pieces of data close to EDRs, which offer functions that may require such pieces as inputs, results in executing the requested functions and returning the results (processed data) to users in a timely manner.

For the Google cloud dataset (Figure 4.9c), the SEND general-purpose and hybrid strategies achieve the highest satisfaction rates (up to 90%). This is attributed to the fact that this dataset contains a relatively balanced mix of requests for both raw data and functions (processed data).

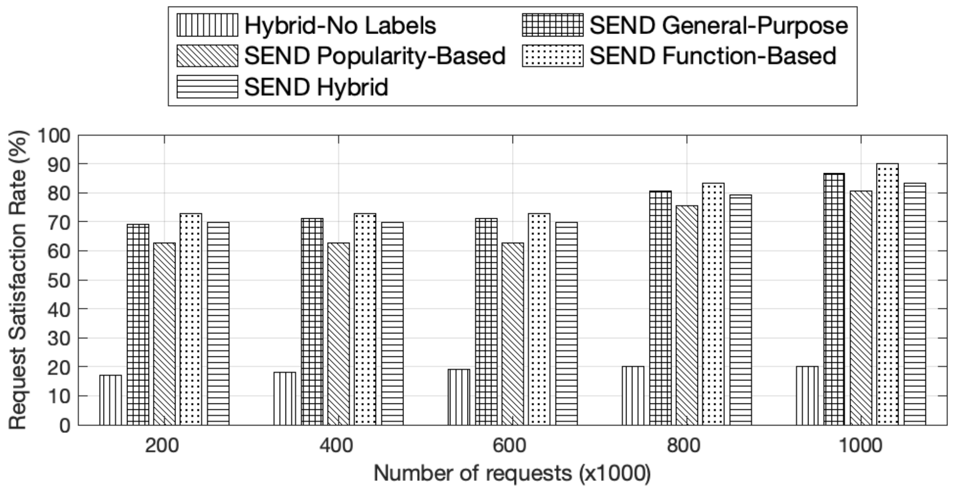
Note that for all datasets, the strategy that does not make use of labels for data placement results in satisfaction rates of roughly 10-55%. These results demonstrate the importance of labels, which make SEND aware of the data context, for the purpose of efficient placement of both data and functions in the EDR environment.

In Figure 4.10, we present the overhead results for different datasets and numbers of user requests. For our synthetic dataset (Figure 4.10a), the general-purpose strategy incurs the highest overhead, since each data label is roughly evenly distributed in the synthetic dataset. As a result, all raw and processed data is replicated at the edge to the same extent, which is also the reason that the general-purpose strategy achieves the highest satisfaction rates (as illustrated in Figure 4.9a). For the Azure dataset (Figure 4.10b), the function-based placement strategy incurs the highest overhead, since the majority of the requests in this dataset are requests for functions.

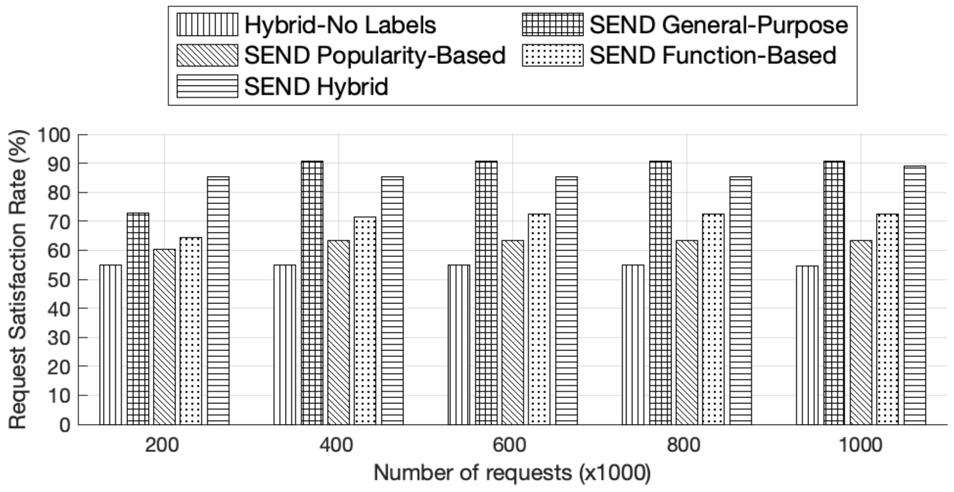
Finally, for the Google cloud dataset (Figure 4.10c), the SEND general-purpose and hybrid strategies result in the highest overheads. This dataset contains fewer labels compared to the previous datasets and a relatively balanced mix of requests for raw and processed data. To this end, the general-purpose strategy repli-



(a) Satisfaction rate for the synthetic dataset.



(b) Satisfaction rate for the Microsoft Azure dataset.



(c) Satisfaction rate for the Google cloud dataset.

Figure 4.9: Request satisfaction rates for different datasets.

cates the majority of data in this dataset, since data with the same label is typically already present in the EDR environment, while the SEND hybrid strategy prioritizes the replication of data based on both their labels and the proximity to relevant function locations at the edge.

4.6 Further Discussions, Considerations and Following Research

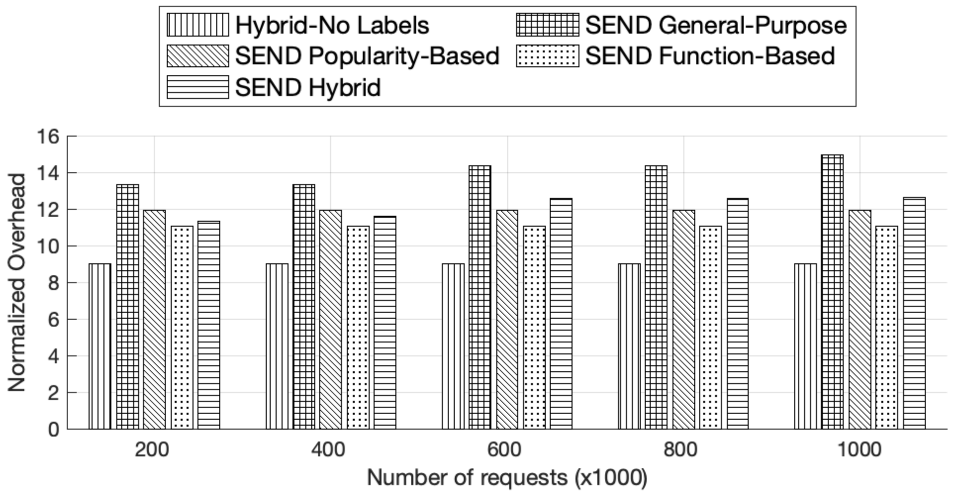
Considering **Locality- and context-based optimisation/improvements**, which is implied and exploited in the above-studied system, there are a few more, dynamic management and routing decisions that could be made, based on different data, measurements and metrics. Since management strategies have been defined and used up to this point, their outputs, or rather their statistics and more importantly, their actions could also be used further. These kinds of metrics and actions could help in the decision process of managing data and, more importantly, applications within the EDR environments, to reduce the amount of space that data uses and to more efficiently manage applications and services, according to their labels, hashes, associated content and results.

Function Data MetaInfo:

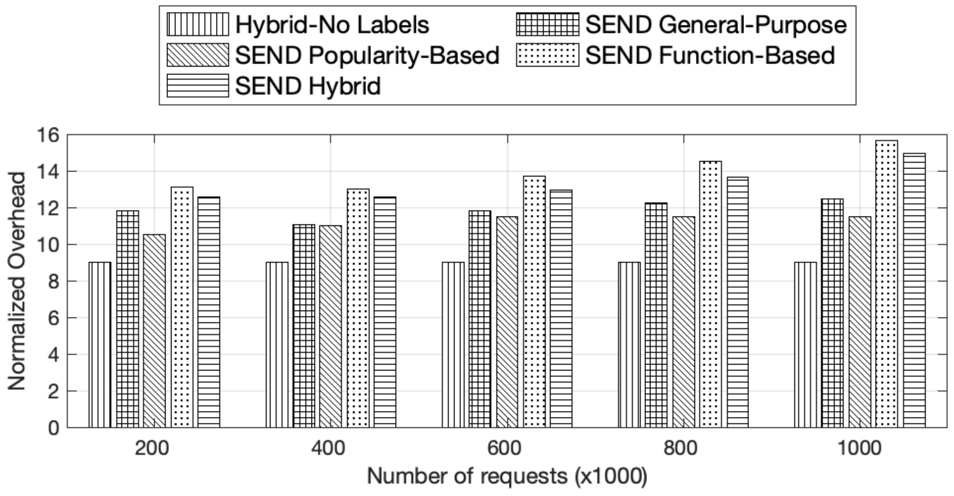
Functions are associated to and come along with the same associated labels as the data corresponding to the needs shown by their MetaInfo field. However, functions are also hashed and have extra restraining/enabling fields within their MetaInfo. Thus, the combination of function MetaInfo and associated data/hash could also be used in management entities to manage function (and data pair) instantiation, provision and distribution.

Research into System Metrics Optimisation

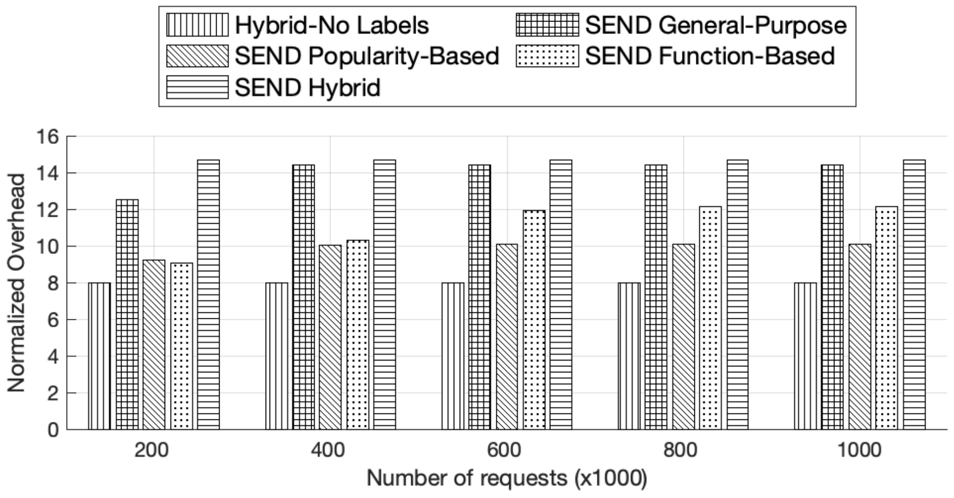
A collaboration towards another EDR-related paper, but considering routing, information efficiency and basing more of the work into some optimisations of the system, using Locality-Sensitive Hashing, was further pursued, after the submission of the above-mentioned work for INFOCOM 2021. The main direction considered was the analysis of Edge data provision, validity and, most importantly, a network



(a) Normalized overhead for the synthetic dataset.



(b) Normalized overhead for the Microsoft Azure dataset.



(c) Normalized overhead for the Google cloud dataset.

Figure 4.10: Normalized overhead for different datasets.

resource usage, information efficiency and QoE improvement mechanism.

This work will be presented in the following chapter and provides a good introduction to more systems-based approaches, stemming from the above work. To be more specific, the following work approaches resource utilisation optimisation, data and functioin management and distribution, and last, but not least, native data privacy preservation, towards a more rounded and effective Edge-based computation system.

Chapter 5

ReStorEdge: An edge storage system with computation reuse semantics

5.1 Introduction

The overall problem statement addressed in this chapter can be formulated as: Given a set of distributed edge nodes that are able to process user queries for a given application, how should queries be directed to edge nodes so that: 1) the opportunity for reuse of prior cached results for similar queries is maximised; 2) load is balanced across edge nodes, given the processing implications of the resulting mix of reused and fully processed queries, to minimise query response time, while maximising query throughput.

This chapter proposes an edge framework, called ReStorEdge, which offers session-less data processing, storage and computation reuse for a range of services, such as media tagging, subtitling, and natural language processing. This chapter makes the following contributions:

- The proposal of an architecture for processing queries on a set of Edge Data Repositories (EDRs). Each EDR features storage and computing resources, which are made available to serve the needs of applications at the edge for low-latency data storage and processing;
- The proposal and evaluation of a set of orchestration strategies to manage the distribution of queries between EDRs to minimise the computational load on EDRs,

maximise the reuse of previous results and maximise Quality of Experience (QoE) for user queries by minimising query latency;

• The evaluation of the proposed architecture by, firstly, implementing a prototype of ReStorEdge to quantify the benefits of computation reuse in a range of applications; and, secondly, using the results of the prototype evaluation in a wider simulation of distributed EDRs to evaluate system performance under different orchestration strategies.

The rest of the chapter is organised as follows. In Section 5.2, we present our ReStorEdge system design. In Section 5.3, we present our experimental evaluation, and finally, in Section 5.7, we discuss our future work and conclude our work.

5.2 System Design

Our design is based around using LSH [112] for improving the performance of storage-enabled networked environments [65]. In Figure 5.1, we present an overview of the ReStorEdge overall operational workflow. ReStorEdge includes a data storage and processing location management component - the orchestrator - which uses EDR-specific and system-wide metrics such as CPU-usage and data reusability to optimise the distribution of processing and storage amongst EDRs.

The other key component is the EDR environment which routes, processes, stores and redistributes queries and data through the system, as directed by the orchestrator.

5.2.1 Definitions of terms

Request/Query - a message containing data to be processed, sent by users to be processed by the network of EDRs.

LSH query/hashing process - this process maps an incoming query to a bucket, which contains all prior queries with the same hash value. Thanks to the LSH algorithm all queries in the same bucket will be similar.

Hash - the result of applying LSH to a request/query.

Bucket - the set of queries with the same LSH hash value.

Reuse - a request/query causes a reuse HIT when the result of LSH (array of

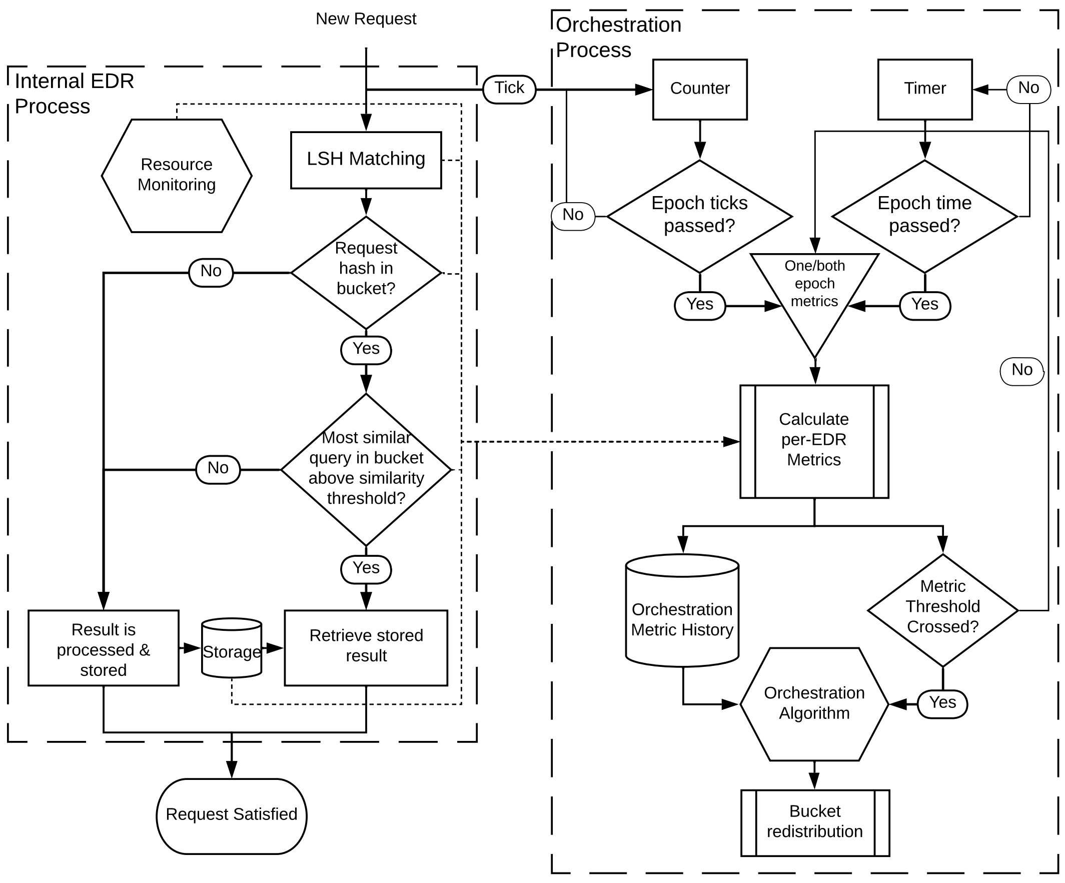


Figure 5.1: Overview of ReStorEdge Workflow

nearest-neighbour hashes), combined with the (two-process-) application-specific-similarity-check, return directly the results of a "similar-enough" query. A query causes a MISS when the above-mentioned process fails any of the checks. The MISS causes the query-associated function to go through the classic "query" or "processing" function.

"Similar enough" - this feature is application-dependent (or it can be EDR-provider-dependent, depending on priorities), and it is determined depending on metrics such as "needed" accuracy, system performance (EDR-environment-related) and/or latency constraints;

Application-specific-similarity-check - this "filters through" a nearest neighbour hash array (resulting from LSH) through TWO processes:

1. application-specific check - which first checks if a specific application/"label"

is associated to the query/request hash;

2. Similarity check - which then checks how much the query data in the new query is structurally-similar to the old query data, and, if “similar enough”, provides the link to the old query/request’s results, to be returned as the answer;

NOTE: In the simulations, the similarity percentage values are parameters of the system and the specific values used were not determined experimentally based on any objective ground truth comparison, user satisfaction scores or similar. They were determined based on the first evaluation experiments, and a reasonable understanding of what would be expected, in regards to accuracy, from certain related applications.

Query/Request processing - once a MISS is registered for certain queries/requests that are within the scope of (buckets processed by) the respective EDR, the associated service/processing has to be carried out (normally within a specified deadline of the query/request).

5.2.2 Description of ReStorEdge Operational Workflow

As it can be seen in Fig. 5.1, after a new request enters the system, at the edge nodes (the first EDR - which should be one hop away from the generation point), it is processed in two different ways, by two different processes, while also being routed (and possibly stored, for different purposes, outside of the context of this study).

- The main processing path is the internal EDR environment process. This includes most of the data routing, so for simplicity, we will not present routing as a separate process. We shall detail this process below:

    1. When the data enters the system, the application tag/label of the data is checked, data is compressed and LSH is applied and the hash is generated;

    2. The data (with its associated hash) is put into a bucket, and if the bucket exists in the system already, the query becomes a reuse candidate

   (application-permitting, of course - which is why application association is the first check);

3. If the bucket is not unique (generated specifically for this query) and the current request data is similar enough with the latest query data in the bucket, (stored) the results of the previous query data are looked up, fetched and returned to the appropriate output gateway.

4. If the request data is dissimilar/not similar enough to previous queries, it is processed and the results are sent to both the appropriate output gateway, as well as to the local (processing EDR's) storage. The query data is stored along its results, for the same hash entry.

- The other, more analytical processing path, is represented by the Orchestration process. This process only takes the most essential metadata from each request and adds it in the context of the system-wide image it creates over a certain time (epoch), in order to provide incremental system optimisation. The process in this path relies on the querries' MetaData (like hash, application identification/label and storage-related MetaData), and the previous queries/previous system state, as detailed below:

  1. First, as there is an environment counter, counting ingress requests, it is incremented by 1, and the epoch ticks are counted. There also is an epoch timer. These are used for different types of orchestration strategies (or in conjunction);

  2. If the epoch has passed on this check (normally within at most a few ms), the following is done in parallel:

     – A check is made, whether the metric threshold of the respective strategy is crossed;

     – The orchestration metric history is updated;

  3. Orchestration algorithm is called;

  4. Buckets are redistributed across the EDR Environment.

5.2.3 Query Routing

As noted in the operational workflow description, a new query/request is routed through the system as follows in these steps: (i) a device sends query data into the system, via its closest (local) EDR, as the first hop; (ii) the local EDR does LSH on the query, then looks up the hash in its forwarding table to find the EDR responsible for the associated bucket; (iii) the EDR responsible for the bucket takes the query and returns the result to the appropriate user (if requested/subscribed).

This system was envisioned as both an on-demand and/or publish-subscribe service provider. Thus, services could either be continuously executed (managed, orchestrated and updated), inside the relevant EDRs and environment, or they could be executed/reused on-demand. In the latter case, services could also be viewed in a hybrid manner (executions in the background and relevant results obtained on-demand); also in the latter case, it is important to note that there may be high disparaty between user requests, which may render the reuse-based system much less imoprtant/interesting to study, which is why we do not approach such (rather outdated) use cases, which, realistically, could also be managed via cloud-based services.

NOTE: The forwarding tables in the appropriate EDRs (where needed), within the local environment, need to be updated whenever the orchestrator is triggered and reassigns buckets between EDRs.

5.2.4 Storage Similarity and Reuse

In videos, similarity comes naturally, thus encouraging the reduction of both storage and processing resources, via reuse. Data of this format can easily be reused at the edge, for low-latency response applications. Otherwise, most of it can be (re-) analysed at a later point, in the cloud, for archival purposes, or less time-sensitive, possibly more (encrypted and anonymised) complex/analytical processing.

The above use case provides the opportunity to identify whether there is a sufficiently similar (similarity threshold can be application-dependent) frame provided in an earlier query so that the results of the earlier image processing service can be returned rather than requiring the frame to be fully processed.

In the case of voice and command recognition, this becomes even more relevant, especially when the same person uses the service over short periods of time. There can be high similarity within the same voice profiles, comprised of encoded/hashed speech snippet datasets, provided by the smart-assistant(s), associated with the NLP service, based on previous speech-to-text classification and feature extraction. This kind of data is very well suited, to be encoded for reuse at the edge, in order to take advantage of locality and the privacy-preserving environment. Otherwise, the readily encoded and anonymised nature of data packaging provides a very good way for it to be further stored and/or disseminated, for other kinds of analytics and processing.

As it can be extrapolated from the above, the algorithm uses such similarity to identify whether there is a sufficiently similar complete command-service association, provided in an earlier query so that the service of the earlier resulting service call processing can be returned rather than requiring the speech to be fully processed (note that this would normally require two processing levels).

Once more data enters an EDR, the history of (application-specific) related data and information can be used in the advantage of the application **and** system. Similarity, based on LSH and the new data's related (previously-stored, application-specific) information and meta-information, helps determine the reusability of an existing query, function and associated resulting data. Data reusability is determined by the existence (and possible renewal) of (still) relevant results, based on previous data and function calls, and new data, similar to the previously-requested/previously-resulted data.[1]

5.2.5 Orchestration Algorithm

The placement of certain buckets in specific EDRs/areas of the EDR environment can be advantageous for many reasons, among which load balancing and resource utilisation optimisation in some cases, due to the predominance of data/functions of a more relevant type, topic, application, provider and/or data similarity in these EDRs or this area. Thus, for the system's service efficiency to increase, an orches-

[1]EDR-based ReStorEdge prototype github link placeholder

tration algorithm is needed in certain cases, in order to optimise bucket placement. Such an algorithm could potentially be easily adapted and placed in an SDN controller, as one of the main implementations of a system like ours.

We now introduce three orchestration algorithms that trade-off system resource usage and QoS. However, another trade-off comes up in the best case scenario, and that is the trade-off between **performance increase in both QoS and resource usage** and **the cost for more strictly and intelligently managing the system over time**. The demonstration of these strategies will be evaluated with different datasets (representing different use cases) in section (5.3).

