Store Edge Networked Data (SEND): A Data and Performance Driven Edge Storage Framework

Adrian-Cristian Nicolaescu
University College London
a.nicolaescu@ee.ucl.ac.uk

Spyridon Mastorakis
University of Nebraska at Omaha
smastorakis@unomaha.edu

Ioannis Psaras
University College London
i.psaras@ucl.ac.uk

***Abstract*—The number of devices that the edge of the Internet accommodates and the volume of the data these devices generate are expected to grow dramatically in the years to come. As a result, managing and processing such massive data amounts at the edge becomes a vital issue. This paper proposes "Store Edge Networked Data" (SEND), a novel framework for in-network storage management realized through data repositories deployed at the network edge. SEND considers different criteria (e.g., data popularity, data proximity from processing functions at the edge) to intelligently place different categories of raw and processed data at the edge based on system-wide identifiers of the data context, called *labels*. We implement a data repository prototype on top of the Google file system, which we evaluate based on real-world datasets of images and Internet of Things device measurements. To scale up our experiments, we perform a network simulation study based on synthetic and real-world datasets evaluating the performance and trade-offs of the SEND design as a whole. Our 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.**

***Index Terms*—Edge computing, Internet of Things (IoT), Data storage at the edge, Data management**

I. Introduction

Traditionally, the flow of data on the Internet follows a "core-to-edge" model based on the assumption that the data reside in the core of the network infrastructure (e.g., in a data-center) and is requested/consumed by users at the edge of the network. 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, in this paper, we propose a data storage and management framework at the network edge, called "Store Edge Networked Data" (SEND). 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.

In this paper, we make two main contributions:

- We present the design of SEND, 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.
- We perform a thorough evaluation study of SEND. First, we implement an EDR prototype on top of the Google file system [4], which we evaluate based on real-world datatets of images and measurements/readings of IoT devices. To scale up our experiments, we perform a network simulation study 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 our paper is organized as follows. In Section II, we present a brief background on storage frameworks and discuss prior related work. In Section III, we present our SEND system model and assumptions, while in Section IV, we present the design of SEND. In Section V, we present our experimental evaluation, in Section VI, we discuss various considerations and extensions of the SEND design, and,

finally, in Section VII, we conclude our work.

II. Background and Prior Related Work

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.

A. Storage Frameworks

Extensive research has been conducted on storage frameworks. The community has explored distributed file systems, such as the Network File System (NFS) [5] and Coda [6], and more recently Hadoop [7] and the Google file system [4]. Alternative database designs (e.g., relational, object-oriented, graph-based) have been proposed for data storage and indexing of the stored data [8]. Lately, the community has focused on key-value stores for the deployment of distributed network storage and middleware applications [9], including mechanisms to increase the performance of such stores through programmable hardware [10]. 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.

B. 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 [11], [12], while solutions to improve the availability of cloud storage services have been also investigated [13].

Data deduplication designs have been studied to ensure that a single copy of redundant data is stored on the cloud [14], [15], 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 [16], [17] and perform operations, such as data searches, in a privacy-preserving manner [18]. Finally, issues, such as multi-tenancy [19], data replication [20], and data management [21], 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.

C. 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 [22]. Liu *et al.* investigated the impact of the size and number of data storage units on the data availability and operational cost [23].

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 [24], [25], [26]. 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 [27].

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 [28]. Social virtual reality applications may require considering the data of users and the processing logic on these data as a bundle [29]. 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 [30].

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.

III. System Model and Assumptions

As we illustrate in Figure 1, we assume an edge computing environment, where a number of EDRs (EDR_1, EDR_2,..., EDR_N) have been deployed one-hop away from user devices. We further assume that the devices generate data based on their nature or the applications they may be running (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 these data are 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 at the edge. As we will discuss in Section IV-B, 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.

We assume that a number of processing functions (e.g., object recognition, data analytics) may be offered by an EDR. We further assume that 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, etc., which utilize edge services and network data storage for the data needed by them or their users.