Strategies' descriptions[2]:

1. **Queue Delay**: trigger orchestrator - when queued processes on any EDR are delayed for more than $2 * mean\_processing\_time$ s. Rank all buckets in terms of their numbers in the queues and move the top ranking buckets to the EDRs that house the buckets that have the lowest (or no) queue utilisation. Rankings:

- for highest - from highest to lowest EDR, with a catch: we track highest delay AND (if needed) highest number of queued requests. If there is a highest delay, the EDR will be added as the highest-ranking EDR (for the loop through the EDRs), otherwise the longest queue is counted. Then, once all EDRs have been added, for each EDR, the same is done as for the CPU ranking;
- for lowest, on the other hand, another process is implemented - First, it is checked if the EDR does not serve any bucket. If that is the case, the EDR is automatically added to the list. Then, if the node has the least amount of queued requests, it is also checked whether the node has the least amount of buckets associated with it. However, the main decision factor here remains the minimum requests in the queue.

2. **CPU-usage**: execute algorithm when CPU load on any EDR exceeds trigger threshold. Rank N_{EDRs} buckets in terms of their contribution to CPU load and move the top buckets to the EDRs that house the buckets that have the lowest (or no) CPU utilisation. Rankings:

[2]Simulation implementation github link placeholder

- Take N highest CPU EDRs;
- In each of these EDRs, take highest-CPU-buckets;
- Take N lowest CPU EDRs;
- Allocate highest-CPU-buckets one-by-one to minimum-workload EDR;
- On each iteration, update equivalent expected CPU utilisation (repeat above process for each highest-lowest pair).

3. **CPU-Workload**: execute algorithm every epoch, of x requests (*epoch_ticks*). Rank $N_{Buckets}$ buckets in terms of their contribution to request workload and move top bucket to the EDRs that house the buckets that have the lowest (or no) workload allocation. Update the ”expected” workload for all EDRs, so that they do not rank highest/lowest for workload allocation on the next-top-bucket-workload allocation iteration and repeat for all buckets. Because of the latter process, this is the most resource-intensive orchestration strategy. However, it is the most thorough, and, as the orchestrator should have an overview of buckets and EDR status, it should also be capable of such orchestration. The main downfall of this strategy is its processing time, where there are many buckets to be taken into consideration. Note that the orchestration time is not taken into consideration in the evaluation of this work, however it would be something to be taken into consideration at a later time (i.e. the trade-off between considering more or less buckets and the impact of such a decision on orchestration timing and/vs. performance improvement) Rankings:

- Take N highest request workload buckets;
- Take lowest request workload EDRs;
- Allocate one-by-one to minimum-workload EDR;
- On each iteration, update equivalent expected workload (repeat above process for each highest-lowest pair).

4. **Reuse-CPU**: Both epoch and trigger-based orchestrator - When the trigger is activated, two updates happen:

The highest CPU-usage buckets are moved to the highest reuse buckets’ locations, and vice versa. This is done mostly because the highest reuse would normally use the least CPU power and would be the best for housing more processing-intensive applications, and the lowest reuse buckets would use the most CPU to satisfy their requests, on a nearly-constant basis, while the least CPU-usage buckets would use the least amount of resources, thus being good candidates for the extra load that generally-/constantly-high CPU-load buckets would impose.

- CPU measurements are obtained in the same way as in the CPU-based strategy.
- The reuse measurements are the same as for the queue delay measurements, with the exceptions that the measurement is for total/per-bucket reuse, and then, the minimum ranking is done without taking either the number of buckets or any empty EDRs into consideration for ranking buckets.

ALGORITHM 2
Triggered, Queue-delay-based Orchestration Strategy

```
1:  **Inputs:** Input $i$ received from a user device
2:  **if** Max_queue_delay($current\_node$) $> 2 * mean\_req\_proc\_time$ **then**
3:      $h\_d$ = highest_queue_delay($no\_of\_buckets$)
4:      $l\_d$ = lowest_queue_delay($no\_of\_buckets$)
5:  **end if**
6:  **for** ($n\_h$, $h$, $n\_l$, $l$ in zip($h\_d$[0], $h\_d$[1], $l\_d$[0], $l\_d$[1])) **do**:
7:      move_bucket($n\_h$, $n\_l$, $h$, $l$)
8:      move_bucket_data($n\_h$, $n\_l$, $h$, $l$)
9:  **end for**
10: function_statistics $fs$ = update_function_statistics($i$)
```

ALGORITHM 3
Triggered, CPU-usage-based Orchestration Strategy

```
1:  **Inputs:** Input $i$ received from a user device
2:  **if** CPU_usage($current\_node$) $> trigger\_threshold$ **then**
3:      **for** ($N_{EDRs}$) **do**:
4:          $n\_h$, $h$ = most_proc_usage()
5:          $n\_l$, $l$ = least_proc_usage()
6:          move_bucket($n\_h$, $n\_l$, $h$, $l$)
7:          move_bucket_data($n\_h$, $n\_l$, $h$, $l$)
8:          update_orchestration_CPU($n\_h$, $n\_l$, $h$, $l$)
9:      **end for**
10: **end if**
11: function_statistics $fs$ = update_function_statistics($i$)
```

Before execution, after a number of requests (epoch), a strategy trigger is checked and it does not reset the request count until the strategy triggers. This

ALGORITHM 4
Epoch-based, CPU-Workload Orchestration Strategy

```
1:  Inputs: Input $i$ received from a user device
2:  if $Request\_ticks > Epoch\_ticks$ then
3:      for ($N_{Buckets}$) do:
4:          $n\_h$, $h$ = most_workload_perBucket_at_update()
5:          $n\_l$ = least_orchestration_workload_perEDR()
6:          move_bucket($n\_h$, $n\_l$, $h$)
7:          move_bucket_data($n\_h$, $n\_l$, $h$)
8:          update_orchestration_Workload($n\_h$, $n\_l$, $h$)
9:      end for
10: end if
11: function_statistics $fs$ = update_function_statistics($i$)
```

ALGORITHM 5
Epoch-based, CPU-triggered, CPU and Reuse Bucket Change Orchestration Strategy

```
1:  Inputs: Request $i$ received from a user device
2:  if CPU_usage($current\_node$) > $trigger\_threshold$ then
3:      for ($N_{EDRs}$) do:
4:          $n\_h$, $hp$ = most_proc_usage()
5:          $n\_hr$, $hr$ = most_reuse()
6:          move_bucket($n\_hp$, $n\_hr$, $hp$, $hr$)
7:          move_bucket_data($n\_h$, $n\_l$, $h$, $l$)
8:          update_orchestration_CPU($n\_hp$, $n\_hr$, $hp$, $hr$)
9:      end for
10:     for ($N_{EDRs}$) do:
11:         $n\_lr$, $lr$ = lowest_reuse()
12:         $n\_lp$, $lp$ = least_proc_usage()
13:         move_bucket($n\_lr$, $n\_lp$, $lr$, $lp$)
14:         move_bucket_data($n\_h$, $n\_l$, $h$, $l$)
15:         update_orchestration_CPU($n\_lr$, $n\_lp$, $lr$, $lp$)
16:     end for
17: end if
18: distribute($i$, bucket_update)
19: distribute($i$, bucket_data_update)
20: function_statistics $fs$ = update_function_statistics($i$)
```

Table 5.1: Algorithms 2-5 function definitions

Notation	**Definition**
Max_queue_delay()	The maximum delay imposed by processing queues inside the node targetted by the function
highest/lowest_queue_delay(x)	The top x buckets to have the highest/lowest queue delays in all EDRs - h_d and l_d are the resulting highest and lowest, respectively, queue delay hashes/buckets and corresponding EDRs
most/least_proc_usage()	The most/least CPU utilisation associated with hash/bucket and its respective EDR in system
update_orchestration_X	Update in-algorithm (temporary/predicted) X as a result of the previous change, to account for in next loop operations
most_workload_perBucket_at_update()	The highest workload hash/bucket and associated EDR at the start of the orchestration update calculation
least_orchestration_workload_perEDR()	EDR with lowest workload associated at the point of the calculation (can be influenced by the above - most_workload_perBucket_at_update() - calculation, and each orchestration update - update_orchestration_Workload(n_h, n_l, h)
max_count	Maximum number of buckets to be considered for exchange between EDRs with highest and lowest CPU usage measurements
n_hr : n	repository, h - top, r - reuse
n_lp : n	repository, l - bottom, p - processing
hr : no n	hash/bucket, h - highest, r - reuse
hr : no n	hash/bucket, h - highest, r - reuse
move_bucket	changing routing tables, to (re)associate buckets to specific EDRs - equivalent to distribute()
move_bucket_data	move the data associated with specific buckets to the newly-associated EDRs

is done in all strategies, except CPU-Workload, where the strategy triggers on every epoch.

5.3 Evaluation

5.3.1 Setup and Assumptions

The evaluation is done in two phases. We first approach a local, prototype implementation, of the LSH-similarity-storage/processing system, and then we look into scaling the system up, to account for the whole EDR environment. The Edge environment we consider in our evaluation is either a smart home or a traffic-based environment/smart city, and the assumptions of the underlying systems and functions/storage services in place (i.e. a few hybrid-based function and storage services and their providers' willingness to use storage and/or LSH). These environments are increasingly pervasive and important to consider, because of their numerous implications and use cases.

Assumptions made on reuse - In this work, while we base our findings on realistic datasets, for the sake of computational resources and time efficiency, but with little loss of accuracy (an overall assumedly higher reusability), we base our simulation results on one reuse rating per bucket, rather than implementing dynamically-varying reusability ratings in the simulations (this would require executing LSH and application similarity checks live, within the simulations, on the real datasets).

Assumptions made on processing resources - Another important assumption we make, for simplifying system complexity is that the CPU cores do not use any pipelining principles. Further, we also assume that requests do not have deadlines in this case, as we are strictly interested in the performance of the Edge environment, and not the interaction of the Edge with the Cloud. However, these simplifications should not affect the overall performance evaluation in any positive manner.

Assumptions made on realistic workload distribution and considerations for fair comparison - In the EDR environment simulator, while we take the realistic LSH evaluation outputs as inputs, we do a random distribution of associated requests and load across the system's ingress EDRs, thus making it harder for the system to process every request AND reach the back-of-the-envelope calculations' use case conditions.

Use cases

As the IoT and Internet Edge environments evolve, the Edge includes more and more audio/video (A/V) devices, which can be used for increasingly diverse tasks and/or services needed throughout the Internet. Thus, we present the use cases of traffic cameras (among the video/image use cases) and personal assistants (such as

Amazon Alexa, Google Assistant) as our main two applications, going through the algorithm implemented within our ReStorEdge design.

We approach two use cases within both parts of our evaluation:

1. detection/classification/counting, in video feed-based data and service management

2. object voice-command identification and service provision management.

In both of these, the request-to-response algorithm is the same, with slight differences of approach, due to data formatting and scope (already accounted for by labels/application-specificity).

The datasets we used (two, to represent different potential extremities of data present) to represent each of our two use cases are: the MNIST dataset[113], a traffic CCTV dataset, generated by one of the associated researchers, working on this paper, from University of Nebraska, at Omaha, placed next to a highway junction and capturing video (the dataset images), which is mainly meant for traffic detection around the junction, a dataset which contains anonymised ”Alexa” wake-word voice snippets [114] and another, general smart-home voice commands dataset, comprised of anonymised voice commands [115]

For this setup we used FALCONN [76] for all LSH queries on the datasets, the ”tiny” version of YOLO object detection ([116]) for the image processing software and pyAudioAnalysis [117] and pocketsphinx [118], as audio classification tools.

5.3.2 EDR-based ReStorEdge Prototype Setup and Metrics

This is a prototype of what is needed to be implemented at scale, for the system to provide storage, processing and reusability. This implementation provides the means for the algorithm explained in section 5.2. Further, this evaluation step provides realistic metrics and adds to the accuracy, credibility and realistic outlook of the second (simulation) part of the evaluation (5.3.5).

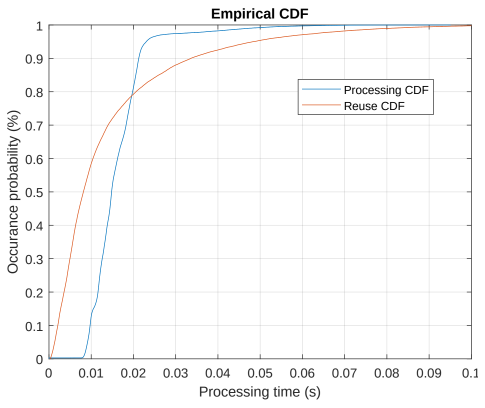
General Reusability of the datasets - This metric provides us with a preview of the amount we would expect data of certain types, topics and from certain sources to be reused, on average. We use this as a metric to determine the way in which the influence of certain orchestration approaches and resource allocation would impact system performance.

Overall request satisfaction times - reuse vs. processing - The trade-off implied here can be less obvious in a networked system, where we have to also take internal system delays and network transmission delays into account. Thus, we take previous measurements, relating to internal lookup and fetch delays, and see the difference between a request-execute-return and a request-match-lookup-fetch-return processes.

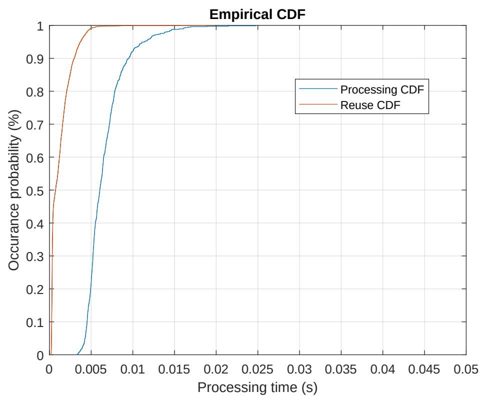
5.3.3 EDR-based ReStorEdge Prototype Evaluation

Processing time improvements - One of the biggest trade-offs we are clearing up in the case of these results are of effective information processing timings. It is **essential** that information is stored, used and processed as efficiently as possible. Thus, looking at the plots of processing/reuse CDFs of the image use cases (Figure 5.2), we can see a definite trade-off between function processing- and reuse-associated delays. A first (and the most obvious) observation is that, the larger the processing times, the more important reuse is for minimising latency. However, a more subtly-impactful and important trade-off is that, with the appropriate orchestration, the top of both curves can be used (subfigure 5.2a), to the advantage of both application providers and the EDR service provider. This will become more obvious in the next subsections' analysis of the ”CPU-usage Strategy” impact on the system, in simulations.

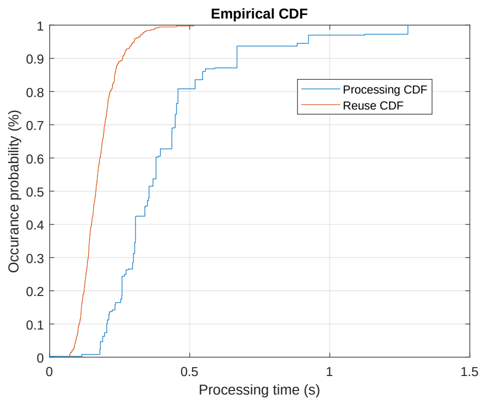
Processing times and LSH searches - While LSH and reuse help by fetching the needed data faster than processing the similar data, there are still other processes that slow the fetch down. However, for a good service and a more efficient system, these processes have to be maintained. In the case of images, this process is image resizing, for lower storage utilisation and faster LSH and result fetching. This was also proven previously to not affect accuracy[66]. For the case of sound files, the above-mentioned process is represented by feature extraction, which, while efficient, it hinders the fast Nearest Neighbours search that LSH entails. Making a parallel comparison, between the image and audio analysis cases, we can see a large



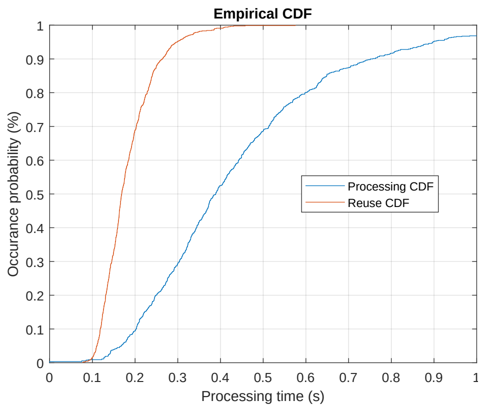
(a) MNIST



(b) Traffic Detection.



(c) Alexa



(d) General Commands

Figure 5.2: Processing vs. Reuse timings for different datasets.

increase in LSH timings (NN search timing) from the audio to the image datasets (LSH timing Plots/Table 5.3), however, we can see a large (but proportional) decrease in processing times in the same direction. Thus comparing the processing plots 5.2 and the LSH timing Table 5.2, we can conclude that, as pre-processing is done regardless of whether queries are just processed from scratch or they trigger reuse, the main importance in the matter of timings should be attributed to whether reuse is/should be done or not.

Table 5.2: LSH Nearest Neighbours Search Timings Table

Similarity	**Datasets LSH nearest Neighbour Search Avg (ms)**			
Threshold	MNIST	Traffic Detection	Alexa	General Commands
60%	5.3	0.6	0.0993	0.14
70%	6.8	0.6	0.0991	0.14
80%	6.8	0.8	0.0888	0.14
90%	6.9	0.9	0.1	0.15

Data Reusability - Since we are interested in the reuse of processed data (for different reasons/applications), we want to see **what applications** can rely more or less on reuse, **when** they can rely on processing reuse and **how much reuse needs to be taken into account**, to obtain the best results. Further, since certain data types and/or topics may be more or less correlated to each other, it may be that some similarity thresholds and application-specific associations need to be used, to ”filter down” most of, if not all, false-positive reused results. Looking at the MNIST part of the Reuse Table (Table 5.3), we can see a large difference in the reusability of data from the 60% similarity to the 70% thresholds. That is due to the big difference in the amount of change expected, having a high amount of space covered strictly by the number in the images of the MNIST dataset.

Table 5.3: LSH Reusability Table

Similarity	**Datasets Reusability Avg (%)**			
Threshold	MNIST	Traffic Detection	Alexa	General Commands
60%	12	70	86.6	18.6
70%	0.74	72.5	81.8	18.4
80%	0.42	67.2	59.3	18.3
90%	0.26	60.8	25.3	19.36

5.3.4 ReStorEdge Environment Simulator Setup and Metrics

As now we have realistic measurements of a localised example and we can expand the model of the system, in order to implement orchestration strategies, in order to make the case for a more realistic, decentralised and coordinated system, we will now scale our experiments up and look at an EDR-based simulation, for which the impact of the orchestration strategies will be critically analysed and assessed.

Orchestration Metrics and Evaluation Considerations

Considering the environment of a smart city, different applications/services can seamlessly and transparently interact through environments like that described in this paper. Thus, we consider 14 repositories (that would probably be distributed across a WAN network), we assume and impose that all EDRs are able to store and process data, but for simplicity, we do not simulate the storage placement and associated timing relevancy of data in this paper. Through this part of the evaluation we demonstrate the efficiency of the data placement algorithms, show superior QoS and resource allocation performance and consider all the imposed trade-offs. Finally, we show how the system could be fine-tuned, for the best outcomes (depending on the situation it is put in and the specific application/service providers' demands). To start with, we provide the simulation conditions and some considerations taken for this evaluation:

Table 5.4: Simulation parameters and used datasets.

Parameter	**Value(s)**
Number of EDRs	14
Simulation Time	5 min. 30 sec.
Distance and link delay between EDRs at the edge level	2ms following values reported by recent studies [110]
Used Datasets	MNIST: 42,000 unique characters Workload: 250-500-1000 reqs/s
	Self-created CCTV Traffic Camera Feed Dataset: 3000 Video Framess Workload: 2000-4000-6000 reqs/s
	Alexa Wake-word Dataset: 365 Audio Filess Workload: 150-300-500 reqs/s
	Smart-Home General Command Dataset: 899 Audio Filess Workload: 150-200-300 reqs/s
Data Sizes	Ranging from 40KB, that represent MNIST letter images, to 5MB, that represent high-quality video frames
Link delay between EDRs	2ms
Link delay between a gateway and its adjacent EDR	2ms

• Back-of-envelope maximum requests/s calculations were made, for workload scaling per-dataset (please be aware that these calculations are for the ideal case, in which everything is known about the system and distribution is perfect), shown in Table 5.5.

• Software simulations complexity vs. Hardware implementation complexity and time scales - There is a big difference between the hardware resources used by a re-

Table 5.5: Table with computing maximum workload calculations.

Dataset	**Back-of-envelope calculation**	
	No-Reuse	Reuse
MNIST	4*15/0.09 = 666.7	+ ∼12%*666.7 ∼= 750
Traffic Detection	4*15/0.026 = 2307	+ ∼80%*2307 ∼= 4150
Alexa	4*15/0.456 = 166.7	+ ∼80%*166.7 ∼= 300
General Commands	4*15/0.456 = 166.7	+ ∼17.5%*166.7 ∼= 200

alistic simulation software and a hardware-implemented prototype, which makes the hardware implementation more desirable, directly applicable and dedicated. However, a hardware implementation would make the evaluation process much slower, complex and costly.

Performance Indicators

• Throughput - This is represented by the number of processed messages per second, and it should generally surpass the number of ingress messages per second. Its main modifiers are as follows:

–Sum of (built up) RTT delays;

–Dataset reusability vs. system processing capacity and dataset processing times

• Resource utilisation - This is the amount of processing power and network resources, consumed to maintain good QoS and/or QoE. In this case we are disregarding storage, due to the fact that reuse is a large factor in this study, and storage usage is negligible because of it. The main modifiers of this indicator are as follows:

–Edge- or Reuse-satisfied requests;

–Processing load distribution;

• Number of orchestration calls - This measurement is done for each algorithm, during each run, and shows how efficient the management system is in doing its job. This metric itself provides background for other trade-offs, between the providers' needs and the EDR Environment manager's performance (in resource utilisation, economics and QoE offered).

5.3.5 ReStorEdge Environment Simulations Evaluation

By doing some reasonable and calculated assumptions on the types, topics and similarity of the processing from the previous part of the evaluation 5.3.3, we have used the following similarity thresholds and resulting reusability results for our simulations-based evaluation (out of the 60%, 70%, 80%, 90% choices):

- MNIST - 60% - MNIST images are very small, they are single-character classification images and they are already reduced in size, so they do not need any pre-processing. Further, because of image quality, it would be very easy to classify the images with roughly the same borders as the same number.
- Traffic Detection - 90% - Due to the fact that cars can be very small, and on a highway, there either is very much traffic or very little, the smallest change, for the shortest time can prove that there is (no) traffic. That means that, while cameras in this area may not be of much use when one car passes from time to time, or even a few, it most definitely can detect larger changes in traffic flow, which is the exact application intended here.
- Alexa - 70% - Because of the repetitive use of this word, and the fact that smart assistants have to be triggered by it, it is especially relevant if the same user calls it more than once, repeatedly. In that way, the voice processing needed in a home would be much faster with reuse, but privacy-preserving, as well (due to hashing).
- General Commands - 80% - The same applies, as in the case of the Alexa dataset, but similarity has to be higher (possibly not caring much about identification - not 90%), but potentially preserving privacy as a priority, while keeping command interpretations accuracy (e.g. ”turn volume up” and ”turn [device] on”) high, without too much (pre-)processing.

It should be noted that the reuse rates associated with the similarity thresholds described above are used for the reader’s reference only here, as, within simulations, once queries are mapped to their bucket by the LSH algorithm, computation will be reused given the current size of cached results in that bucket. The predefined reuse rates are not used as inputs to the orchestration algorithms being evaluated. The orchestrator algorithms observe the amount of processing and reuse per bucket and make decisions on future bucket allocation based on these measurements.

The main purpose for this part of the evaluation is to show the diversity of data and the most appropriate approaches for different data types, topics and situations, depending on different system-, user-, provider- and environment-defined variables. We will also analyse the system from a design and development point of view in the end, to show how resource constraints, service constraints, dynamicity and data distribution can affect the way in which the system can be managed/configured, in order to appropriately balance these (and potentially other) trade-offs.

Throughput - We shall start by comparing throughput and unprocessed messages at the end of simulations, among the different datasets and the applied orchestration strategies. Looking at the throughput of the different orchestration strategy case results obtained for the MNIST dataset (Figure 5.3), we can see from the first look that the CPU utilisation strategy performs best among all orchestration strategies and across use cases. In the traffic detection dataset, the throughput provided by both the queue-based and the CPU-reuse-based strategies have a (mostly) positive impact(Figure 5.4), with the CPU-Reuse strategy providing continuous improvement to the no-orchestration, reuse case. The throughput of the CPU-Reuse strategy in the case of the Traffic Detection dataset matches the performance of the queue-based and CPU-usage strategies in all workload cases (Figure 5.4). While the throughputs of the Alexa dataset demonstrate a good overall performance of the orchestration strategies, in the general, smart-home commands dataset, the CPU-Workload orchestration strategy provides the best performance improvements overall (Figure 5.6), detaching itself in the measurements of performance indicators, even though queue-based may be better in some cases/settings.

For all datasets, we can also see that when the system is clearly overflowing with requests, the queue-based strategy performs best, but the amount of unprocessed requests still remains too high to keep up with user/provider requirements

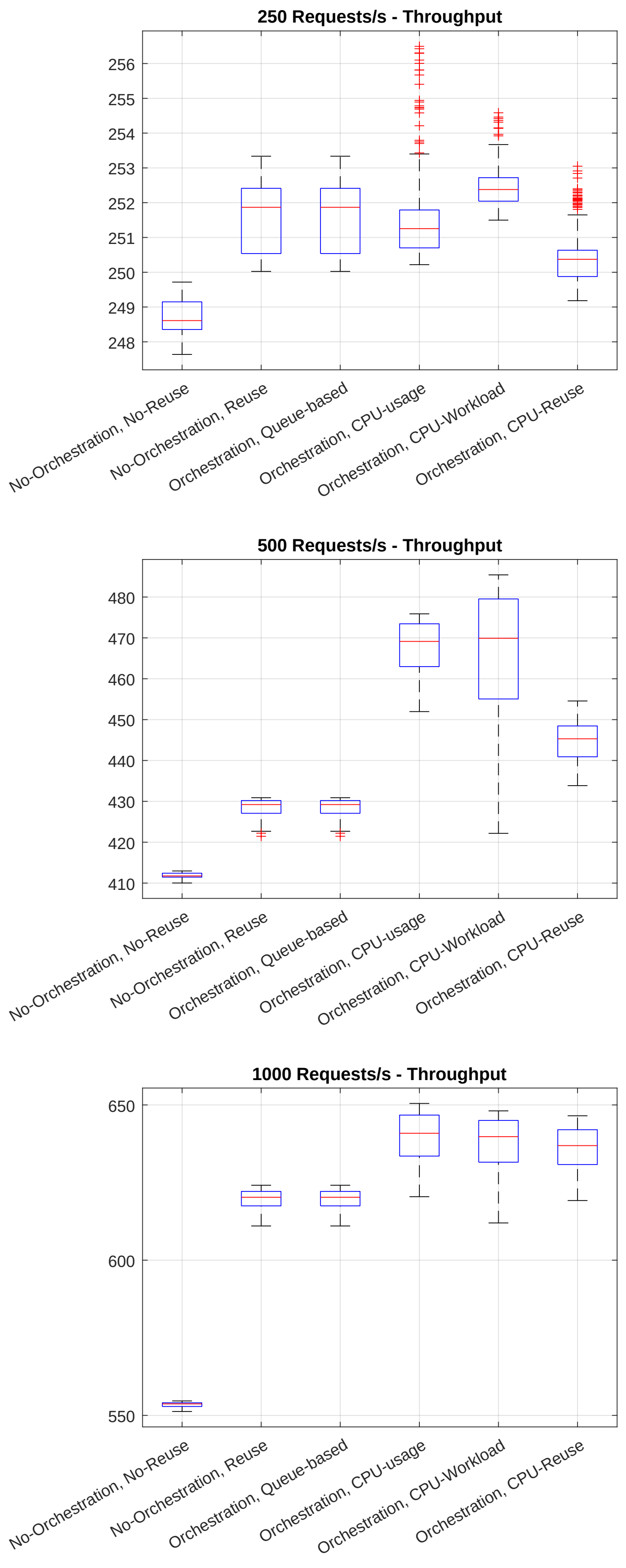


Figure 5.3: Throughputs for different rates with MNIST Dataset.

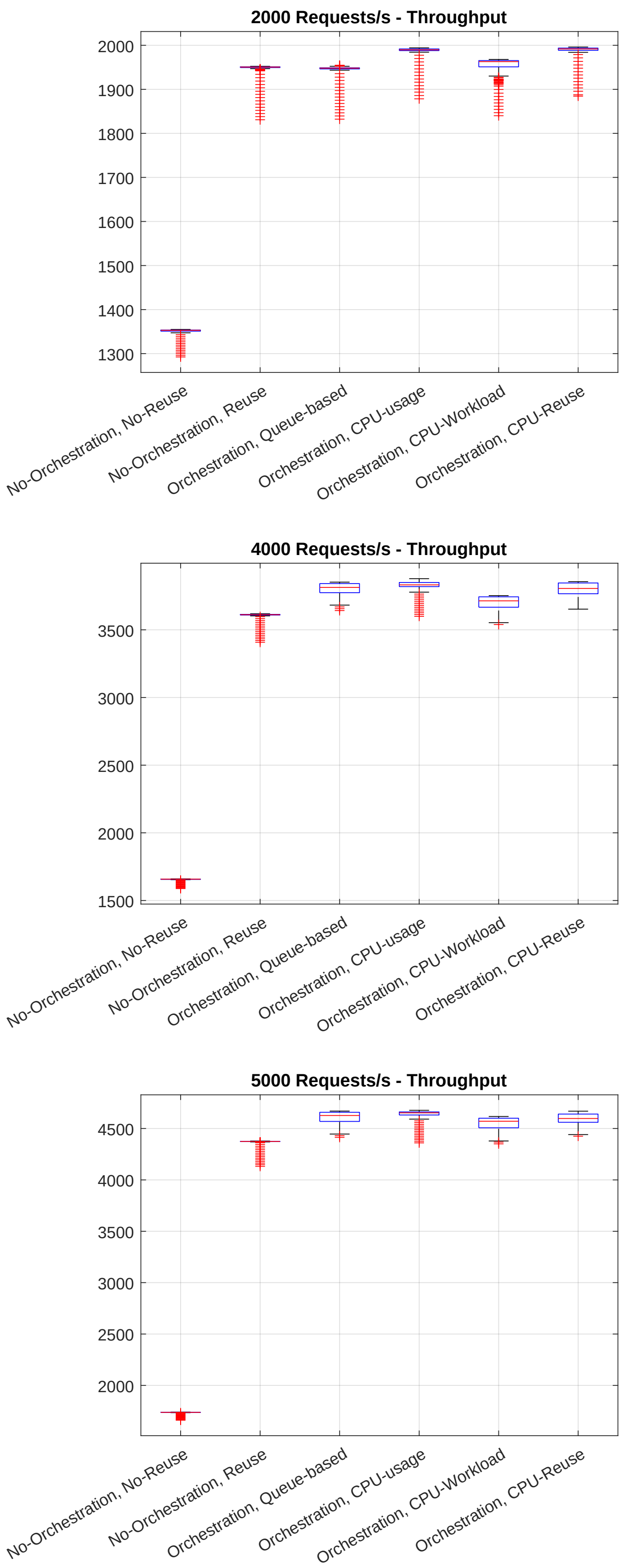
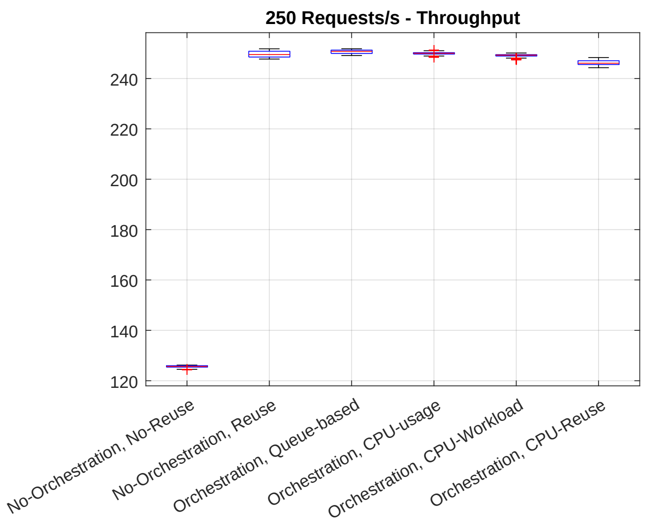
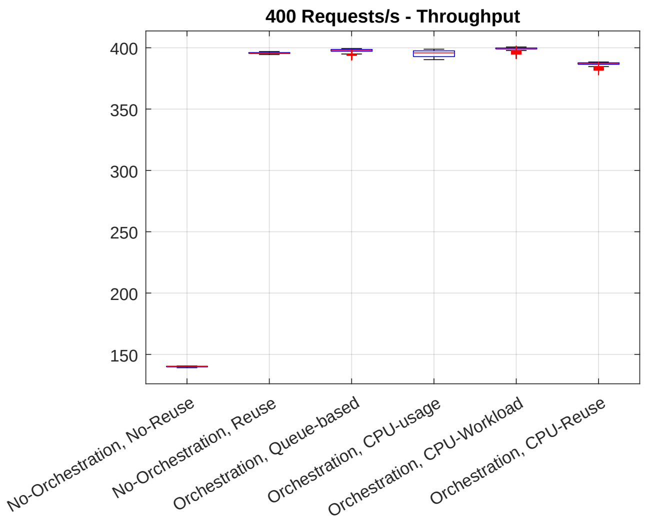


Figure 5.4: Throughputs for different rates with Traffic Detection Dataset.





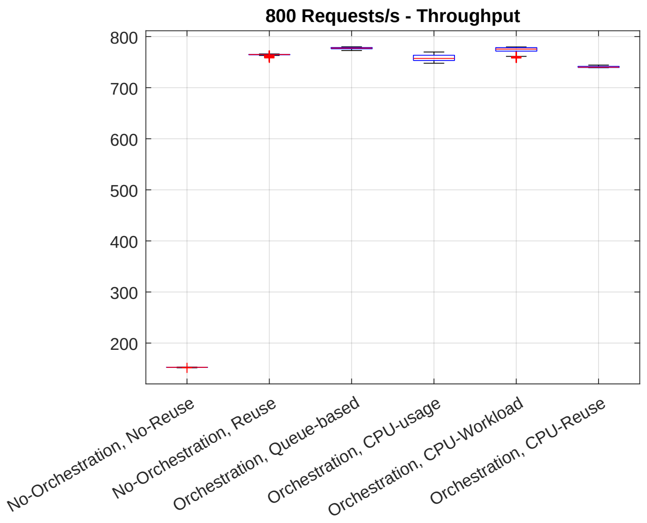


Figure 5.5: Throughputs for different rates with Alexa Dataset.

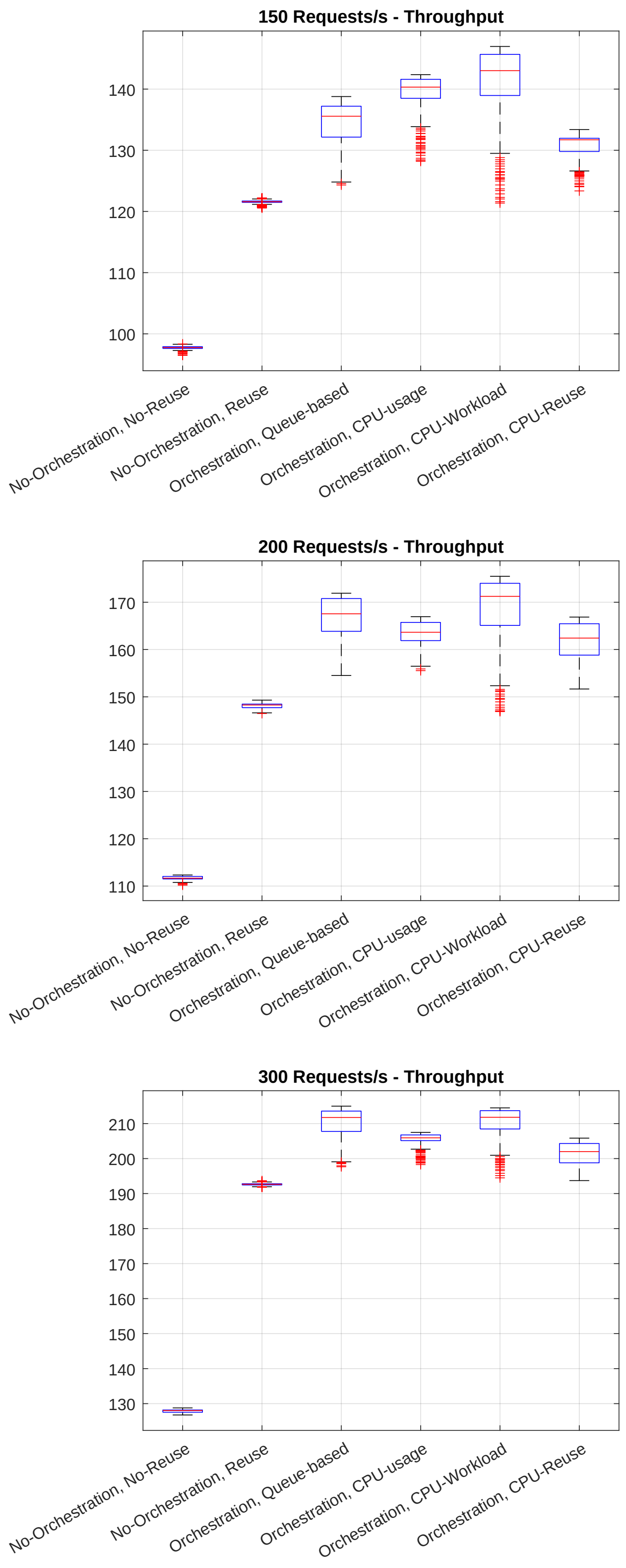


Figure 5.6: Throughputs for different rates with General Commands Dataset.

without heavy storage/later reuse, to catch up.

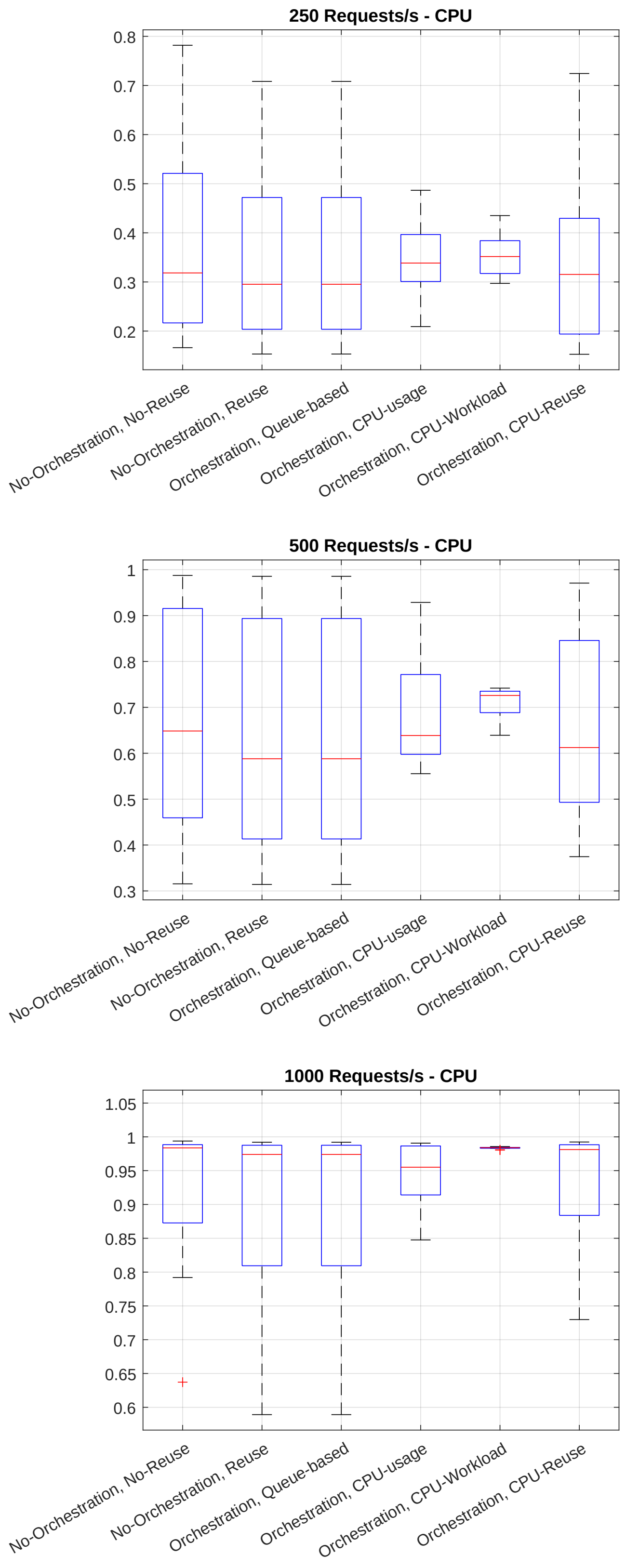


Figure 5.7: CPU utilisation for different rates with MNIST Dataset.

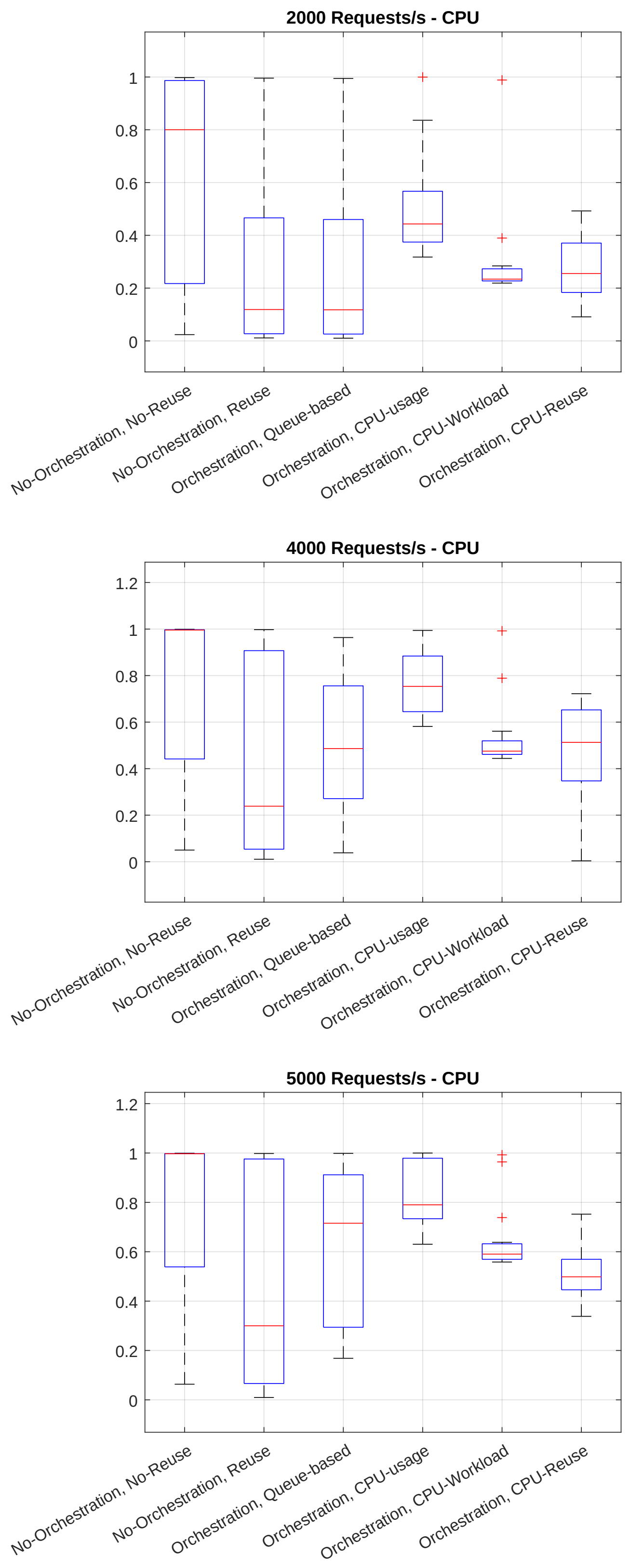


Figure 5.8: CPU utilisation for different rates with Traffic Detection Dataset.

CPU utilisation - Considering the throughputs, we shall now look at CPU utilisation, to consider system resource utilisation and transmission performance. In this regard, we saw from the first part that the CPU-usage strategy of MNIST fares very well, even though the Workload strategy came in to a close second. With the Traffic detection dataset, the situation is wholly different. The CPU-usage is the most balanced and well distributed in the case of the CPU-Reuse strategy, with the CPU-Workload strategy achieving very good load balancing across EDRs (with higher median load in some cases). Thus, the most stable and efficient strategy in the case of the Traffic detection dataset is the CPU-Reuse strategy. In the case of the Alexa dataset, there is a distinct advantage of the CPU-Workload strategy in regards to CPU utilisation, due mainly to the better distribution of CPU-bound requests across EDRs. However, the queue-based strategy shows more localised CPU utilisation management (Figure5.9), which puts it in a good place, as a contender for the best strategy to work towards the best performance improvement associated with the Alexa dataset, because of less load balancing and more overall CPU-utilisation reduction. The general commands dataset is where the queue-based orchestration strategy shines brightest, as it provides the best results for this dataset (Figure 5.10).

Orchestration calls - Considering the above, there is a part we cannot neglect, and that is orchestration processing and command overhead. This can become a real burden if the feedback system is not controlled appropriately. For MNIST, while the queue-based strategy indicates potential in the overflow case, the impact it has is small, while it also imposes a much higher orchestration overhead (Figure 5.11a). In the case of the CPU-workload strategy, simulation times are longer, so while fewer orchestration calls may be necessary for it to have a greater, better effect on the performance, the actual processing time taken to execute the orchestration takes a lot longer than in the cases of the other strategies. Consequentially, orchestration timing is influenced by computational power and the start of the orchestration cycle at a certain point, for it to be able to execute each epoch. The timing of each orchestration's effective feedback window is also affected.