In the context of our system, we assume the existence of a logically centralized EDR management module at each edge network. This component can be realized through, for instance, an SDN controller. 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.

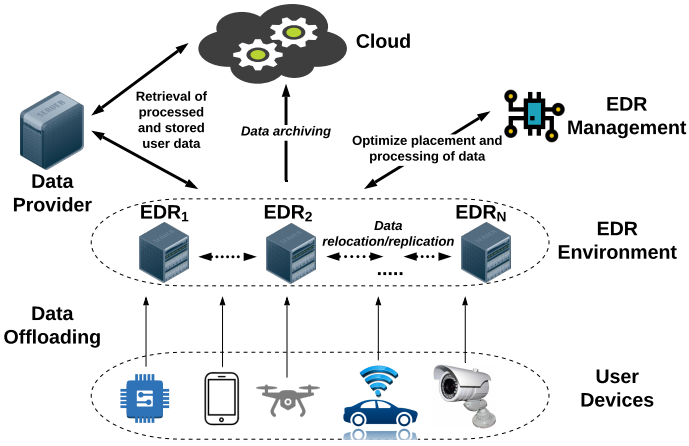


Fig. 1: SEND System Model. Data offloaded by user devices are stored in the EDR environment, which is managed by a logically centralized EDR management entity. Data providers can retrieve raw and processed data from EDRs or the cloud once the data are archived from the edge to the cloud.

IV. SEND Design

In this section, we first present an overview of the SEND design and, subsequently, its design components in detail.

A. SEND Design Overview

In Figure 2, we present an overview of the SEND design and operational workflow. As illustrated on the left of Figure 2, 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 are introduced to the system. As a result, data are 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 the retrieval of the data themselves.

As illustrated on the right side of Figure 2, SEND interacts with network flows, accepting data that transit 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) for the execution of such functions; and (ii) storage-related data that are 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. This "usefulness" time period is represented by the shelf life of the data, which we further discuss in Section IV-B. Once the shelf life of raw or processed data expires, these data may be offloaded from the edge to the cloud for archival purposes. Note that all data attributes are considered important, however, the most important attribute for our design, which also has not been given much consideration until now, 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, we present the components of the SEND design in more detail. We first present different **data attributes and categories of data as defined in the context of SEND** (Section IV-B). We then present **how the most important data attributes (labels and hashes) improve data storage and servicing** (Section IV-C) and describe **the SEND data management operation and different strategies for the placement of data at the edge** (Section IV-D). Finally, we put all the components of the SEND design together to elaborate on **the overall operation process of EDRs** (Section IV-E).

B. 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 represents a deadline for the execution of a time-sensitive processing function, while the shelf life refers to the maximum storage period or the maximum window for the data to be processed by a function at the edge. Once the data shelf life expires, the data are archived to the cloud. These attributes help with data management decisions such as storage time, processing needs, and storage/archiving after processing.

Labels are a data characteristic flexible enough to be attributed to all types of data entering our system. This attribute is rather flexible—it can be lightweight so that it does not impose high overhead on the network or our system or it can have a more sophisticated form that provides deeper context into