The orchestration calls plots indicate a definite winner in the case of the Traf-

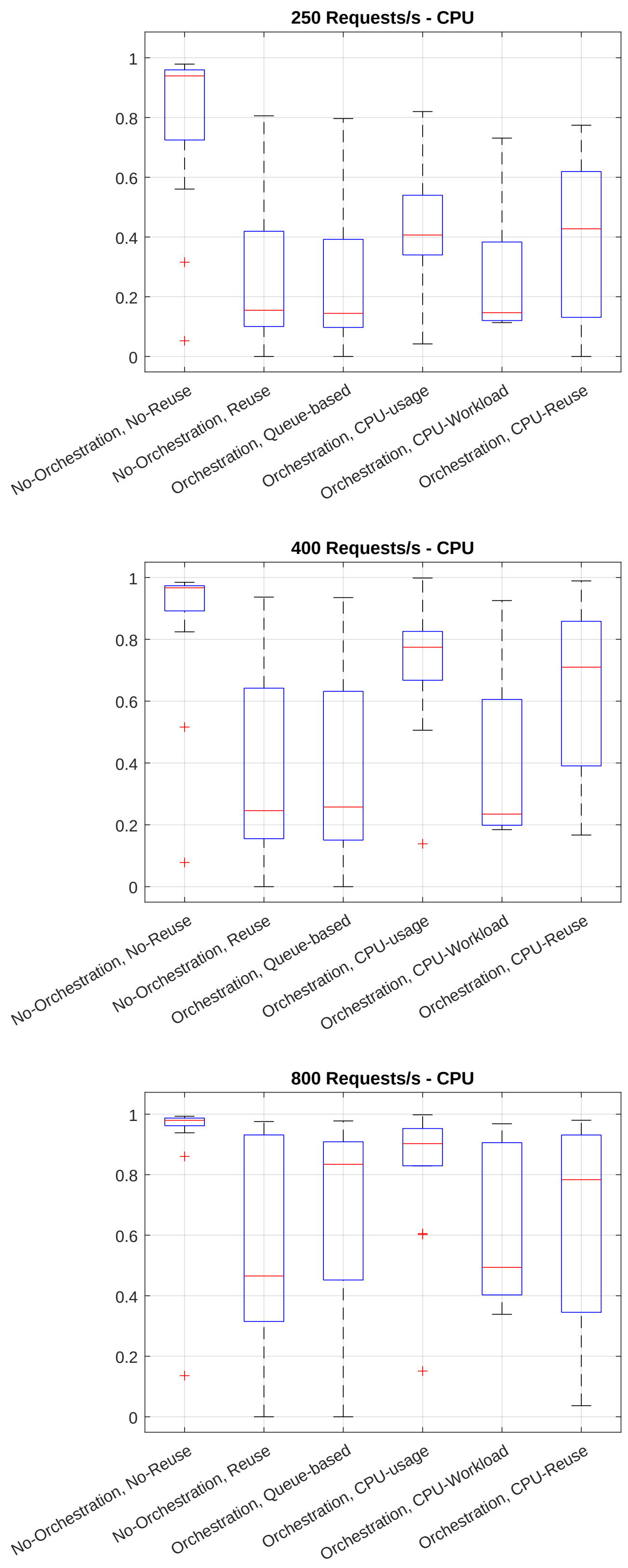
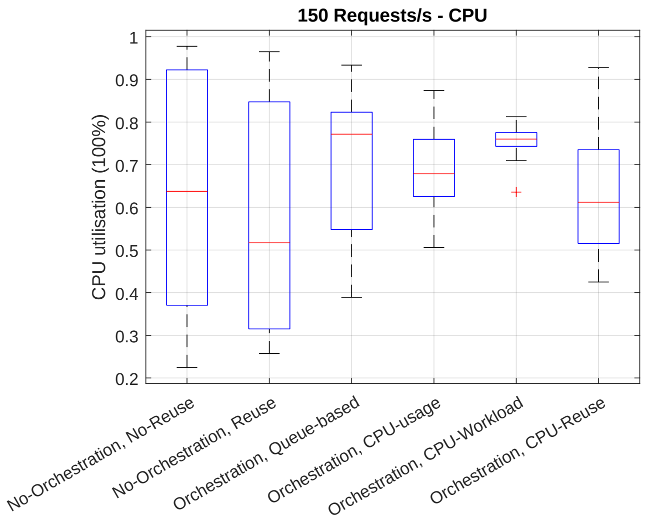
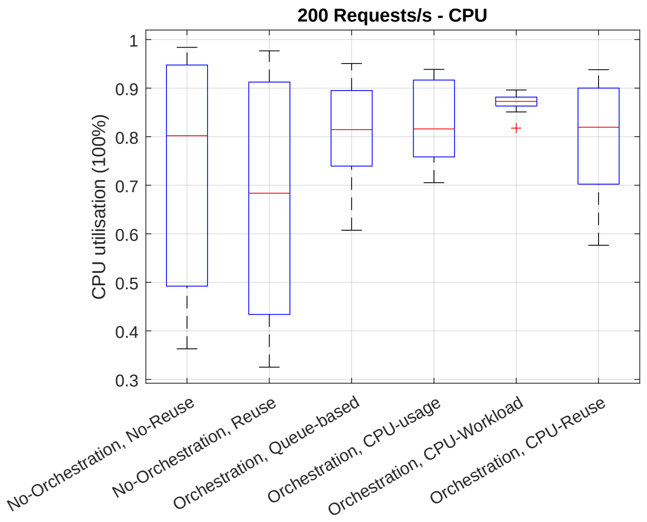


Figure 5.9: CPU utilisation for different rates with Alexa Dataset.





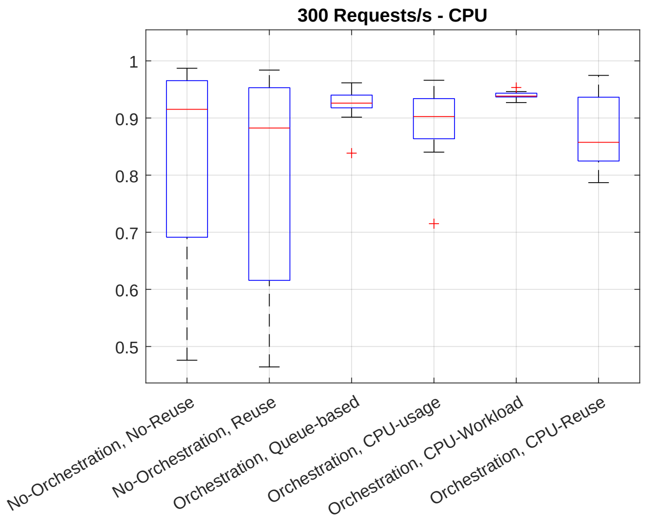
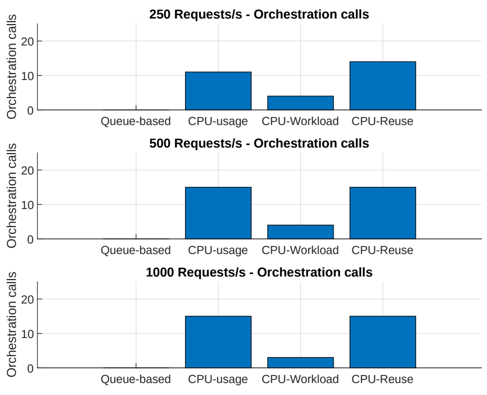


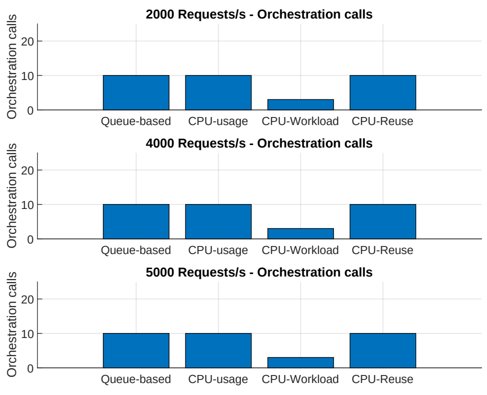
Figure 5.10: CPU utilisation for different rates with General Commands Dataset.

fic Detection dataset, as overall, it registers the fewest calls, and considering the results it provides (explained in the above paragraphs), it seems, for this reason, that it provides the best performance improvement, with the least amount of intervention/resource utilisation. In the case of the general commands dataset, the queue-based strategy makes itself stand out, having the highest (but reasonable still) number of calls throughout the simulations.

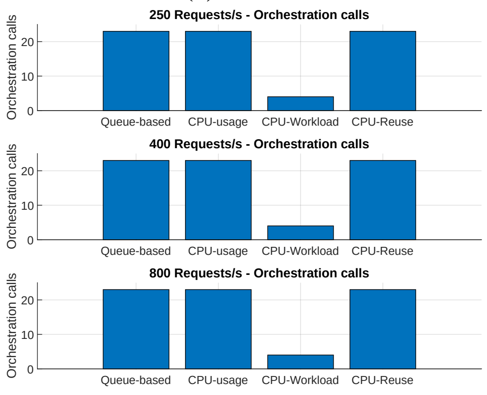
NOTE: The CPU-Workload strategy was limited to two interventions throughout simulations, which is why it is not considered for this specific comparison. Another dataset-specific characteristic to note and consider in the above comparison is that the different datasets had different kinds and amounts of data generation rates, which also influences the complexity, frequency and potential reuse for distinct data generation patterns.



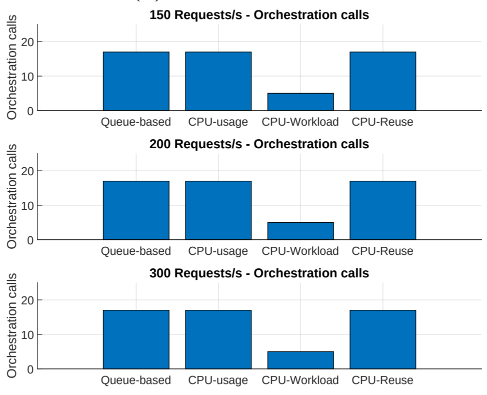
(a) MNIST



(b) Traffic Detection.



(c) Alexa



(d) General Commands

Figure 5.11: Orchestration call counts for different datasets.

5.3.6 Evaluation Conclusions and Deployment Options

Considering the above discussion, based on the results of the evaluation, the background of today's technology and user needs, we can draw the main conclusions of the above evaluation, also associating the orchestration strategies with data specific to the different datasets and their associated use cases/user environments, according to the performance improvement exhibited by each strategy.

1. The algorithms implemented in this evaluation were four relatively simple strategies that were used to demonstrate the benefits of the idea of orchestration in this edge computation with storage and reuse scenario, with different environments and use cases in mind, and their refinement to improve performance further is a subject for future work and engineering.
2. The four orchestration strategies showed improvements over the simple reuse case in terms of CPU load balancing and request throughput. The degree of improvement of each strategy varied with the application use case due to the computational complexity of processing queries for that application, the nature of the served data and the degree of query similarity in that use case which impacts the amount of reuse that is possible. A single orchestration strategy did not perform best in all combinations of use case and workload level, which indicates that a combination of orchestration strategies would be beneficial and that designers of future algorithms should consider multiple factors:
• Resource Utilisation - this represents and is represented by both processing resources and storage. We only analyse processing resources in this case, as storage is less affected by the cases studied here, due to reuse;
• User-/Application Provider-imposed QoE and QoS - these service metrics could be integrated as ”tuning knobs”, in potential designs and implementations of the system;
• Orchestration Overhead - all the management and optimisation comes at a price, and the price is comprised of more computing resource utilisation, combined with timing constraints and a small, periodic routing overhead. The special case, of the workload strategy brings up a few new topics of study in this regard, as well, in-

cluding bucket splitting, orchestration timing influence on computational power and orchestration/epoch feedback window sizing. These have to be considered when working on the system design and development based on the system defined within this paper.

Looking at the system, as a whole, and with the above in mind, we conclude that, while reuse reduces the processing load on EDRs and hence increases the throughput of requests, orchestration improves the distribution of load amongst the EDRs and further increases request throughput which improves the QoE of the users and better manages system resources, for more resource utilisation control.

One option for deployment of the orchestration algorithms presented in this paper is to implement them as part of the Network Function Virtualisation (NFV) architecture [119] as an application in the Management and Orchestration (MANO) framework [120]. A MANO environment of kubernetes clusters or OpenStack-managed servers would provide a suitable environment for the execution of optimisation functions that: 1) gather usage and performance data from distributed EDRs; 2) calculate bucket placement strategies based on the collected operational data; and 3) implement the movement of buckets and caches between EDRs and to configure the associated routing table changes.

5.4 Further Workload Study

As mentioned in the design section, above, the decision was made that the above study needed more proof for impact and efficacy, thus, the decision was made to study the assumption of random workload allocation, towards obtaining a fourth, workload-based (and possibly based on times of day/locations later on - up for discussion), orchestration algorithm. Thus, this short study is presented as follows: the request generation patterns are studied, in order to find diversity and research potential for this fourth strategy, then the orchestration strategy is briefly presented and explained, for it to then be evaluated against the rest of the strategies.

NOTE: The new results have now been included in the above paper, and will have already been (at least partially) addressed. This is a more in-depth analysis of the short workload-based study.

5.4.1 Workload Analysis

The approximate request generation speed was taken into consideration, and because the plots suggested an almost-linear development, the decision was taken to develop a new, workload strategy. The workload plots considered were of EDR- and bucket-specific request generation and ingress quantities, and are presented in the plots of Figs. 5.12, 5.13, 5.14, 5.15 and 5.20.

Both the whole population of requests generated and strictly the population of processing-bound requests will be analysed, in order to show the needs and effect of the CPU-Workload strategy. This analysis was done for the feasibility of implementing the CPU-Workload strategy. However, a few more interesting findings and discussions resulted from it and the strategy's application.

In Figure 5.12, the final counts of processing-bound requests per-EDR (per-EDR workload) are shown. The CPU-workload strategy can be seen to have a big impact here, and this can also be seen to have a sizeable impact on the resource utilisation metrics, as well.

Considering the above plots and the disproportionate generated requests per bucket, the decision was made to also define a (Total-)Workload-based bucket-splitting mechanic, which was implemented strictly with the CPU-workload orchestration, and the results show a much better distribution. (Figure 5.26 and 5.27)

However, from an implementation point of view, the application of the bucket-splitting technique also means that extra complications would be introduced, as in a real-life implementation, this mechanic would introduce the possibility of fragmentation, specifically relevant for reuse. On the other hand, if it would be introduced strictly for CPU utilisation (no-reuse buckets), it would have the highest possible impact, due to the redistribution of needed computational resource utilisation.

5.4.2 Workload Orchestration Algorithm

For a more complete picture and to highlight development potential for the new approach to the orchestration of storage, computation and reuse, the workload strategy was developed and evaluated. The algorithm was presented in Algorithm 4

5.4.3 Algorithm Evaluation and Conclusions

The more stabilised and better distributed workload imposed by the CPU-Workload orchestration algorithm can be seen in Plots 5.16 - 5.19. A further observation we can make is that in the cases where some buckets can be much more popular than others, the CPU-Workload strategy can have less of an impact on workload (and thus on certain performance indicators, as well). This observed problem was further studied and will be presented shortly.

As it can be seen in the above study (submitted for review), the workload-based algorithm can outperform some of the other orchestration strategies, in most of the case studies, however, it is less specialised, and while it generally may have a higher impact at all workload thresholds the other, more specialised, strategies can surpass its performance advantages with their more overhead-efficient orchestration algorithms and sometimes in regards to resource efficiency and . The CPU-Workload strategy also imposes quite high system overhead, in terms of timing, statistics exchange and network resources. While the above can be advantageous in systems with very high expectations in regards to performance (both for resource utilisation and QoS), the performance-orchestration overhead trade-off needs to be taken under very serious consideration with this strategy.

One important conclusion that could be drawn from this kind of orchestration is that, while bucket numbers may stabilise over a certain amount of time, if all buckets of a domain are taken into consideration for workload redistribution, the orchestration time would potentially be longer than an epoch, and the algorithm would be ineffective, because of the amount of time/processing power it would take for the algorithm to gather all needed data and the timing of the orchestrated data itself.

Another important discussion point, resulting from this further performance analysis is the influence that eventual consistency, information/label/bucket validity and retention times could have on the efficiency/efficacy of orchestration. Even further, an important point/metric, parallel to the above in regards to orchestration efficiency and efficacy, is the workload, since the analysis is fresh in mind. The amount of workload and its variations in time and space (e.g. different parts of a city, at different hours of the da or even different days of the week) could influence the way in which the system reacts to outside stimuli in many more (and probably better) ways, and increase its impact, depending on the different situations it finds itself in.

5.5 Workload-based Bucket Splitting

Considering the results and comments of the above sections, and that workload-based management cannot be very efficient, if one bucket is more popular than the others, but by a large margin. This can be seen in Figure 5.20.

Thus, we have decided to do a more detailed study on bucket splitting. With this in mind, we split the buckets as follows:

1. The sum of all workload, across buckets, is taken, split among EDRs (dividing by 15), which results in an average per EDR. Any bucket that is over that threshold is to be split the least amount of times, so that each subdivision is smaller than the per-EDR average.

2. The split buckets then have the same amount of predicted workload as the original (in the view of the orchestrator, at least, at the time of orchestration)

3. Stored results are then deduplicated to the new, split-bucket-EDRs

4. The splitting process will be done before any re-allocation

5. Need to allocate the new requests, entering the system, at each “receiver”, to the different ”sub-buckets” - should be done with round-robin allocation

5.5.1 Workload-based Splitting Evaluation

Considering that, theoretically, the splitting of buckets could greatly improve performance, it was tested against the normal, workload-based strategy. Thus, we present the results in the subfigures of 5.21, and critically analyse them.

In the cases of the MNIST and General Commands datasets, there is no splitting observed, due to the lack of need. As shown in Figure 5.20, all buckets were less "requested" than the average split of requests over all EDRs, so there is no reason to present them. The calculation of the averages are based on the total, epoch-specific counts, just before each orchestration round.

We only compare the performance (Figures 5.22, 5.23, 5.24 and 5.25) and workload (Figures 5.26 and 5.27) distribution for higher workload cases (6000 req/s for Traffic and 1000 req/s for Alexa), of the relevant cases (Traffic and Alexa datasets), for clarity and efficiency, considering the previous results presentation and the above-mentioned figures. Note that in the case of the workload plots (Figures 5.26 and 5.27), the figures indicate the total workloads for each EDR, and not only processing-bound workload (which is the only workload type considered in the CPU-Workload orchestration strategy).

While performance improvements were observed in both the Alexa and Traffic dataset study cases (with a very large difference in load balancing performance - Figures 5.29, 5.28, 5.31, 5.30), the scale of the impact, compared to simply applying the woorkload strategy on orchestration (without splitting the buckets accordingly) is low (Figures 5.22, 5.23, 5.24 and 5.25). Further, if the processing-bound buckets would be **strictly** considered for splitting, there would be no problem with the reuse of data. However, if reuse-bound buckets are to be split, there is great potential for other issues (like fragmentation) to occur in the LSH- (bucket-) and application-/label-matching-based rerouting of requests.

Noting, for both datasets, Figures 5.28 and 5.29 cannot be directly compared to 5.30 and 5.31. The former represent the orchestration "expectations", after the buckets have been reassigned, and that is also the reason for which the measurements remain constant for the remainder of the orchestration periods. The latter represent the results of the bucket splitting and orchestration, combined, and they represent a final simulation snapshot of the actual measurements of per-EDR request counts. With that in mind, we can look at the results, and, by critically analysing the plots, we can conclude that bucket splitting did not provide enough of a performance impact to expose in the above study. However, this short study on workload-based strategies has exposed a new orchestration strategy that can potentially be very beneficial and it has provided a few more (promising) areas of interest for this research, which cannot be ignored.

Thus, if it is not known that (specific bucket-associated) requests have to be processed towards service satisfaction (without reuse), sensible reasoning would indicate that bucket splitting is not worth the extra calculations, computational work (resource utilisation) and time to orchestration.

NOTE: All the above figures are representations used strictly for the comparison of results obtained from the application of only the workload strategy for orchestration and the applicaton of **both** splitting and the workload stragety. The rest of the results relevant for workload strategy application can be found in the above study.

5.6 Potential Further Work

This section shows the progress of this PhD in research and impact, and proposes an interesting research progression direction from the work: In an ideal world, we could integrate better versions of the ”research artefacts” (designs, simulators, simulations and evaluations) into a complete Edge computing and storage system evaluation environment, which could help engineers with design and to do pre-deployment/prototype system evaluations, for different parts of the system. The following sections approach this subject, exactly, by putting together the work in the previous chapters and creating a more complete image of a fully-fledged system, by also filling in most gaps that were not approached (but without going into deep, systematic analysis).

Locality- and context-based optimisation/improvements

Since labels can be used within the environment, for the efficient management of information, along with the more exact identification, tracking and flexibility of hashes, these two can be used together, over longer periods of time (or by specific services/applications, interested in similar data), to identify commonalities within services and data placed and delivered within an EDR environment, for example. Another potential use for both of these routing and application-layer identifiers would be to identify gaps in services needed in the area, but unsatisfied, or satisfied in an inefficient way, due to others using up otherwise useful resources, or to use the needed metrics(e.g. satisfaction rates and overheads - identified in the SEND part of the previous chapter) , to determine a better way to implement labelling and hashing within the system. For this, the potential of Locality-Sensitive Hashing (LSH) and application-/domain-specific label management will be exploited.

A good example to show this principle would be illustrated by having two applications sharing similar data, looking to apply their own services/functions to the data, which is stored in close-by locations (EDRs). These pieces of data come as one, sharing a hash and even labels, if the different applications either interpret the same label identifier (for example part of the hash, or a number - or status - identifier, within the MetaData) differently or they have different labels for their own applications, and that data is identified by both labels, that data can serve both applications - thus needing less routing and storage resources within the EDR domain. In this case, a good trade-off would be to only store the similar data once in the EDR environment, and if the appropriate metadata/hashes are available for that data, that (collection of) packet(s) could either be completely sent by the main storage repository, or found and replicated (with a scope of 0 replications) and sent with the appropriate labels back to the application providers or associated users, for request satisfaction. Thus, this algorithm makes for an efficient system and complements the approaches of the data storage (re)allocation strategies presented in the last chapter, by using them as the main, empowering entity to detect similarity, optimise for satisfaction and/or overhead and thereafter make the system more efficient by reducing resource usage through the use of the LSH mechanism.

5.6.1 Light-weight information locality improvement

The most essential parts of the envisaged system are the Edge Data Producers (whether mobile or static), as they are the main origin of data. These devices should generally be associated with at least one Application/Data/IoT Provider and their servers. Their produced data should always be signed with a public key either pre-set (offline bootstrapping) or refreshed (after ERP connection) by their provider.

Easily implemented by Edge Data Producers and Application/Data Providers

The hash domains, labels and data associated with producer-specific packets are to be defined by their associate Application providers, for full compatibility with EDR environment management systems and with the Application Provider's EDR-instantiated functions and/or data. These hashes and labels are simply designed, used and periodically updated by Application/Data Providers, updated/bootstrapped "offline" on their Edge Producers and developed for the improvement of their devices, knowledge, services and users' experience.

Very light-weight and adaptive hashes and labels used for routing

Even though they play a very important role in the ERP system, the Edge Data Producers (EDPs) will always be unreliable with regards to power, wake times, connection times and data collection/sensing, sometimes benefiting from mobility as well. All these factors affect their network and connection reliability. Thus, we have a generally unreliable data source that plays a key role in many of the newer networks, due to the enormous amounts of data expected from them. This is the main problem we start our system's development with.

The produced data has to be transmitted as soon as possible after creation, due to the system's low storage capacity. While most of the newer end devices have a full-stack transmitter(-receiver) on-board, they mostly lack any other available computational or storage resources. Thus, the on-board network-layer operations have to be as limited as possible. This is why a short-length, simple naming/hashing system and a Data push mechanism are needed at this stage. On the other hand, while hashing has to be simple and short, associated "MetaData" has to also be included in the packets, for Intra-EDR-domain function and data management and Edge-domain analytics. In the "MetaData" part of the header, different tags (e.g. the data attributes mentioned in the previous chapter, labels and other tags, for "overlapping" [32] purposes) and restricting function, shelf-life, hop scope and/or aggregation can also be provided. This information has a big impact on the management and analysis of processing and data distribution in ERP-managed environments, as we have seen and will explain in the next sub-section.

As Edge Data Producers are generally very constrained, with regards to network and processing/storage, data needs to be accompanied by light-weight, yet effective network identifiers. In this case, the best solution is the combination of hashes and labels, which makes the data and/or request messages sent from the Edge both light-weight and very effective in interpretation, malleable, with regards to environment (EDR) usability and information efficiency, and secure, because of their traceability, hash-verifiable authenticity and localised and secure environment, housing the data.

5.6.2 Service Provision

The production, routing and distribution of functions is among the simplest parts of the system. The production of a function can be achieved by any network-registered Data/Application/IoT provider, which is approved by several peers (ERPs and other providers), to provide functions for execution within any part of the network. After a function is produced, according to certain execution, compatibility and/or size/complexity standards, it can be registered into the network, with its name and origin, then routed and distributed to the entities/environments where it is needed most or most popular.

The importance of data proximity and storage development in this context

The distribution of function is done naturally. Once a function is produced by its provider and is made known and available to the network (registered into the provider's NDNS domain), it can be requested by any ERP domain which is supposed to use the function for processing. Once this first move is done, the function can spread around the network, to associated ERPs and other Data/Application/IoT Providers. The distribution should occur naturally, securely and within some restrictions set initially or updated subsequently, with versions of the function and public key epochs, set by the originating provider. Once an ERP has obtained a function, it should hold a copy of its executed function for as long as it is executed. As soon as the function stops being executed within an ERP's domain, if there is no copy of the same function in the close vicinity of the ERP, it should keep it indefinitely, otherwise drop it after a determined amount of time (by policy), also announcing it beforehand.

The placement of functions directly affects both EDR domain performance (QoS and resource efficiency) and data placement (in some cases more, in others the influence is reversed). Since both the functions placed and executed and the data placed and (re)allocated in EDRs within the EDR environment are flexible and adaptable to fast exchanges, they are ideal for a dynamic, mobility-friendly environment, while also being adaptable towards optimisations in the long run.

On the other hand, the intra-ERP ”life” of functions is more complicated and different. ERP domain servers (managers) are the main policy enforcers and management providers for their respective EDR domains. They achieve this by enforcing specific commercially- and economically-driven policies in their EDR clusters, in the context of the data that they obtain and the connections that they have with the Data/Application/IoT Providers' servers and/or data producers. The specifics of management development and parameters on which management can rely will be described and discussed in the following sub-section.

Function naming and mapping is done in a simple way, with functions originating from Application/IoT Providers. If the Provider is situated in the cloud, then the function shall be produced with the Cloud Provider's name, followed by the IoT Provider's name, and finally by the function name. The Function would then be advertised to the appropriate authoritative NDNS server, for example. Thus, the hierarchy of the function name is determined by its authoritative NDNS server.

If a function's execution is requested within the data request name, but the function is not present in the NDNS system, a NACK can be returned for the function. However, the NDNS query could continue, with forwarding hints instead, providing the data location, for later function instantiation (after production) and, possibly, even distribution from the specific ERP where it was first instantiated.

5.6.3 Repository Environment Management and Routing

Both functions and data have to be managed and distributed according to their various hashes, applications and ”MetaInfo”, within the different ERP environments. This, as described above, is done by ERP servers (managers). We will now look at the main data and function parameters (”MetaInfo” from data and ”selectors” and ”guiders” from function Interests) that will guide the way in which ERP servers manage their respective domains.

(Data) Location, location, location

As ERPs would be a hub for all these transactions and deployments, Client-Server relationships would also be blurred within EDR environments. This uncertainty is due to these environments mediating all the transactions and policing, which is needed more and more due to the nature of today’s Internet needs and developments. This approach provides a view of the client-server state only once specific actions are taken.

Light-weight information locality improvement

Data Producers send data out into the EDR environments with specific MetaInfo, for constraining or aiding the ERP servers and their monitored and/or managed EDR domains in handling and/or delivering sets of data. These MetaInfo specifications are set up either at the Data Producer’s inception, as a default policy, or updated each time their data is accepted by an ERP environment, which has received the necessary credentials and/or policy to handle the Producer data. Furthermore, for any Edge Data Producer device, these values are either fixed throughout its lifetime or updated periodically (over long periods of time), or as mentioned above, once connections are created, updated periodically (if needed), by their provider owners.

The selectors provided by the users or providers in their data and/or function (result) requests need to be of a known range of suffixes that the producers can provide. Another condition that these fields must respect is to be used strictly as data filters, for data to be selected by name, scope or series, e.g. aggregation flag, aggregation number (content quantity), starting and direction identifier. Otherwise, selectors are not used for much, and their part in the header should not use up much space either.

Function- and information-based routing and storage improvement

Data management and distribution within ERP domains are done independently, by EDRs, for the internal routing and distribution of independent data chunks; then distributed and computed (generating feedback) in (aggregated) datasets, with the help of a (potentially adaptive) data storage distribution management system. The ERP local monitoring and management entity has a central role in its domain and the surrounding routing environment. Locally, the management entity has to provide support for distribution, locality of reference for data and performance monitoring of its domain. Monitoring and management are done by using the MetaInfo extracted and provided by its EDR environment as feedback. The main management task of the ERP is to adjust the different resource and network traffic allocation to different parts of its domain.

It must be mentioned that all the data which needs aggregation for function execution needs to have MetaData/MetaInfo that accepts the conditions of processing and data handling.

Function Data MetaInfo:

As both functions and data are associated to and come along with the same labels in their MetaInfo field, the decision was made to move on with the point of the function requests coming hashed, thus implementing label-and-service-specific LSH and, due to hash and data connection, similarity checks, towards function reuse. Since this was not enough material and complexity to develop the design and provide an innovative and impactful design into the research community, the decision was made to go one step further, and with the realistic data and patterns obtained from the local results, to obtain a realistic setting, to implement orchestration strategies, towards obtaining the best possible QoE, system resource usage and orchestration overhead trade-offs possible, from our use cases and developed strategies. This implementation provides the step of function-, hash- and service-driven optimisation, needed for the last stepping-stone, towards the system envisaged at the start of the PhD.

On-the-fly routing and resource efficiency improvements

Because of the different parts/modularity of the hashing scheme, if new parts need to be added to this system, then they need to adhere to this modular design, or adapt it so that (part of) it is used in the new design, if interfacing is needed and standards are to be respected.

Regardless of implementation or modular design used, the strategies and associated system have interconnecting parts that are influenced by the strategies and the services it provides. The basic principles of functioning, for each part of the hashing/naming scheme have to be implemented in a certain way in order for the scheme and system to work together. Thus, the hashing scheme implementation is very important for the modularity, connectivity and precedence of the system's modules, functionalities and effectivity.

Considering the **Locality- and context-based optimisation/improvements**, explained above, in subsection 5.6.3, there are a few more, dynamic management and routing decisions that could be made. Since management strategies have been defined and used up to this point, their outputs, or rather their statistics and part of their actions could also be used further, within the decision process of managing data and applications within the EDR environments, by the (ERP) servers, to further improve the efficiency and performances of the system and to more efficiently manage labels, in a more intelligent way. This could be done by dynamically improving allocation and provision models, depending on the patterns of data and dynamic updates, via machine learning models and AI, which can implement network adaptivity 'on-the-fly' (even though this term is not perfectly fitted here since normally it refers to passing messages and not live-system updates).

5.6.4 Repository Environment Name Exchange

Maybe ERPs could advertise their offered services as names (through their local servers)? Services could include abstractions of function and data classes, such as video compression, IoT storage, aggregation and trending (applying average, summation etc. on datasets); or more precise ones, e.g. storage-only, compression-only, BW-efficiency-optimisation; or very functional, routing-and-direction-based ones, e.g. exclusively proactive (data comes with pre-associated functions and is processed and pushed to providers/clouds), exclusively reactive (EDRs only accept requests from providers and their users), exclusively reactive, with one-off requests (intended strictly for providers' user applications) etc. The service names (similar to the type of service field in IP headers) could be placed in between the function and producer data names (if function names are included), due to the fact that functions could either be restrictive for the type of service they support and services can restrict data availability for the functions executed, function execution scope and/or function result distribution.

Regardless of whether ERP service names are included in the network-wide naming scheme or not, it can be agreed that the EDR clusters represent the biggest naming inflection point in the whole network, with regards to data retrieval, processing, naming and/or delivery. The EDRs have to put all these pieces together, in a very fast and efficient manner, which is why these systems' complexity, architecture, management and impact factors have to be considered, researched and eventually designed very carefully, to provide good reasons for their implementation's viability. These can be the most complex systems within networks, yet be used to solve the network's problems in very simple and straight-forward ways (at least apparently).

5.7 Conclusion

This chapter presents an LSH- and Storage-based Computation and Reuse management framework, named ReStorEdge. This framework has the main purpose to improve on all resource utilisation and QoS performance of the system, by exploiting its addressing, storage, computing and data reusability capabilities and potential. Its realisation is possible through the deployment of EDRs, starting at 1-hop away from users, seamlessly interacting with network traffic originating at the Edge through a logically centralized management plane, ReStorEdge implements orchestration strategies, towards better function and data placement, based on data hashes obtained through LSH and their associated buckets. For the evaluation of the system, we implemented a local EDR, with LSH, similarity checks and storage, for the purpose of demonstrating the mechanic for each dataset, proving that the process is legitimately useful and providing realistic reusability values for a larger scale, simulated environment, with 14 EDRs and the same amount of gateways, generating realistic amounts of data, to prove the feasibility and analyse the performance of the developed orchestration strategies.

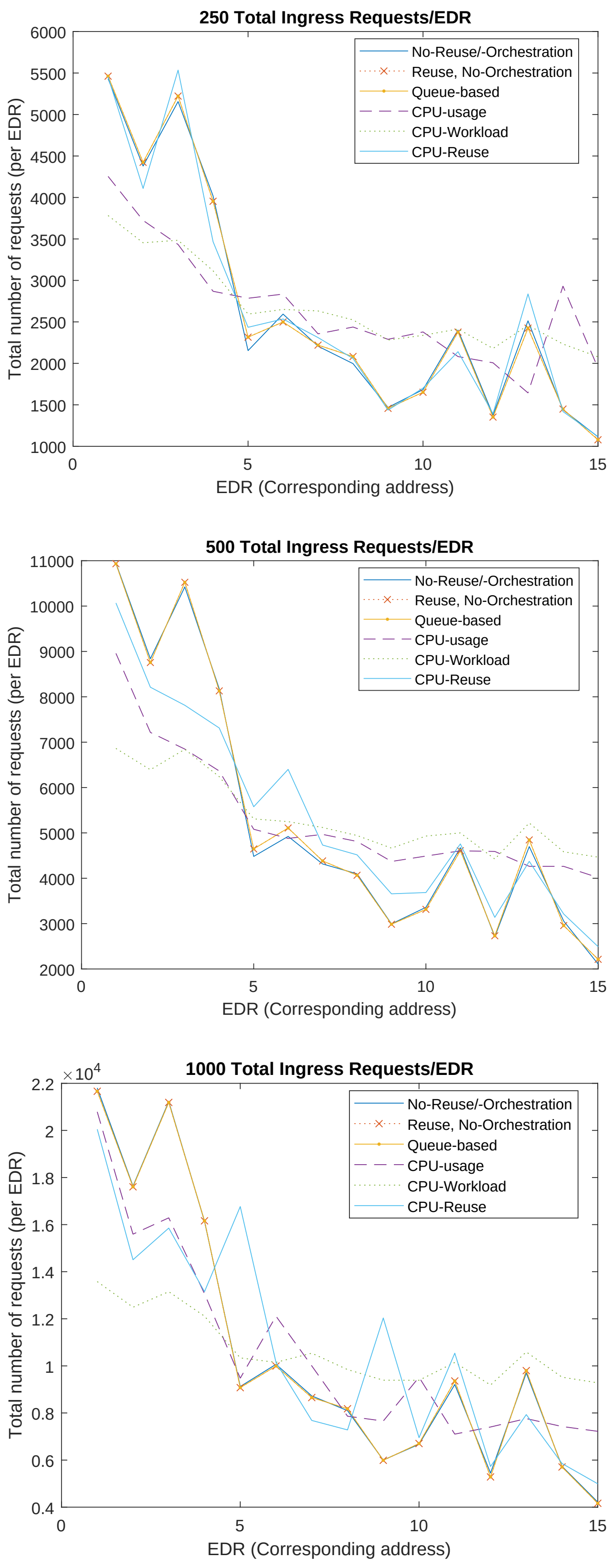
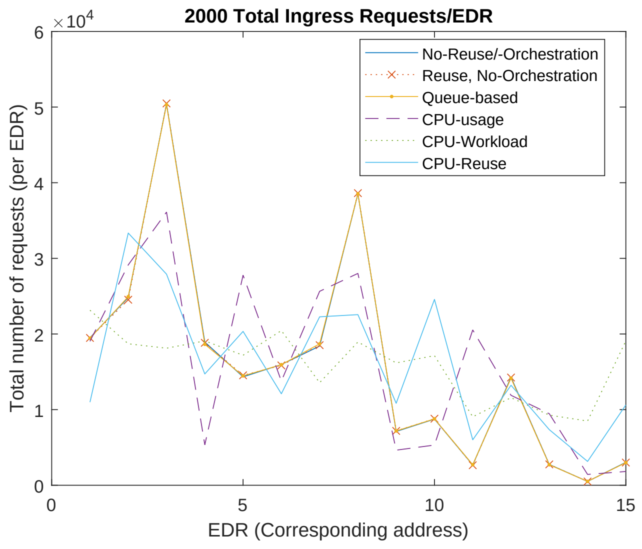
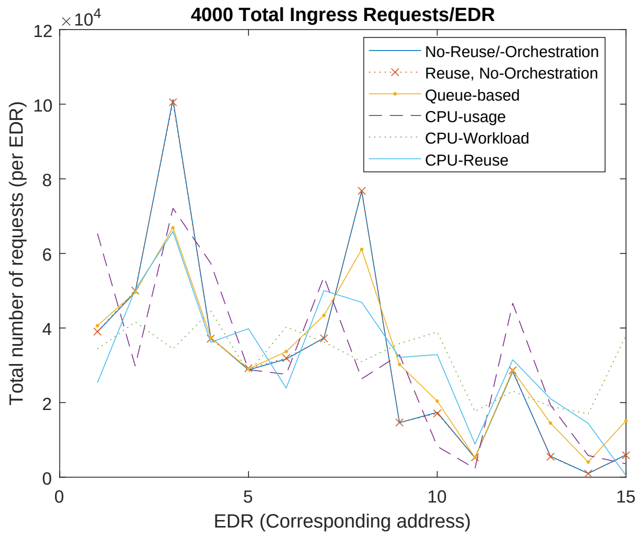


Figure 5.12: Workloads for different rates with MNIST Dataset.





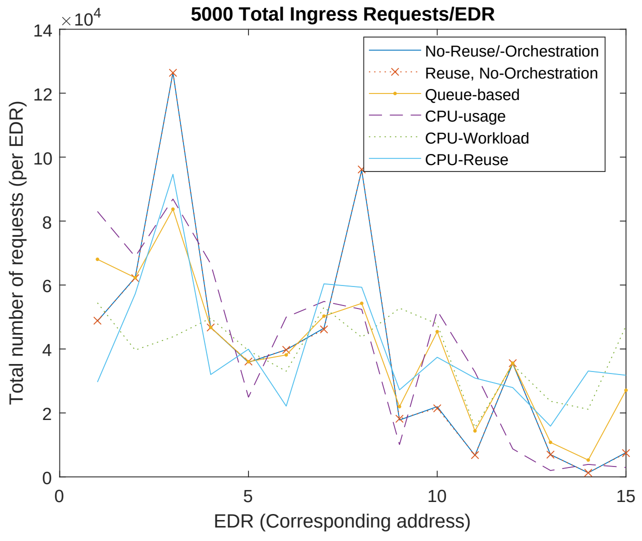
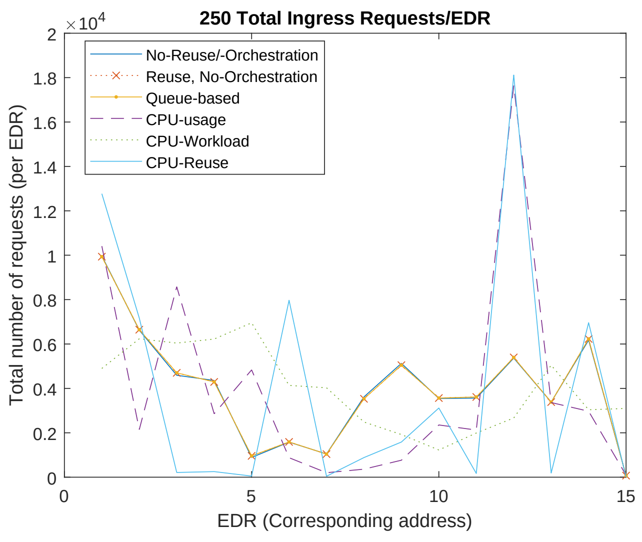
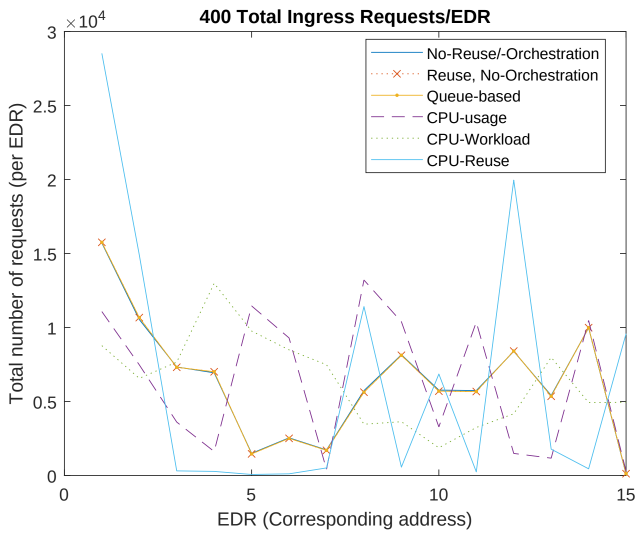


Figure 5.13: Workloads for different rates with Traffic Detection Dataset.





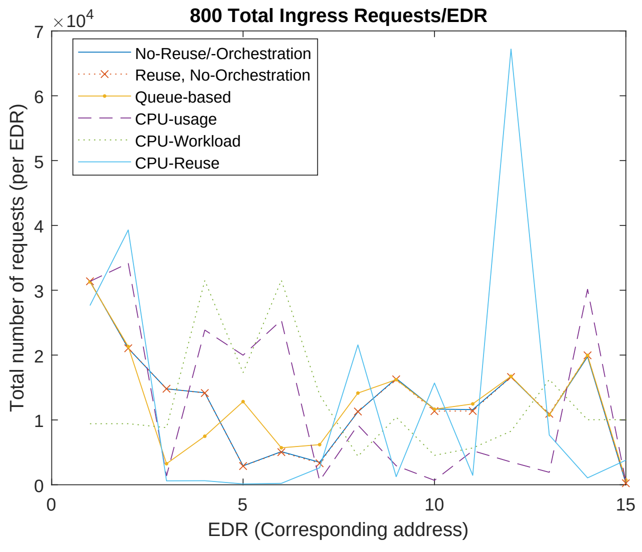


Figure 5.14: Workloads for different rates with Alexa Dataset.

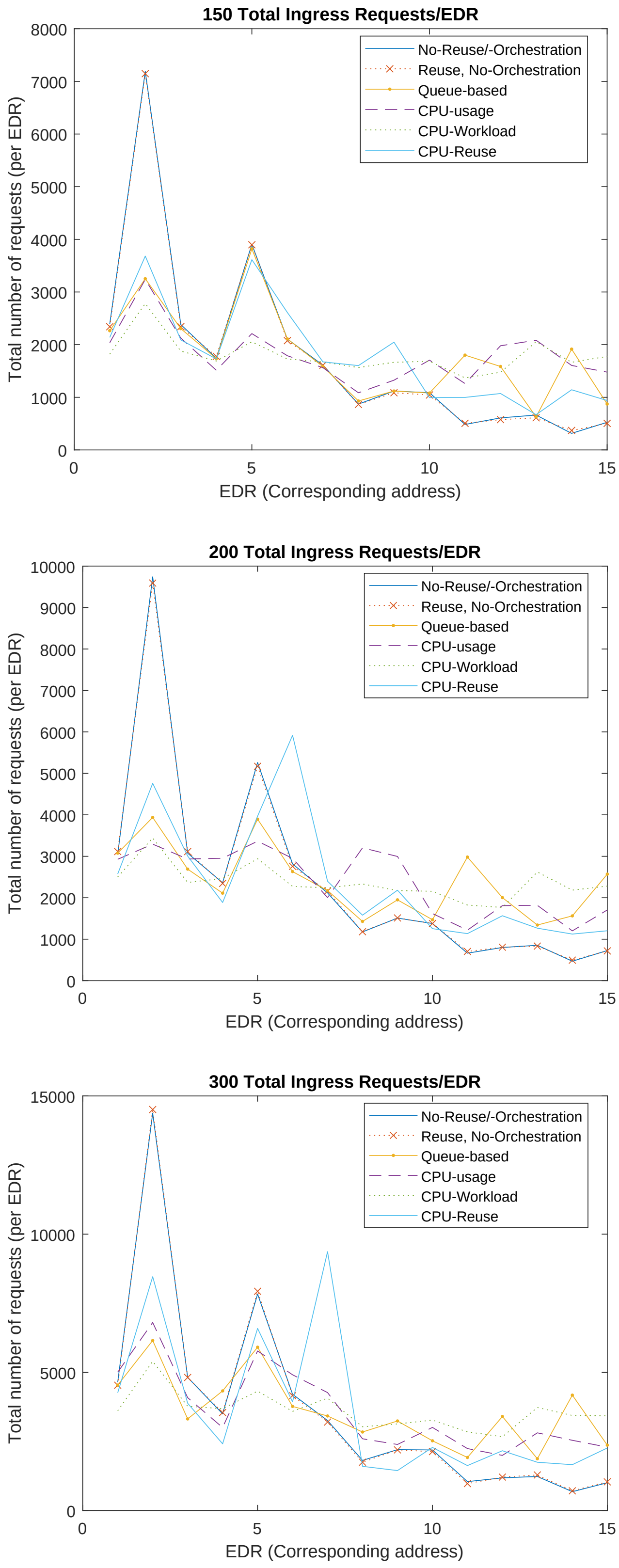


Figure 5.15: Workloads for different rates with General Commands Dataset.

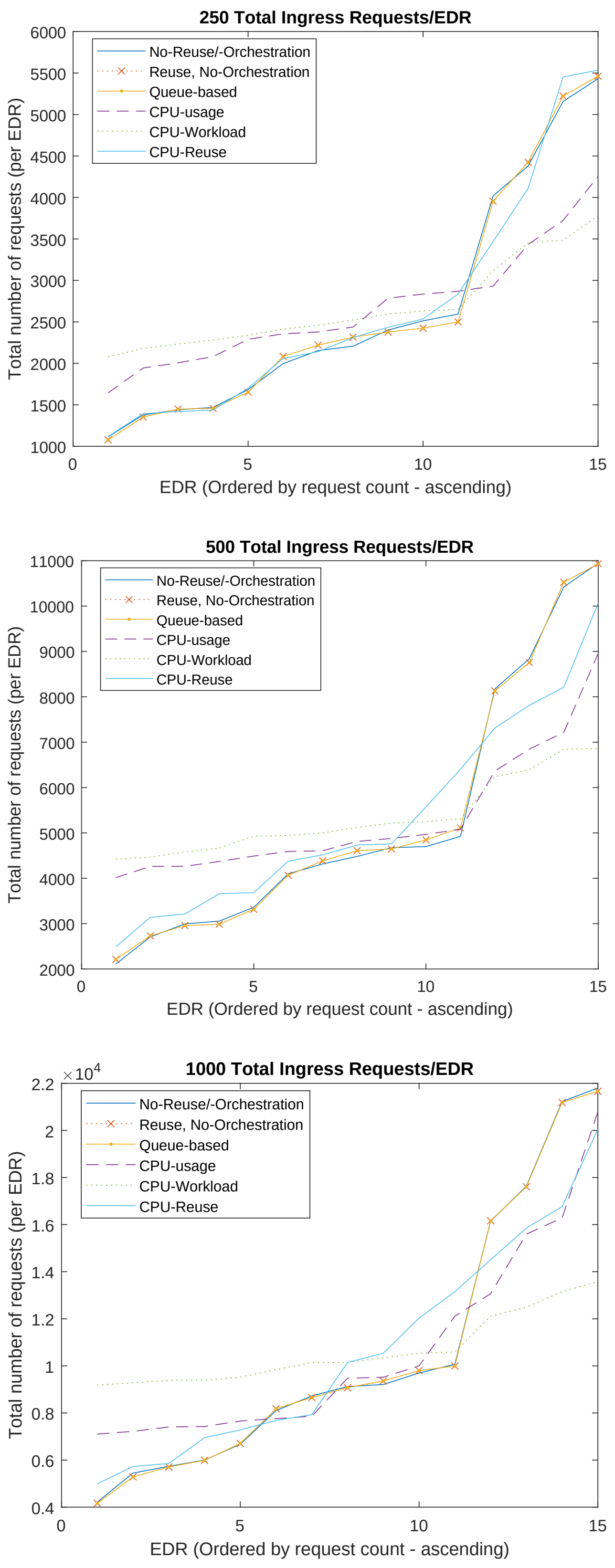


Figure 5.16: CPU utilisation for different rates with MNIST Dataset.