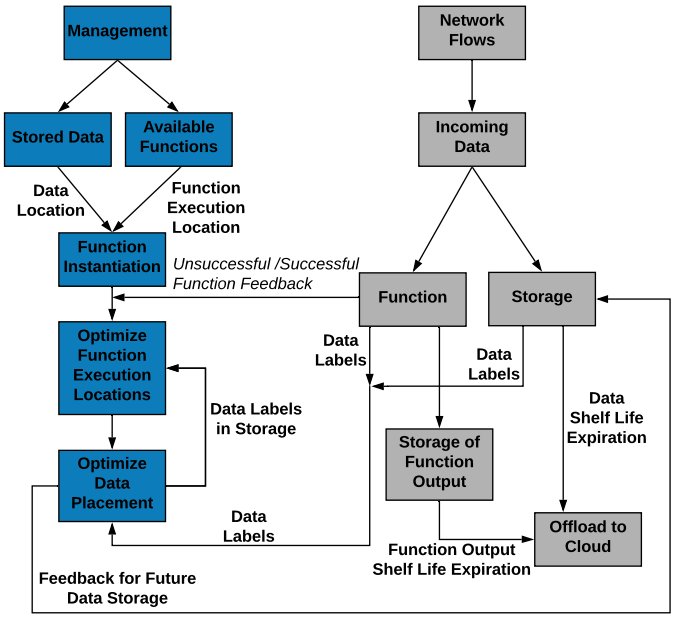


Fig. 2: SEND Design Overview. SEND seamlessly interacts with network flows to accept function- and storage-related data. Raw and processed data are stored at the edge until its shelf life expires. After shelf life expiration, the data are archived on the cloud. The SEND management plane receives feedback from EDRs about the labels of received data and successful/unsuccessful on-time completion of function requests to optimize function and data placement at the edge.

the data. For example, temperature data generated by an IoT sensor can be labelled based on its type (e.g., "temperature-data"), 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. 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:* These data have the purpose of being stored at the edge. Their future use may vary. For example, they can be used as input for the future execution of functions or be stored for distributed and time-constrained access.
- *Function-related data:* These data may be the function code 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:* These 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 its shelf life expires.

C. 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 IV-D) 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 their source to their 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 inspect certain data pieces to verify their validity.

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" are 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 context of the data that it handles and improves its performance by enabling accurate data placement and storage decisions.

D. 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.
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 they are 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.

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 need 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 have been already stored, data with the same label and generated within the last hour will also be given priority.

Popularity-based strategy: The data are 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 are placed close in terms of network hops to where functions that expect input data with the same label(s) have been already instantiated. The data are collected in common, pool-like storage, so that several functions can make use of them. For example, let us consider functions that expect input data with labels "video-data", "IoT-measurements", and "highway-traffic". When data with these labels are 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 are placed at EDRs based on its popularity.

E. EDR Operation

Having discussed the different components of SEND, in this subsection, we present the operation of EDRs (Algorithm 1). Once a user application/device offloads data or requests the execution of a function, these data or requests will reach the EDR closest to the user—for example, let us assume that EDR_1 of Figure 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 are available at EDR_N instead of EDR_1 in Figure 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/replication 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.

V. Evaluation

In this section, we present the evaluation of SEND using two different setups[1]. First, we implement an EDR prototype, which we evaluate based on real-world datasets. To scale up our experiments, we perform a network simulation study to evaluate the SEND framework as a whole.

A. EDR Prototype Experiments

We have implemented an EDR prototype on top of the Google file system [4]. We evaluated our prototype 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 our first scenario, we used 100,000 images from the COCO dataset [31]. For our second scenario, we used a dataset captured and publicized in the context of the IoT Inspector project [32] from 3,000 users and 30,000 IoT devices. For both scenarios, we evaluated the following metrics (without any network latency): (i) the time to insert a data piece (image or IoT measurement) to the EDR as we increase the number of data pieces inserted (load on the EDR); and (ii) the time to search for stored data based on labels and other attributes such

[1]We make our prototype and simulation code available to the community at https://github.com/⟨omitted-for-double-blind-review⟩.

ALGORITHM 1
EDR Operation

Inputs: Input i received from a user device
if (type(i) == data) **then**
    data_destination_EDR = data_placement_strategy(labels(i));
    if (data_destination_EDR == local_EDR) **then**
        store(i);
    else
        replicate(i, data_destination_EDR);
    end if
    label_statistics ls = update_label_population(labels(i));
else if (type(i) == function_request) **then**
    function_execution_EDR = routing_process(i);
    if (function_execution_EDR == local_EDR) **then**
        input = get_input_data(i);
        output o = execute(function_to_execute(i), input);
        output_EDR = data_placement_strategy(o);
        replicate(o, output_EDR);
    else
        distribute(i, function_execution_EDR);
    end if
    function_statistics fs = update_function_statistics(i);
end if
if (time_to_send_update()) **then**
    send_update_to_management_entity(ls, fs)
end if

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. We ran each experiment ten times and we report on the average results.