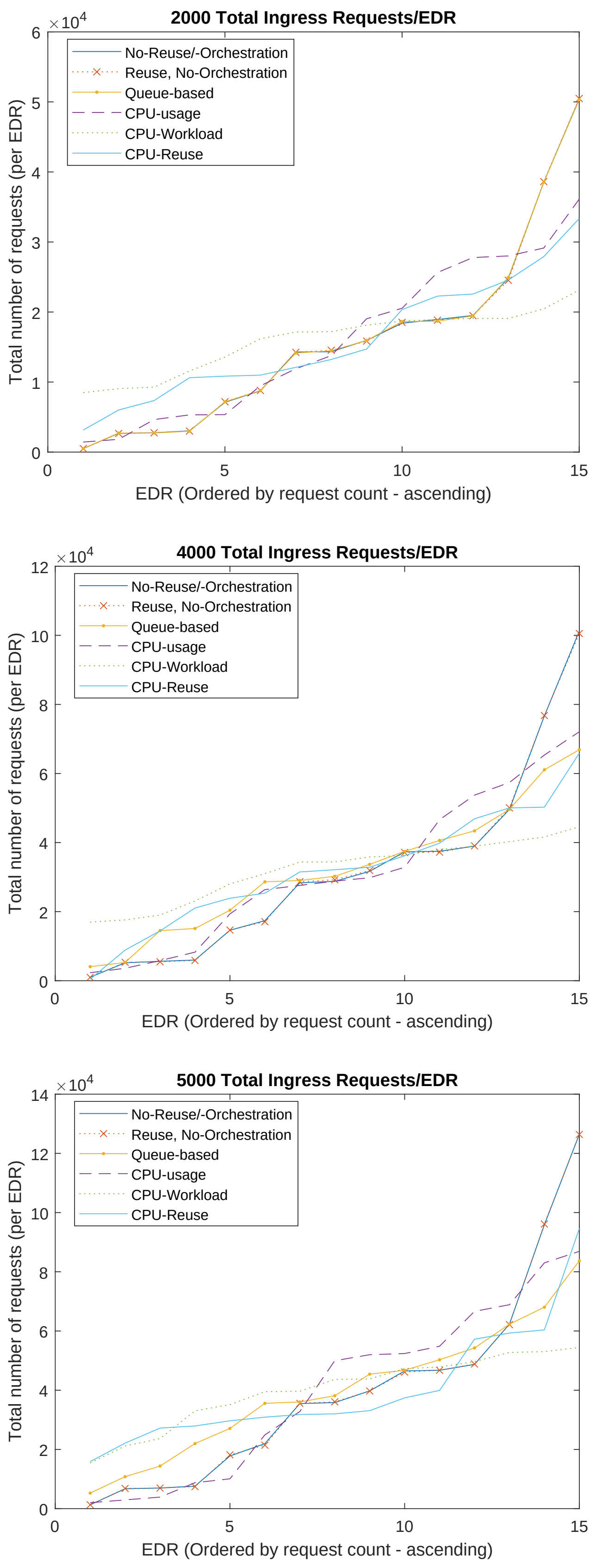


Figure 5.17: CPU utilisation for different rates with Traffic Detection Dataset.

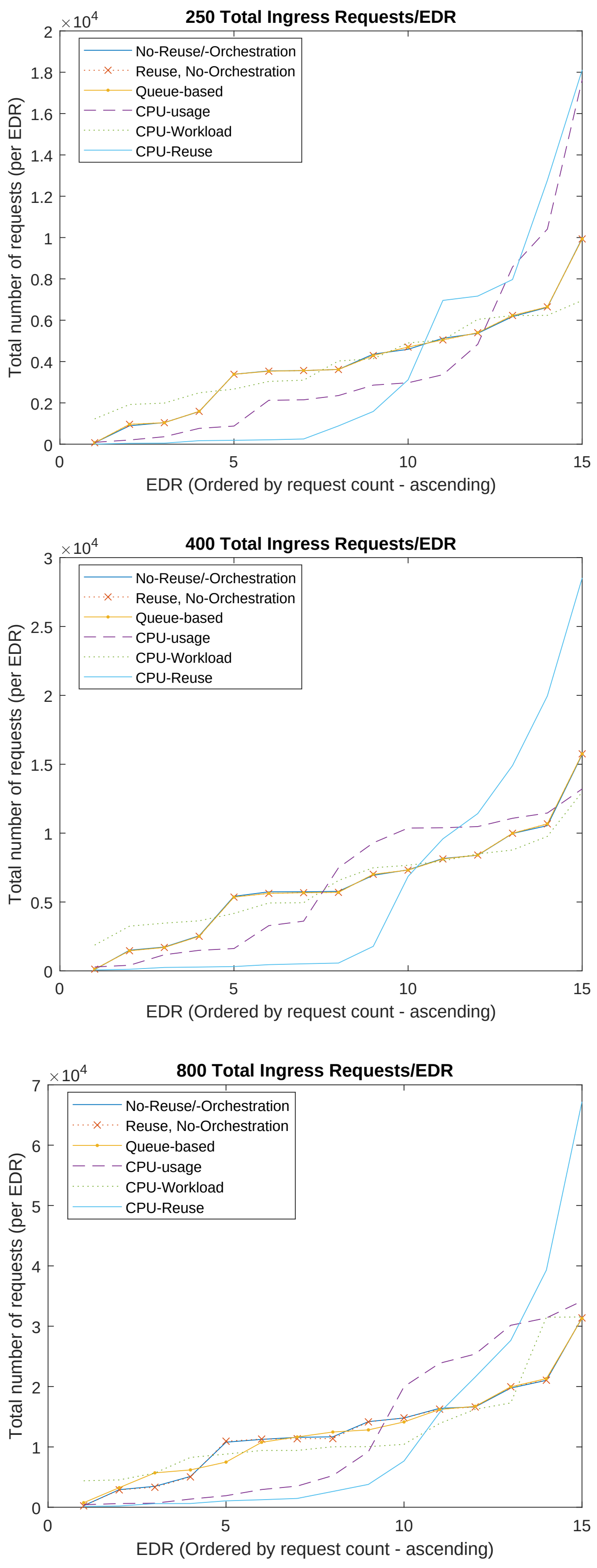


Figure 5.18: CPU utilisation for different rates with Alexa Dataset.

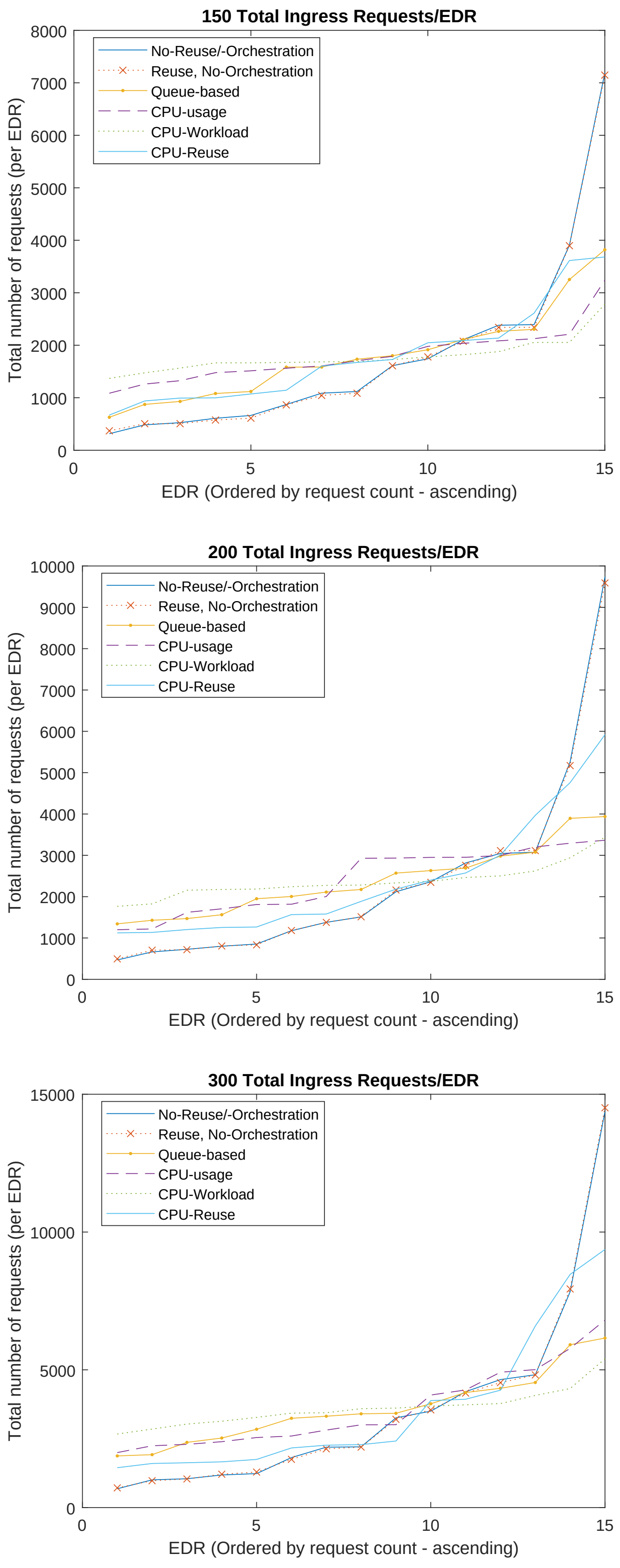


Figure 5.19: CPU utilisation for different rates with General Commands Dataset.

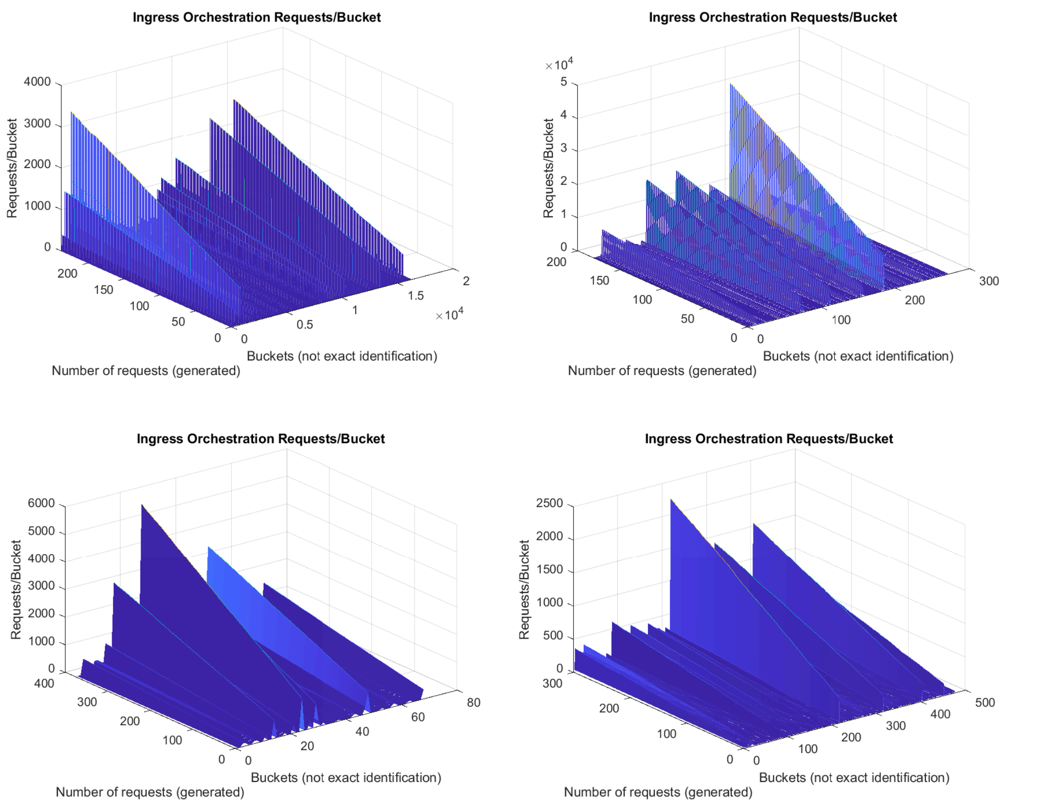


Figure 5.20: Bucket-specific workloads, for the MNIST, Traffic, Alexa and General Commands datasets, respectively. (left to right, top to bottom)

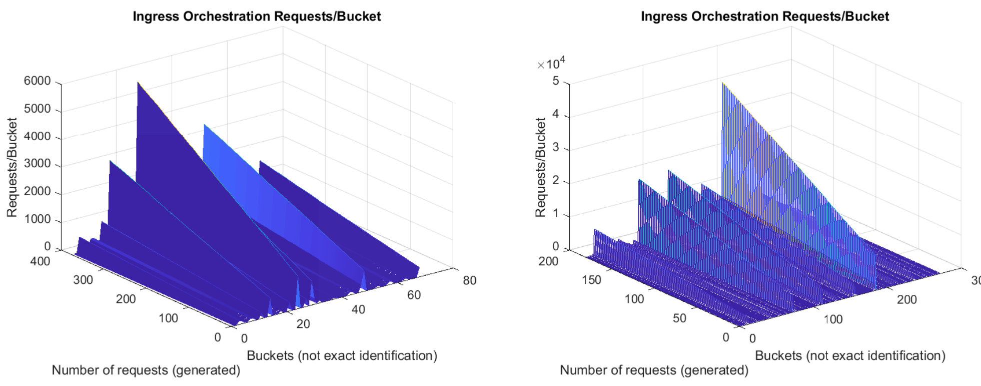


Figure 5.21: Bucket-specific workloads, for the Alexa and Traffic datasets, respectively.

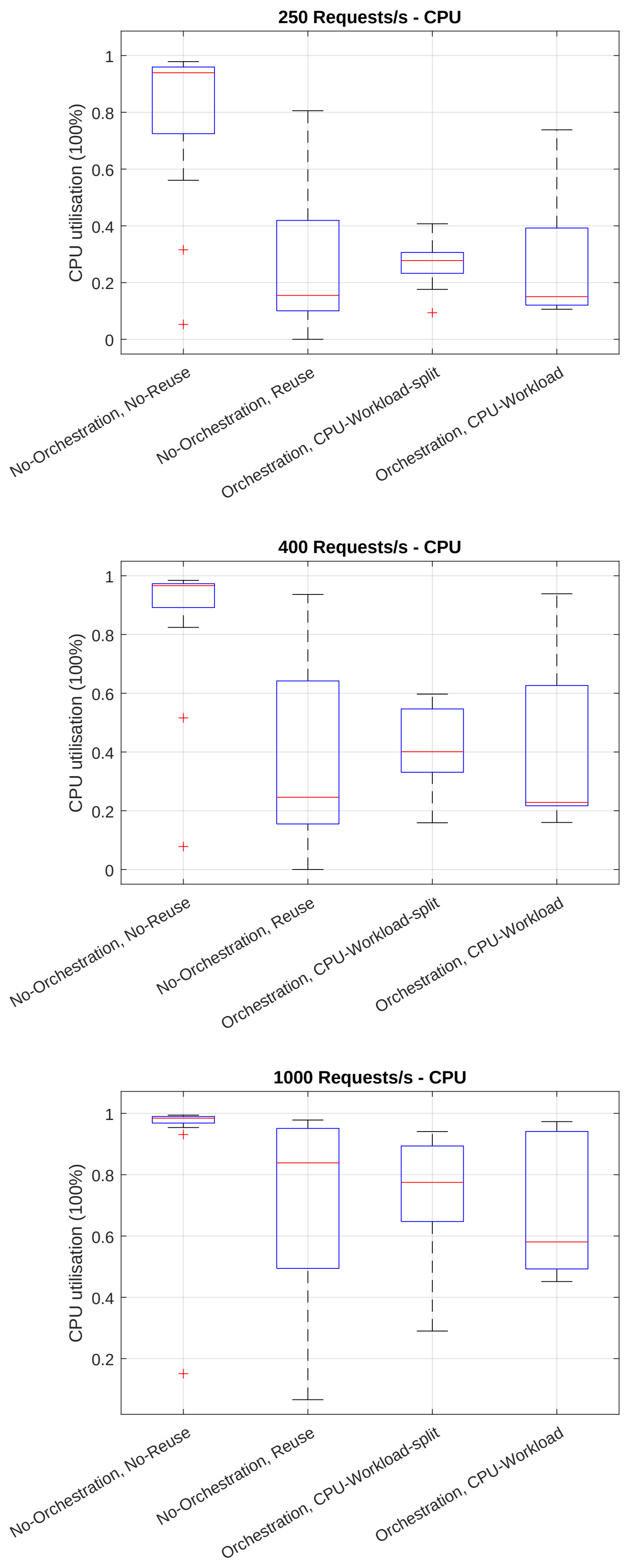


Figure 5.22: CPU utilisation for Alexa Dataset.

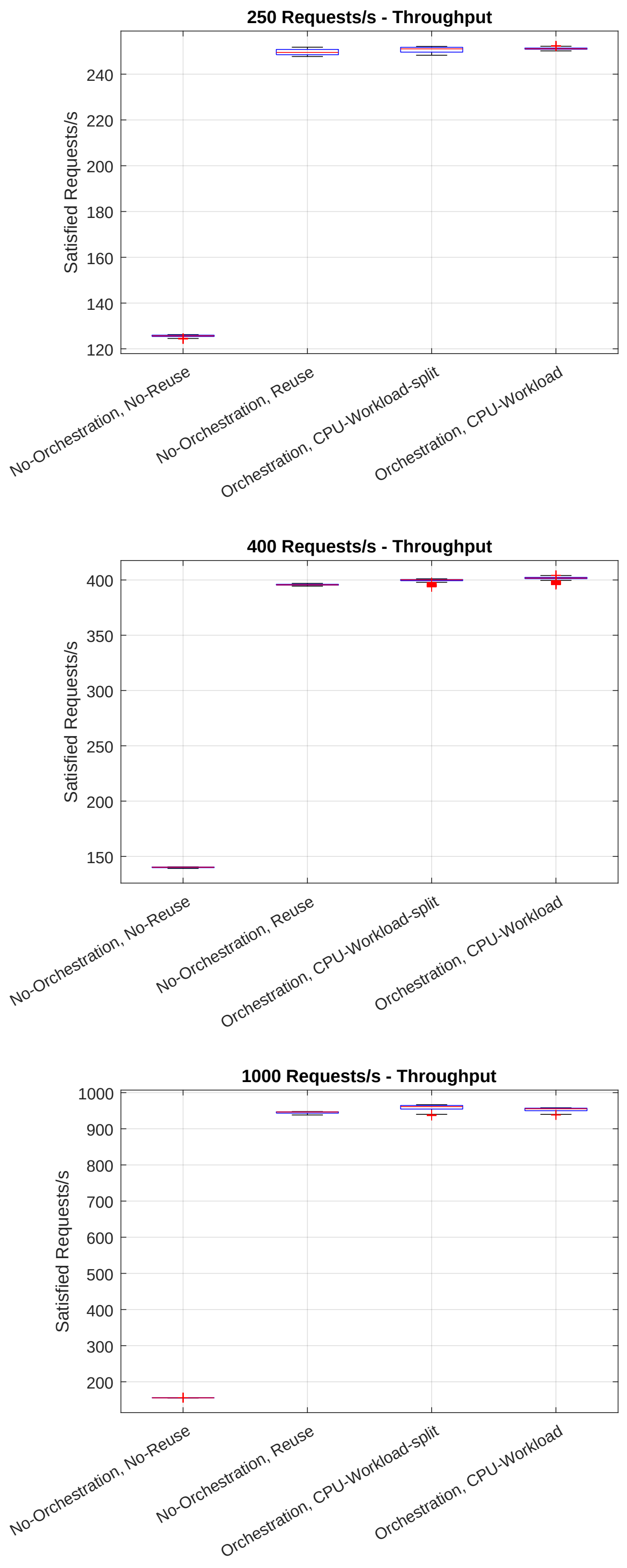


Figure 5.23: Throughput for Alexa Dataset.

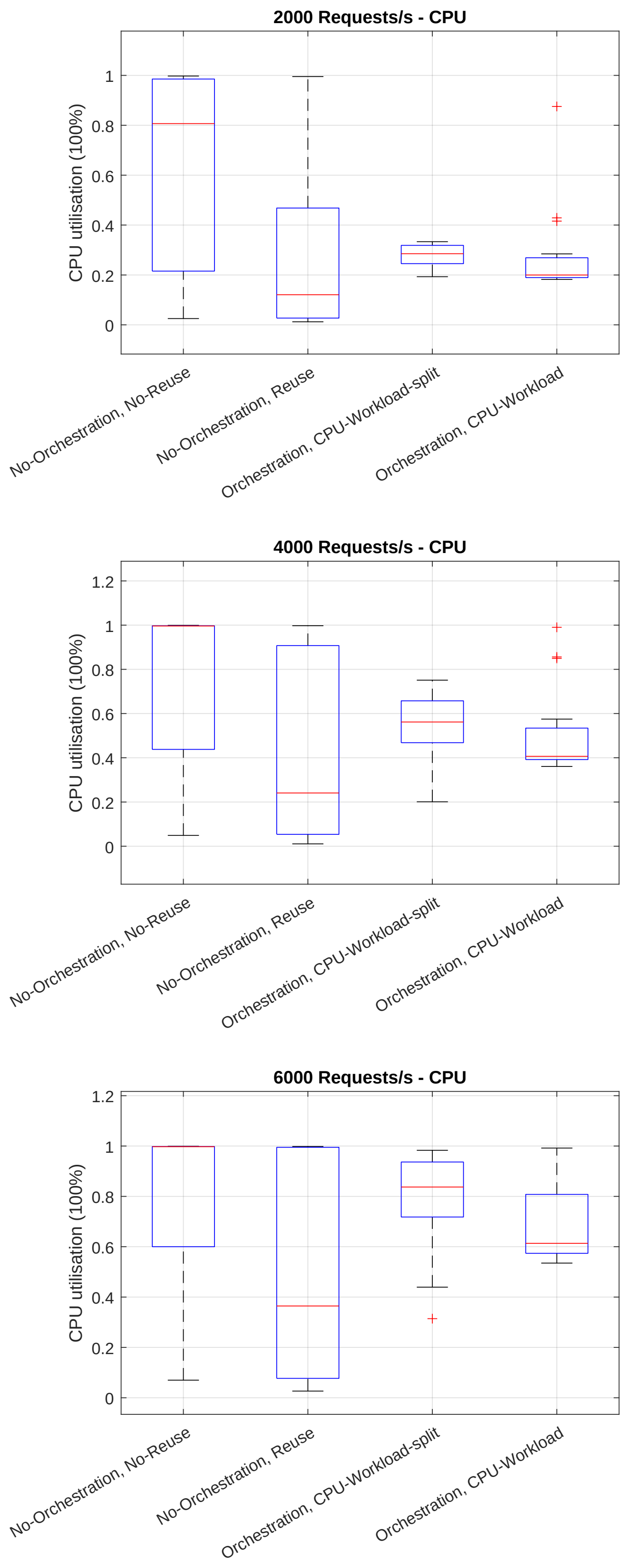


Figure 5.24: CPU utilisation for Traffic Dataset.