Data insertion time: In Figure 3, we present the insertion time results for every 20,000 data insertions for images and IoT measurements. For the image use-case, we vary the total number of labels for the inserted data. Our 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.

For the use-case of IoT measurements, we consider varying numbers of labels as well as data generation timestamps. Our results indicate that the different attributes do not significantly affect the insertion time. As we increase the total amount of inserted data, 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 also the total size of inserted data is smaller than the use-case of images.

Data lookup time: In Figure 4, we present the data lookup time results for every 20,000 insertions for both images and IoT measurements. Our results demonstrate that the lookup

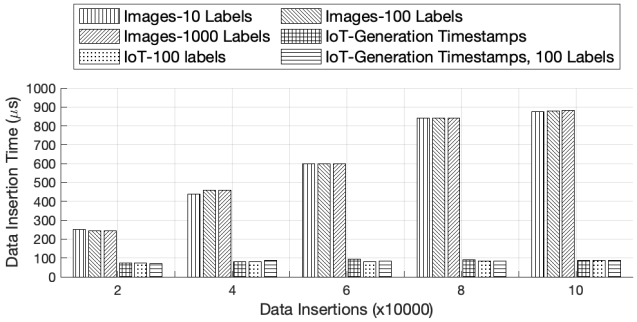


Fig. 3: Data insertion times. Results reported for every 20,000 insertions for both use-cases (images and IoT measurements).

time increases as we increase the total amount of stored data. The lookup time also increases with the complexity of the 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 ranges 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 pieces (IoT measurements) matching our lookup queries.

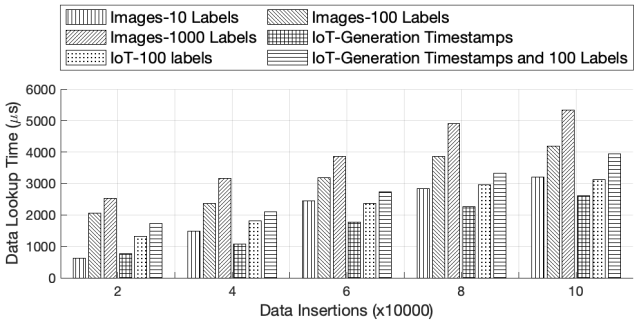


Fig. 4: Data lookup times. Results reported for every 20,000 insertions for both use-cases (images and IoT measurements).

B. SEND Simulation Experiments

Simulation setup, parameters, and evaluation metrics: For our simulation environment, we utilized the Icarus network simulator [33], a python-based, discrete-event simulator. We extended Icarus with an EDR implementation and the concept of data labels, and we also implemented the data placement strategies mentioned in Section IV-D.

In Figure 5, we present our tree-like topology, 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. We present our simulation parameters and datasets in Table I. At the beginning of each simulation, there is a data generation round where EDRs are populated with data. After this round, users continue to generate data, but also 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, we implement a strategy that optimizes data and function placement based

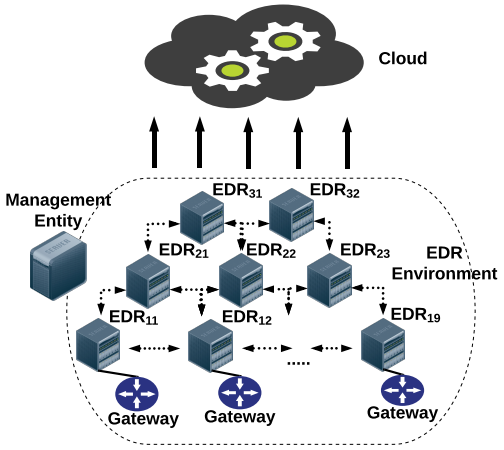


Fig. 5: Simulation topology.