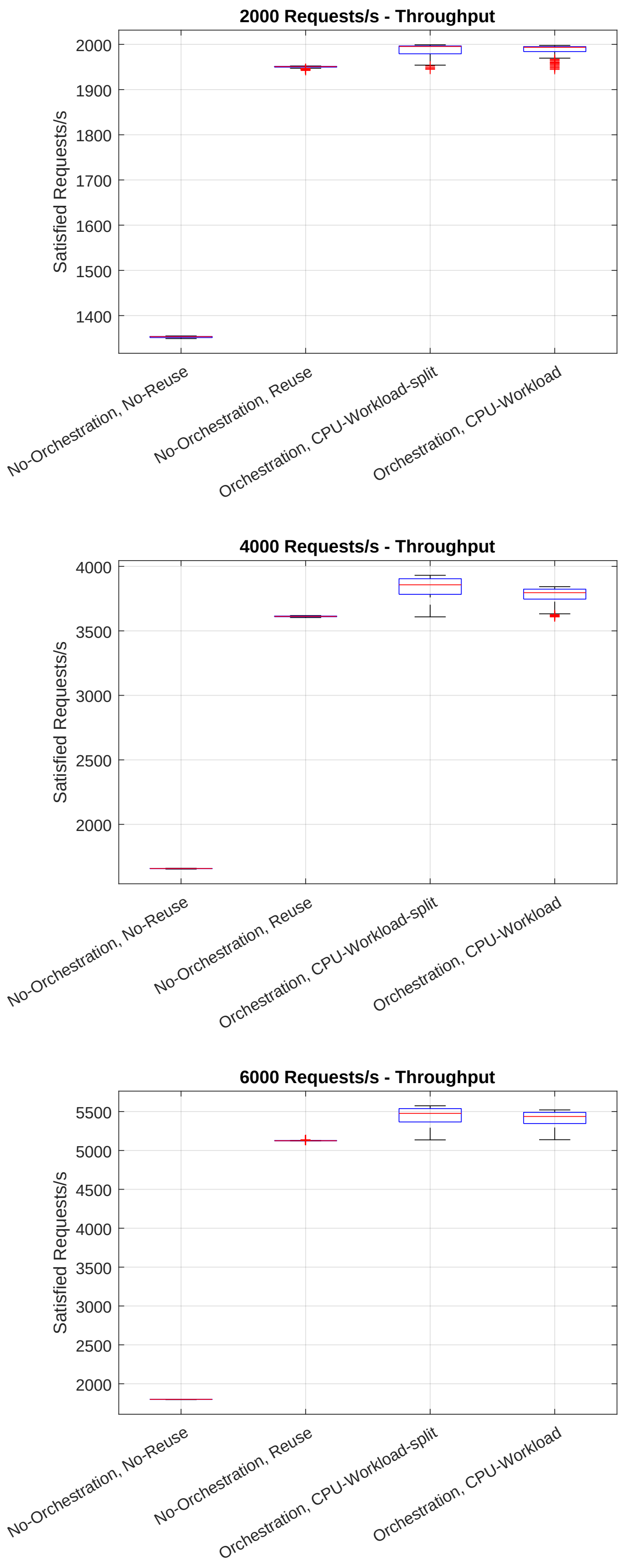


Figure 5.25: Throughput for Traffic Dataset.

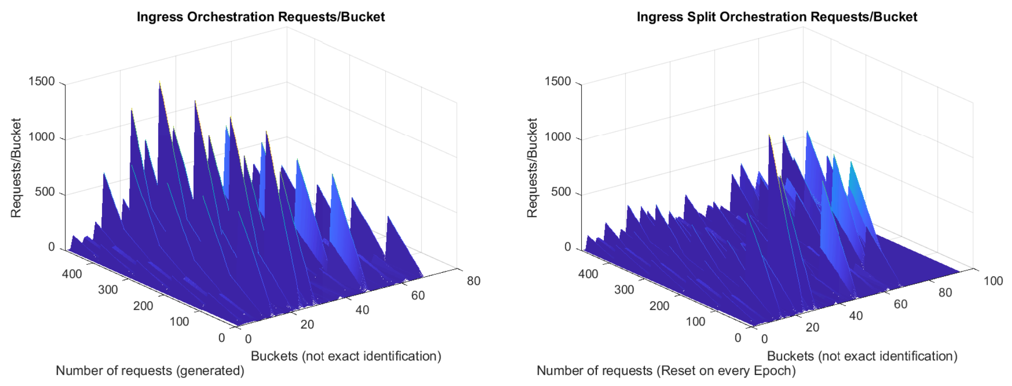


Figure 5.26: Bucket-specific workloads, for the CPU-Workload and CPU-Workload-Split cases of the Alexa dataset, respectively.

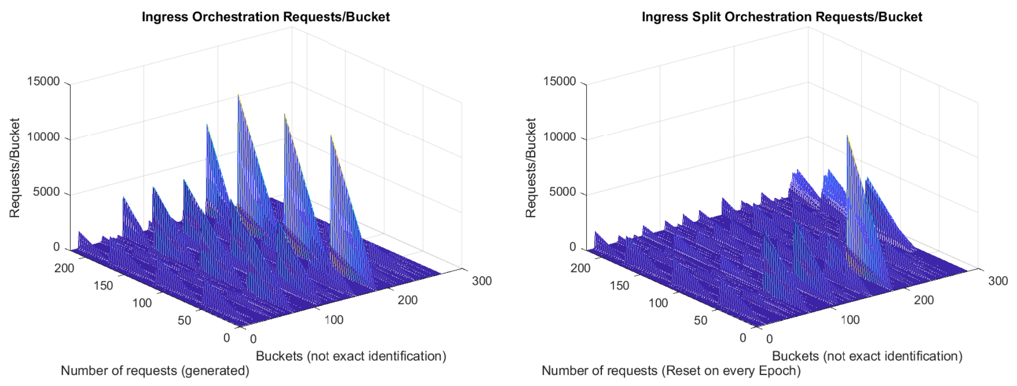


Figure 5.27: Bucket-specific workloads, for the CPU-Workload and CPU-Workload-Split cases of the Traffic dataset, respectively.

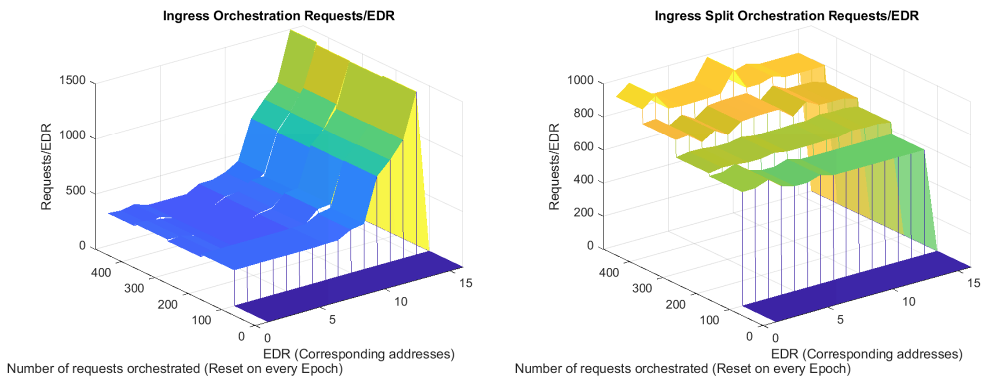


Figure 5.28: EDR-specific orchestrated workloads, without and with bucket splitting, for the Alexa dataset, respectively.

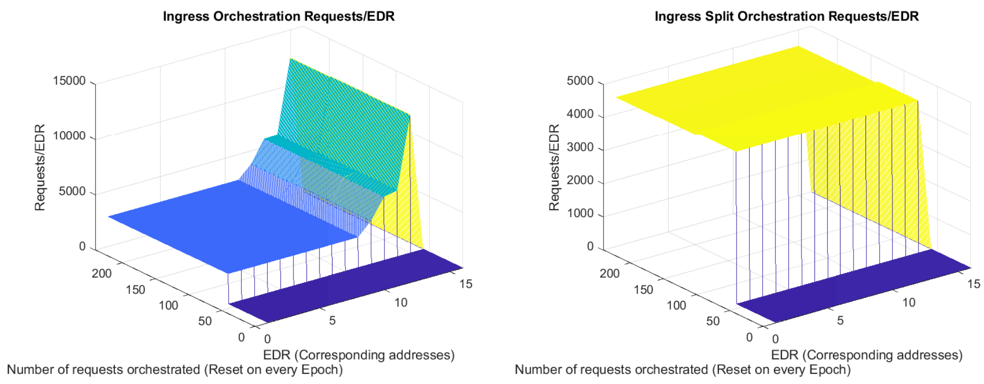


Figure 5.29: EDR-specific orchestrated workloads, without and with bucket splitting, for the Traffic dataset, respectively.

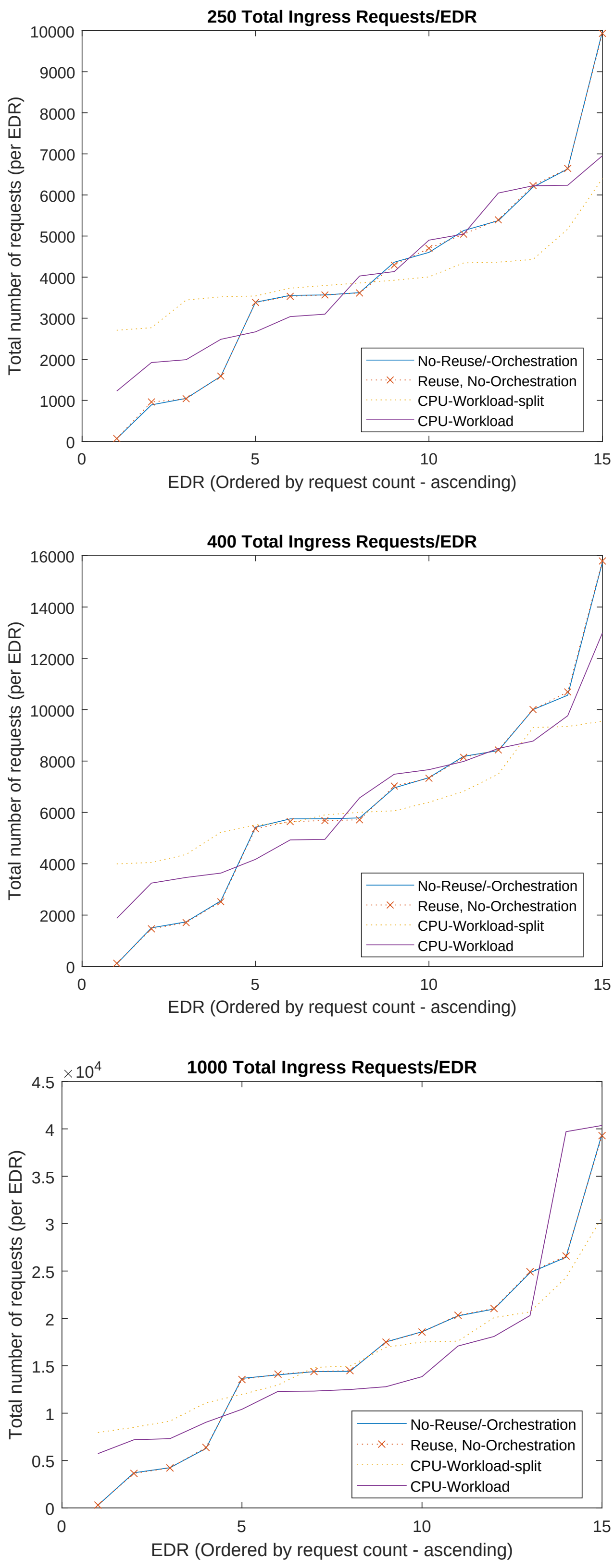


Figure 5.30: Increasing-workload-ordered EDR Workloads for different workload cases, for the Alexa Dataset.

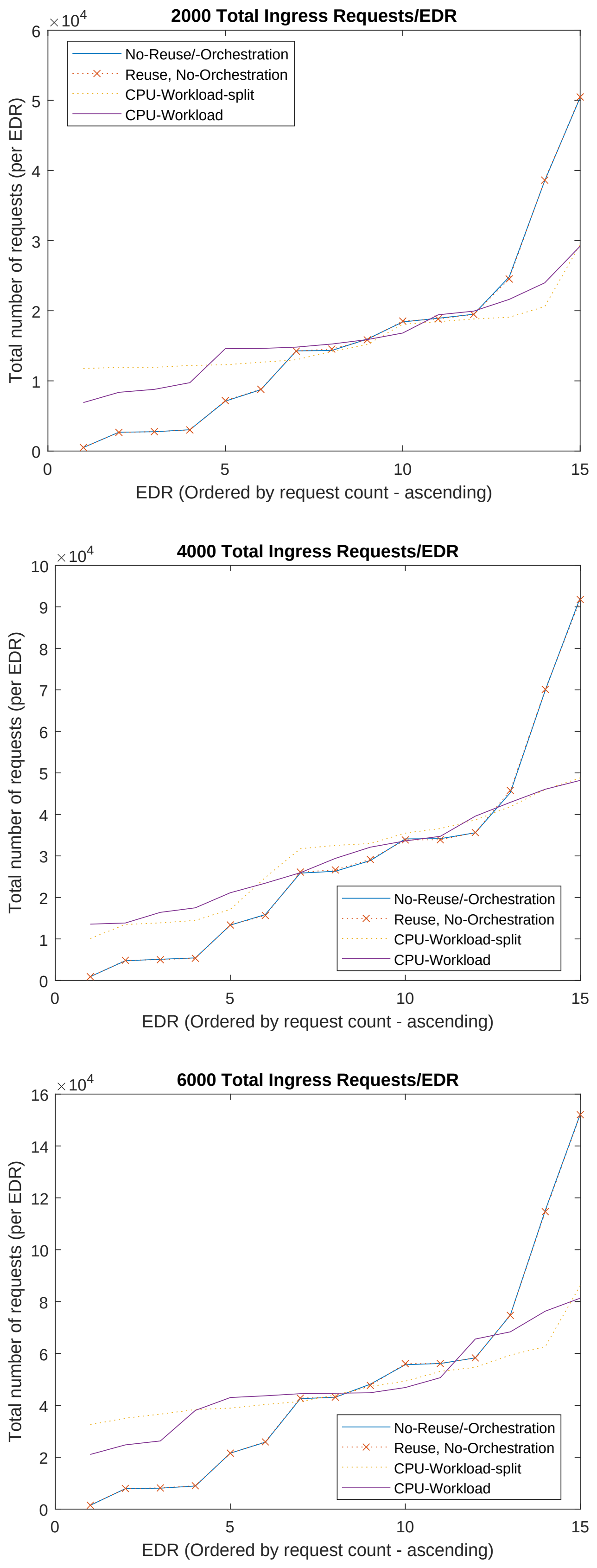


Figure 5.31: Increasing-workload-ordered EDR Workloads for different workload cases, for the Traffic Dataset.

Chapter 6

Discussion on Potential Next Steps and Conclusions

The development of this system, by working on different pieces, and opening up even more possibilities for research into edge computing systems, provides an insight into the progress made throughout the PhD, and shows the research done to date. The work done during the PhD, on researching and developing the presented system, has the main purpose of demonstrating the potential for and partially achieving intelligence at the edge, with the help of both computing and storage (demonstrated even through some citations of the presented published materials). The results of the research within the funding period and beyond have demonstrated good potential for the system's implementation and high impact towards the research domain of networked computing (edge and cloud alike).

6.1 Conclusion of PhD Outcomes

The Introduction and Literature review chapters provide a background of the ideas developed for the implementation of Edge processing and storage systems, by putting the system in its place,with regards to the current technological setting.

Using this knowledge, existing and proposed research questions within the Edge networks computing management domain, a hypothesis and a potential system structure were formed, defined as the ERP system. From this, a hierarchical structure and a network background is given to the system.

We first took a look at the following research question:

What are the improvements and impact that an efficient and more intelligent edge networked storage system could have on the QoE and performance provided through edge networks in terms of capital investments and operational costs offered and needed by Edge networks by/to providers and/or (their) users?
in order to then approach some of the main challenges that it imposes and implies:

- User mobility, upload bottlenecks, created by edge-based devices and users accessing cloud-based services, and data availability and validity timings were approached as the first challenge, with use cases and real-world traces, to provide realistic connectivity, Edge processing and storage to Edge nodes, Edge-based users and Data Providers. This first challenge was addressed through the use-case-led design and evaluation of the EDR environments in realistic scenarios and to provide a good insight into the simple, Edge-based deployment feasibility of EDRs, to provide connectivity, Edge processing and storage, as presented in the first paper.

- At the point of the second paper, as a solution for the problems that information-centricity in networked computing environments was still facing, for data storage efficiency, and keeping the timeliness of data utilisation in mind, together with its distribution, SEND was designed. To aid data placement and management decisions, SEND relied, among other attributes, on system-wide identifiers of the data context, called labels. The experimental results demonstrated that SEND achieves data insertion times of 0.06ms-0.9ms, data lookup times of 0.5ms-5.3ms, and on-time completion of up to 92% of user requests for the retrieval of raw and processed data. With these results, comparing to the previous solution, of the uncoordniated processing system to which the SEND system is compared, an improvement of QoS performance of approximately 40-60% can be observed.

- A few other problems were then tackled, as well: information efficiency, processing and storage efficiency, function and data distribution efficiency and service delivery mechanisms. These problems were approached and (at least partially) solved in ReStorEdge. With this, information retention, delivery efficiency improvement and the environment's communication and computation efficiency improvement were studied, by integrating the concept of Locality-Sensitive Hashing and computation reuse with the system, and implementing a hybrid, Hash-and-MetaData routing optimisation system, to preserve both storage and network resources. To achieve this system, we implemented a similarity-based data classification system, which we evaluated with real-world datasets of images and smart-assistant speech measurements, and with realistic processing functions and timings. We further performed a network simulation study to evaluate different orchestration strategies based on real-world datasets, evaluating the performance and trade-offs of the design as a whole.

To summarise, the first steps in research and development for this system were first presented, as the local resource management and service feedback algorithm and the (mainly) label-based SEND data storage management system. The main output of this PhD would be the development of several Edge-based store-process-send principles and modular parts of a system, that could be used in different ways, within different scenarios, but complementing each other very well within EDR Environments. The EDR-based system and network architecture themselves are the main proposed output ideas of the PhD, motivating the completion of the research necessary for the full implementation and proof of viability of these systems, for the modular concepts to constitute a whole.

As conclusions from the resulting publications of this thesis, three points have to be made here (by looking at citations, reviews done and publication of work):

1. The fact that the publicaitons were cited in relevant fields, in the intended way, within appropriate research approaches and directions and as envisaged both in the future work and discussions demonstrates that the efficient communication of newly developed research has been learned and achieved;

2. Because of the above, new standards and new networked systems considered

with much more ease, with the necessary guidance;

3. Finally, the research direction of intelligent edge networked computing was and will be influenced by the work achieved within this PhD, considering that the new research and connections made within the period of study have been useful to a few researchers, other than the author.

6.2 New Research Directions

A study on Edge Data Computing, Storage and Validity - Several ideas/research directions resulted from the first publication within this PhD (Chapter 3), and one of them was approached in Chapter 4, by approaching the problem of multi-application storage placement, towards efficient processing and timely data placement/utilisation. One other resulting research direction is more open-ended, and should deal with the processing part of the data flows, the validity of data for certain cases/types of functions and allocated times. An interesting relationship between the freshness period and the shelf lives of data was discovered, and could potentially be exploited by the network, in order to obtain better performance, both in QoS and system resource utilisation.

6.2.1 Next steps in architecture definition and design

The Scalability of EDR Environments - From Chapter 4, another few interesting and potentially very impactful research directions were uncovered. Of course, the idea that certain data and functions could be of common interest at different times to an application and could be **reused** was approached, exploited and different orchestrations, in the same spirit as in Chapter 4, were approached in Chapter 5. However, With data stored in one environment and potential use for it in so many other ways, in so many different other places on the Internet, there is great potential in the development of inter-EDR-domain collaboration and data sharing mechanics and systems, based on secure, privacy-preserving and financially-beneficial systems. From this point on, quite a few other opportunities arise, as from this point, we are dealing with systems development.

The potential of Cross-application, Session-less Storage, Computation and Reuse - As single-application reuse is approached in Chapter 5, one of the main, systems-related topics, that results from this chapter is the investigation of how different applications, from the same contextual domains/of the same types (map directions, traffic data, public transport/voice commands, news, ads or video, CCTV, VR, AR, photos) could work together, towards (several) different goals, but, because of previous developments, this could also be easily done in a privacy-preserving, secure and application-aware manner, so that no application-, user- **or** repository-provider-impsed policies are violated.

As a conclusion on these new research directions, and looking at the progress made in the development of the Internet throughout this PhD, it is clear to see that the Internet should mainly be developed in an information-centric manner, due to the new development possibilities that this approach could offer.

Bibliography

[1] Weisong Shi, Jie Cao, Quan Zhang, Youhuizi Li, and Lanyu Xu. Edge computing: Vision and challenges. IEEE internet of things journal, 3(5):637–646, 2016.

[2] Ioannis Psaras, Onur Ascigil, Sergi Rene, George Pavlou, Alex Afanasyev, and Lixia Zhang. Mobile data repositories at the edge. In {USENIX} Workshop on Hot Topics in Edge Computing (HotEdge 18), 2018.

[3] Mahadev Satyanarayanan. The emergence of edge computing. Computer, 50(1):30–39, 2017.

[4] Van Jacobson, Diana K. Smetters, James D. Thornton, Michael F. Plass, Nicholas H. Briggs, and Rebecca L. Braynard. Networking named content. In Proceedings of the 5th International Conference on Emerging Networking Experiments and Technologies, CoNEXT '09, page 1–12, New York, NY, USA, 2009. Association for Computing Machinery.

[5] Victor Souza, S. Rangarajan, R. Stadler, J. Carapinha, Bjorn Bjurling, Rebecca Steinert, P. Aranda, M. Keller, Daniel Turull, and Avi Miron. Future networks project 257448 " sail – scalable and adaptable internet solutions " d-5 . 3 (d . d . 2) description of implemented prototype. 2012.

[6] Lixia Zhang, Alexander Afanasyev, Jeffrey Burke, Van Jacobson, kc claffy, Patrick Crowley, Christos Papadopoulos, Lan Wang, and Beichuan Zhang. Named data networking. SIGCOMM Comput. Commun. Rev., 44(3):66–73, July 2014.

[7] Teemu Koponen, Mohit Chawla, Byung-Gon Chun, Andrey Ermolinskiy, Kye Hyun Kim, Scott Shenker, and Ion Stoica. A data-oriented (and beyond) network architecture. In Proceedings of the 2007 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications, SIGCOMM '07, page 181–192, New York, NY, USA, 2007. Association for Computing Machinery.

[8] Lorenzo Saino, Ioannis Psaras, and George Pavlou. Hash-routing schemes for information centric networking. In Proceedings of the 3rd ACM SIGCOMM Workshop on Information-Centric Networking, ICN '13, page 27–32, New York, NY, USA, 2013. Association for Computing Machinery.

[9] K. Cho, M. Lee, K. Park, T. T. Kwon, Y. Choi, and Sangheon Pack. Wave: Popularity-based and collaborative in-network caching for content-oriented networks. In 2012 Proceedings IEEE INFOCOM Workshops, pages 316–321, 2012.

[10] Chien-Chun Hung, Ganesh Ananthanarayanan, Peter Bodik, Leana Golubchik, Minlan Yu, Paramvir Bahl, and Matthai Philipose. Videoedge: Processing camera streams using hierarchical clusters. In 2018 IEEE/ACM Symposium on Edge Computing (SEC), pages 115–131. IEEE, 2018.

[11] Peizhen Guo and Wenjun Hu. Potluck: Cross-application approximate deduplication for computation-intensive mobile applications. In Proceedings of the Twenty-Third International Conference on Architectural Support for Programming Languages and Operating Systems, pages 271–284, 2018.

[12] Md Washik Al Azad and Spyridon Mastorakis. The promise and challenges of computation deduplication and reuse at the network edge. IEEE Wireless Communications, 2022.

[13] Athena Vakali and George Pallis. Content delivery networks: Status and trends. IEEE Internet Computing, 7(6):68–74, 2003.

[14] Kiryong Ha et al. Towards wearable cognitive assistance. Proceedings of the 12th annual international conference on Mobile systems, applications, and services - MobiSys '14, pages 68–81, 2014.

[15] Andrea Zanella et al. Internet of Things for Smart Cities. IEEE Internet of Things Journal, 1(1):22–32, 2014.

[16] Alex Alley. Investment firm, james hay partnership's website and phones down after data center outage, 2020.

[17] Cisco. Cisco Visual Networking Index : Forecast and Trends. 2018.

[18] Sebastian Moss. Equinix ld8 data center experiences major outage, 2018.

[19] Shanhe Yi, Cheng Li, and Qun Li. A Survey of Fog Computing. Proceedings of the 2015 Workshop on Mobile Big Data - Mobidata '15, pages 37–42, 2015.

[20] James F. Kurose and Keith W. Ross. Computer Networking: A Top-Down Approach (6th Edition). Pearson, 7th edition, 2017.

[21] H. Moustafa, E. M. Schooler, and J. McCarthy. Reverse cdn in fog computing: The lifecycle of video data in connected and autonomous vehicles. In 2017 IEEE Fog World Congress (FWC), pages 1–5, Oct 2017.

[22] Dianlei Xu, Tong Li, Yong Li, Xiang Su, Sasu Tarkoma, Tao Jiang, Jon Crowcroft, and Pan Hui. Edge intelligence: Architectures, challenges, and applications. arXiv preprint arXiv:2003.12172, 2020.

[23] Ioannis Psaras et al. Mobile data repositories at the edge. In USENIX Workshop on Hot Topics in Edge Computing (HotEdge 18), Boston, MA, 2018. USENIX Association.

[24] Abdul Hannan, Sobia Arshad, Muhammad Awais Azam, Jonathan Loo, Syed Hassan Ahmed, Muhammad Faran Majeed, and Sayed Chhattan Shah.

Disaster management system aided by named data network of things: Architecture, design, and analysis. Sensors, 18(8), 2018.

[25] M. Amadeo, C. Campolo, and A. Molinaro. Internet of things via named data networking: The support of push traffic. In 2014 International Conference and Workshop on the Network of the Future (NOF), pages 1–5, Dec 2014.

[26] Muhammad Faran Majeed, Syed Hassan Ahmed, and Matthew N Dailey. Enabling Push-Based Critical Data Forwarding in Vehicular Named Data Networks. IEEE Communications Letters, 21(4):873–876, 2017.

[27] Onur Ascigil et al. A keyword-based ICN-IoT platform. Proceedings of the 4th ACM Conference on Information-Centric Networking - ICN '17, (September):22–28, 2017.

[28] Tao Ouyang, Zhi Zhou, and Xu Chen. Follow me at the edge: Mobility-aware dynamic service placement for mobile edge computing. IEEE Journal on Selected Areas in Communications, 36(10), 2018.

[29] Ting He, Hana Khamfroush, Shiqiang Wang, Tom La Porta, and Sebastian Stein. It's hard to share: joint service placement and request scheduling in edge clouds with sharable and non-sharable resources. In 2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS). IEEE, 2018.

[30] A. C. Baktir, A. Ozgovde, and C. Ersoy. Enabling service-centric networks for cloudlets using sdn. In 2017 IFIP/IEEE Symposium on Integrated Network and Service Management (IM), pages 344–352, 2017.

[31] Sripriya Adhatarao, Mayutan Arumaithurai, Dirk Kutscher, and Xiaoming Fu. NeMoI: Network mobility in ICN. Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), 11227 LNCS:220–244, 2019.

[32] A. Mtibaa, R. Tourani, S. Misra, J. Burke, and L. Zhang. Towards edge computing over named data networking. In 2018 IEEE International Conference on Edge Computing (EDGE), pages 117–120, July 2018.

[33] Michał Król and Ioannis Psaras. NFaaS. Proceedings of the 4th ACM Conference on Information-Centric Networking - ICN '17, 11:134–144, 2017.

[34] Michal Krol et al. Computation offloading with ICN. Proceedings of the 5th ACM Conference on Information-Centric Networking - ICN '18, 2018.

[35] MichałKról and Ioannis Psaras. Nfaas: Named function as a service. In Proceedings of the 4th ACM Conference on Information-Centric Networking, ICN '17, pages 134–144, New York, NY, USA, 2017. ACM.

[36] Onur Ascigil et al. On uncoordinated service placement in edge-clouds. In 2017 IEEE International Conference on Cloud Computing Technology and Science (CloudCom), pages 41–48. IEEE, 2017.

[37] Yuhang Ye et al. PIoT: Programmable IoT using Information Centric Networking. Proceedings of the IEEE/IFIP Network Operations and Management Symposium (NOMS), (Noms):825–829, 2016.

[38] Eve M. Schooler et al. An Architectural Vision for a Data-Centric IoT: Rethinking Things, Trust and Clouds. Proceedings - International Conference on Distributed Computing Systems, pages 1717–1728, 2017.

[39] I. Psaras, W. K. Chai, and G. Pavlou. In-network cache management and resource allocation for information-centric networks. IEEE Transactions on Parallel and Distributed Systems, 25(11):2920–2931, Nov 2014.

[40] Utsav Drolia, Nathan D. Mickulicz, Rajeev Gandhi, and Priya Narasimhan. Krowd: A key-value store for crowded venues. In MobiArch, 2015.

[41] Dave Hitz, James Lau, and Michael A Malcolm. File system design for an nfs file server appliance. In USENIX winter, volume 94, 1994.

[42] James J Kistler and Mahadev Satyanarayanan. Disconnected operation in the coda file system. ACM Transactions on Computer Systems (TOCS), 1992.

[43] Konstantin Shvachko, Hairong Kuang, Sanjay Radia, and Robert Chansler. The hadoop distributed file system. In 2010 IEEE 26th symposium on mass storage systems and technologies (MSST), pages 1–10. Ieee, 2010.

[44] Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung. The google file system. In Proceedings of the nineteenth ACM symposium on Operating systems principles, pages 29–43, 2003.

[45] Renzo Angles and Claudio Gutierrez. Survey of graph database models. ACM Computing Surveys (CSUR), 40(1):1–39, 2008.

[46] Xin Jin et al. Netcache: Balancing key-value stores with fast in-network caching. In Proceedings of the 26th Symposium on Operating Systems Principles, pages 121–136, 2017.

[47] Michaela Blott, Kimon Karras, Ling Liu, Kees Vissers, Jeremia Bär, and Zsolt István. Achieving 10gbps line-rate key-value stores with fpgas. In Presented as part of the 5th {USENIX} Workshop on Hot Topics in Cloud Computing, 2013.

[48] Yunqi Ye, Liangliang Xiao, I-Ling Yen, and Farokh Bastani. Secure, dependable, and high performance cloud storage. In 2010 29th IEEE Symposium on Reliable Distributed Systems, pages 194–203. IEEE, 2010.

[49] Emil Stefanov and Elaine Shi. Oblivistore: High performance oblivious cloud storage. In 2013 IEEE Symposium on Security and Privacy, pages 253–267. IEEE, 2013.

[50] Bo Mao, Suzhen Wu, and Hong Jiang. Improving storage availability in cloud-of-clouds with hybrid redundant data distribution. In 2015 IEEE International Parallel and Distributed Processing Symposium, pages 633–642, 2015.

[51] Jibin Wang et al. I-sieve: An inline high performance deduplication system used in cloud storage. Tsinghua Science and Technology, 2015.

[52] Danny Harnik, Benny Pinkas, and Alexandra Shulman-Peleg. Side channels in cloud services: Deduplication in cloud storage. IEEE Security & Privacy, 8(6):40–47, 2010.

[53] Cong Wang, Sherman SM Chow, Qian Wang, Kui Ren, and Wenjing Lou. Privacy-preserving public auditing for secure cloud storage. IEEE transactions on computers, 62(2):362–375, 2011.

[54] Seny Kamara and Kristin Lauter. Cryptographic cloud storage. In International Conference on Financial Cryptography and Data Security, pages 136–149. Springer, 2010.

[55] Qin Liu, Guojun Wang, and Jie Wu. Secure and privacy preserving keyword searching for cloud storage services. Journal of network and computer applications, 35(3):927–933, 2012.

[56] Jose M Alcaraz Calero, Nigel Edwards, Johannes Kirschnick, Lawrence Wilcock, and Mike Wray. Toward a multi-tenancy authorization system for cloud services. IEEE Security & Privacy, 8(6):48–55, 2010.

[57] Qingsong Wei et al. CDRM: a cost-effective dynamic replication management scheme for cloud storage cluster. In 2010 IEEE international conference on cluster computing, pages 188–196. IEEE, 2010.

[58] Anand Prahlad et al. Cloud gateway system for managing data storage to cloud storage sites, December 30 2010. US Patent App. 12/751,953.

[59] Adrian-Cristian Nicolaescu, Onur Ascigil, and Ioannis Psaras. Edge data repositories - the design of a store-process-send system at the edge. In Proceedings of the 1st ACM CoNEXT Workshop on Emerging In-Network Computing Paradigms, ENCP '19, page 41–47, New York, NY, USA, 2019. Association for Computing Machinery.

[60] Jianshen Liu, Matthew Leon Curry, Carlos Maltzahn, and Philip Kufeldt. Scale-out edge storage systems with embedded storage nodes to get better availability and cost-efficiency at the same time. In 3rd {USENIX} Workshop on Hot Topics in Edge Computing (HotEdge 20), 2020.

[61] Marisol García-Valls, Abhishek Dubey, and Vicent Botti. Introducing the new paradigm of social dispersed computing: applications, technologies and challenges. Journal of Systems Architecture, 91:83–102, 2018.

[62] Bing Lin et al. A time-driven data placement strategy for a scientific workflow combining edge computing and cloud computing. IEEE Transactions on Industrial Informatics, 15(7), 2019.

[63] Lin Wang, Lei Jiao, Ting He, Jun Li, and Max Mühlhäuser. Service entity placement for social virtual reality applications in edge computing. In IEEE INFOCOM 2018-IEEE Conference on Computer Communications, pages 468–476. IEEE, 2018.

[64] Quyuan Luo, Changle Li, Tom H Luan, and Weisong Shi. Edgevcd: Intelligent algorithm inspired content distribution in vehicular edge computing network. IEEE Internet of Things Journal, 2020.

[65] Adrian-Cristian Nicolaescu, Spyridon Mastorakis, and Ioannis Psaras. Store edge networked data (send): A data and performance driven edge storage framework. In IEEE INFOCOM 2021 - IEEE Conference on Computer Communications, pages 1–10, 2021.

[66] M. Al Azad and S. Mastorakis. Reservoir: Named data for pervasive computation reuse at the network edge. In 2022 IEEE International Conference on Pervasive Computing and Communications (PerCom), pages 141–151, Los Alamitos, CA, USA, mar 2022. IEEE Computer Society.

[67] Angela Ning Ye, Zhiming Hu, Caleb Phillips, and Iqbal Mohomed. Alertme: Towards natural language-based live video trigger systems at the edge. In

Proceedings of the 4th International Workshop on Edge Systems, Analytics and Networking, EdgeSys '21, page 67–72, New York, NY, USA, 2021. Association for Computing Machinery.

[68] Bo Hu and Wenjun Hu. Linkshare: Device-centric control for concurrent and continuous mobile-cloud interactions. In Proceedings of the 4th ACM/IEEE Symposium on Edge Computing, SEC '19, page 15–29, New York, NY, USA, 2019. Association for Computing Machinery.

[69] Youngki Lee, Younghyun Ju, Chulhong Min, Seungwoo Kang, Inseok Hwang, and Junehwa Song. Comon: Cooperative ambience monitoring platform with continuity and benefit awareness. In Proceedings of the 10th International Conference on Mobile Systems, Applications, and Services, MobiSys '12, page 43–56, New York, NY, USA, 2012. Association for Computing Machinery.

[70] Jibin Wang, Zhigang Zhao, Zhaogang Xu, Hu Zhang, Liang Li, and Ying Guo. I-sieve: An inline high performance deduplication system used in cloud storage. Tsinghua Science and Technology, 20(1):17–27, 2015.

[71] Jonathan Lee, Abderrahmen Mtibaa, and Spyridon Mastorakis. A case for compute reuse in future edge systems: An empirical study. In 2019 IEEE Globecom Workshops (GC Wkshps), pages 1–6, 2019.

[72] Lixia Zhang, Alexander Afanasyev, Jeffrey Burke, Van Jacobson, kc claffy, Patrick Crowley, Christos Papadopoulos, Lan Wang, and Beichuan Zhang. Named data networking. SIGCOMM Comput. Commun. Rev., 44(3):66–73, jul 2014.

[73] Spyridon Mastorakis, Abderrahmen Mtibaa, Jonathan Lee, and Satyajayant Misra. ICedge: When Edge Computing Meets Information-Centric Networking. IEEE Internet of Things Journal, 7(5):4203–4217, 2020.

[74] Jiayi Meng, Sibendu Paul, and Y. Charlie Hu. Coterie: Exploiting Frame Similarity to Enable High-Quality Multiplayer VR on Commodity Mobile

Devices, page 923–937. Association for Computing Machinery, New York, NY, USA, 2020.

[75] Anirudh Sabnis, Tareq Si Salem, Giovanni Neglia, Michele Garetto, Emilio Leonardi, and Ramesh K. Sitaraman. Grades: Gradient descent for similarity caching. In IEEE INFOCOM 2021 - IEEE Conference on Computer Communications, pages 1–10, 2021.

[76] Alexandr Andoni, Piotr Indyk, Thijs Laarhoven, Ilya Razenshteyn, and Ludwig Schmidt. Practical and optimal lsh for angular distance. In Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 1, NIPS'15, page 1225–1233, Cambridge, MA, USA, 2015. MIT Press.

[77] Anirban Dasgupta, Ravi Kumar, and Tamas Sarlos. Fast locality-sensitive hashing. In Proceedings of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD '11, page 1073–1081, New York, NY, USA, 2011. Association for Computing Machinery.

[78] Qin Lv, William Josephson, Zhe Wang, Moses Charikar, and Kai Li. Multi-probe lsh: Efficient indexing for high-dimensional similarity search. In Proceedings of the 33rd International Conference on Very Large Data Bases, VLDB '07, page 950–961. VLDB Endowment, 2007.

[79] Wei Zhang, Ke Gao, Yong-dong Zhang, and Jin-tao Li. Data-oriented locality sensitive hashing. In Proceedings of the 18th ACM International Conference on Multimedia, MM '10, page 1131–1134, New York, NY, USA, 2010. Association for Computing Machinery.

[80] Antonios Katsarakis, Yijun Ma, Zhaowei Tan, Andrew Bainbridge, Matthew Balkwill, Aleksandar Dragojevic, Boris Grot, Bozidar Radunovic, and Yongguang Zhang. Zeus: Locality-aware distributed transactions. In Proceedings of the Sixteenth European Conference on Computer Systems, EuroSys '21,

page 145–161, New York, NY, USA, 2021. Association for Computing Machinery.

[81] Narayanan Sundaram, Aizana Turmukhametova, Nadathur Satish, Todd Mostak, Piotr Indyk, Samuel Madden, and Pradeep Dubey. Streaming similarity search over one billion tweets using parallel locality-sensitive hashing. Proc. VLDB Endow., 6(14):1930–1941, sep 2013.

[82] Batyr Charyyev and Mehmet Hadi Gunes. Locality-sensitive iot network traffic fingerprinting for device identification. IEEE Internet of Things Journal, 8(3):1272–1281, 2021.

[83] Wenquan Xu, Haoyu Song, Linyang Hou, Hui Zheng, Xinggong Zhang, Chuwen Zhang, Wei Hu, Yi Wang, and Bin Liu. Soda: Similar 3d object detection accelerator at network edge for autonomous driving. In IEEE INFOCOM 2021 - IEEE Conference on Computer Communications, pages 1–10, 2021.

[84] Peizhen Guo, Bo Hu, Rui Li, and Wenjun Hu. Foggycache: Cross-device approximate computation reuse. In Proceedings of the 24th Annual International Conference on Mobile Computing and Networking, 2018.

[85] Penghao Zhang, Heng Pan, Zhenyu Li, Peng He, Zhibin Zhang, Gareth Tyson, and Gaogang Xie. Accelerating lsh-based distributed search with in-network computation. In IEEE INFOCOM 2021 - IEEE Conference on Computer Communications, pages 1–10, 2021.

[86] Stephen Farrell, Dirk Kutscher, Christian Dannewitz, Börje Ohlman, Ari Keränen, and Phillip Hallam-Baker. Naming Things with Hashes. RFC 6920, April 2013.

[87] Lorenzo Saino, Ioannis Psaras, and George Pavlou. Icarus: a caching simulator for information centric networking (icn). In SimuTools. ICST, 2014.

[88] Xulong Tang, Mahmut Taylan Kandemir, and Mustafa Karakoy. Mix and match: Reorganizing tasks for enhancing data locality. In Abstract Proceedings of the 2021 ACM SIGMETRICS / International Conference on Measurement and Modeling of Computer Systems, SIGMETRICS '21, page 47–48, New York, NY, USA, 2021. Association for Computing Machinery.

[89] Ke Zhang et al. Mobile-Edge Computing for Vehicular Networks. IEEE Vehicular Technology Magazine, 12(June):2–10, 2017.

[90] M. Satyanarayanan. The emergence of edge computing. Computer, 50(1):30–39, Jan. 2017.

[91] Joel Obstfeld. Global engineering:Enabling a Connected Transport future, 7 2019.

[92] Lixia Zhang. The role of data repositories in named data networking. In 2019 IEEE International Conference on Communications Workshops (ICC Workshops), pages 1–5, May 2019.

[93] I Lujic and H Truong. Architecturing elastic edge storage services for data-driven decision making. In 13th European Conference on Software Architecture (ECSA), September 2019.

[94] Ari Keränen et al. The ONE simulator for DTN protocol evaluation. Proceedings of the Second International ICST Conference on Simulation Tools and Techniques, 2009.

[95] S. Khare, H. Sun, K. Zhang, J. Gascon-Samson, A. Gokhale, X. Koutsoukos, and H. Abdelaziz. Scalable edge computing for low latency data dissemination in topic-based publish/subscribe. In 2018 IEEE/ACM Symposium on Edge Computing (SEC), pages 214–227, Oct 2018.

[96] TfL. Case study: Transport for Londom. pages 1–2, 2000.

[97] Eton Manor. CASE STUDY Wi-fi access at London 2012 The London 2012 Olympic Park wi-fi network delivered 100 per cent availability throughout the Games. 2012.

[98] Junguk Cho et al. Acacia: Context-aware edge computing for continuous interactive applications over mobile networks. In Proceedings of the 12th International on Conference on Emerging Networking EXperiments and Technologies, CoNEXT '16, pages 375–389, New York, NY, USA, 2016. ACM.

[99] Timespace. Timespace V400, 2018. Issue 4.

[100] Anil Madhavapeddy, Richard Mortier, Charalampos Rotsos, David Scott, Balraj Singh, Thomas Gazagnaire, Steven Smith, Steven Hand, and Jon Crowcroft. Unikernels: Library operating systems for the cloud. SIGARCH Comput. Archit. News, 41(1):461–472, March 2013.

[101] Yuan Ai, Mugen Peng, and Kecheng Zhang. Edge computing technologies for internet of things: a primer. Digital Communications and Networks, 4(2):77 – 86, 2018.

[102] G. S. Aujla, N. Kumar, A. Y. Zomaya, and R. Ranjan. Optimal decision making for big data processing at edge-cloud environment: An sdn perspective. IEEE Transactions on Industrial Informatics, 14(2):778–789, 2018.

[103] Michał Król, Karim Habak, David Oran, and Dirk Kutscher. RICE : Remote Method Invocation in ICN.

[104] Tsung-Yi Lin, Michael Maire, Serge J. Belongie, Lubomir D. Bourdev, Ross B. Girshick, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C. Lawrence Zitnick. Microsoft COCO: common objects in context. CoRR, abs/1405.0312, 2014.

[105] Danny Yuxing Huang, Noah Apthorpe, Gunes Acar, Frank Li, and Nick Feamster. Iot inspector: Crowdsourcing labeled network traffic from smart home devices at scale. arXiv preprint arXiv:1909.09848, 2019.

[106] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. Microsoft coco: Common objects in context. In European conference on computer vision. Springer, 2014.

[107] Eli Cortez et al. Resource central: Understanding and predicting workloads for improved resource management in large cloud platforms. In Proceedings of the 26th Symposium on Operating Systems Principles, pages 153–167, 2017.

[108] John Wilkes. Yet more Google compute cluster trace data. Google research blog, April 2020. Posted at `https://ai.googleblog.com/2020/04/yet-more-google-compute-cluster-trace.html`.

[109] Khalid Omer Mahgoub Saied. Network latency estimation leveraging network path classification, 2018.

[110] BGP in 2018 — The BGP Table, May 2020. [accessed 25 May 2022].

[111] AWS Architecture Blog: Internet Routing and Traffic Engineering, May 2020.

[112] Parisa Haghani, Sebastian Michel, Philippe Cudré-Mauroux, and Karl Aberer. Lsh at large - distributed knn search in high dimensions. In WebDB, 2008.

[113] Stuart Colianni. Mnist as .jpg. Kaggle database website, 2017. Posted at `https://www.kaggle.com/datasets/scolianni/mnistasjpg`.

[114] Amir Anhari. Alexa dataset. Kaggle database website, 2018. Posted at `https://www.kaggle.com/datasets/aanhari/alexa-dataset`.

[115] fluent.ai. Fluent speech commands: A dataset for spoken language understanding research. fluent.ai website, April 2020.

[116] Joseph Redmon, Santosh Kumar Divvala, Ross B. Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. CoRR, abs/1506.02640, 2015.

[117] Theodoros Giannakopoulos. pyaudioanalysis: An open-source python library for audio signal analysis. PLOS ONE, 10(12):1–17, 12 2015.

[118] D. Huggins-Daines, M. Kumar, A. Chan, A.W. Black, M. Ravishankar, and A.I. Rudnicky. Pocketsphinx: A free, real-time continuous speech recognition system for hand-held devices. In 2006 IEEE International Conference on Acoustics Speech and Signal Processing Proceedings, volume 1, pages I–I, 2006.

[119] ETSI. ETSI GS NFV-IFA 005, 8 2018.

[120] ETSI. OSM VNF ONBOARDING GUIDELINES, 6 2019.