on the number and deadlines of requests but without the use of labels [24], which we call "hybrid-no labels" strategy. To perform a fair comparison with this baseline strategy, we use the same method to generate request deadlines that Ascigil *et al.* used [24]. 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). We run each experiment ten times and report on the average results. 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 are replicated among EDRs according to the selected placement strategy, while once the data shelf life expires, the data are archived on the cloud. For our experiments, we use the following datasets: (i) a synthetic dataset; and (ii) a Microsoft Azure dataset [34] and a Google cloud dataset [35], 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, the archival of data from the edge to the cloud, and the feedback from and to the management entity.

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 I: Simulation parameters and used datasets.

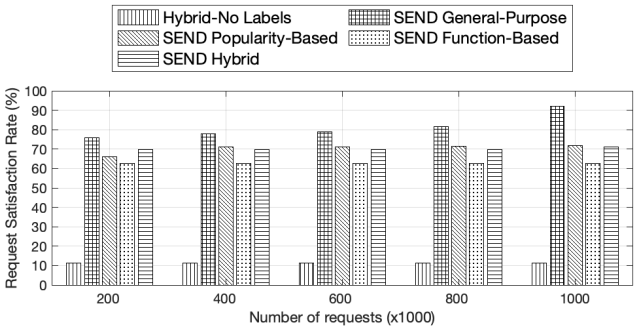
Parameter	Value(s)
Number of EDRs	14
Distance and link delay between EDRs at the highest edge level and cloud	3-5 hops and 40ms following values reported by recent studies [36], [37] and cloud computing trends [38]
Used Datasets	Synthetic dataset: 10,000 unique raw data pieces, 10,000 unique functions, 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 6, we present the request satisfaction rate results for different datasets. For our synthetic dataset (Figure 6a), 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, lower satisfaction rates than the general-purpose strategy, being able to satisfy on-time 61-72% of the requests.

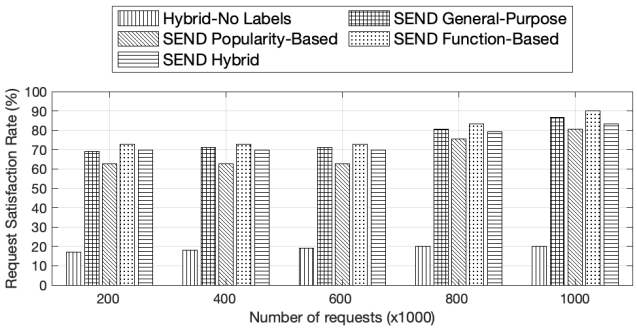
For the Azure dataset (Figure 6b), the function-based placement strategy achieves the highest request satisfaction rates (up to 90%). The SEND hybrid and the general-purpose strategies perform slightly worse than the function-based strategy reaching about 85% of satisfaction rates as the number of requests 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 6c), 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 demonstrates the importance of labels, which make SEND aware of the data context, for the efficient placement of both

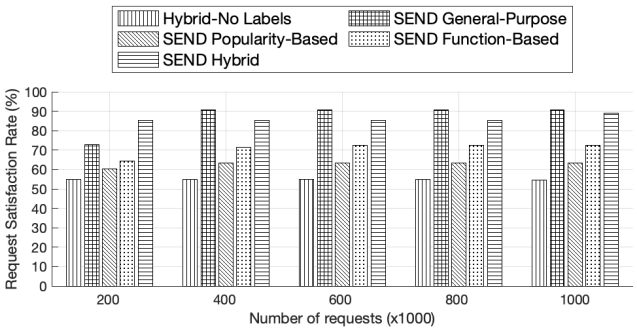
data and functions in the EDR environment.



(a) Satisfaction rate for the synthetic dataset.



(b) Satisfaction rate for the Microsoft Azure dataset.

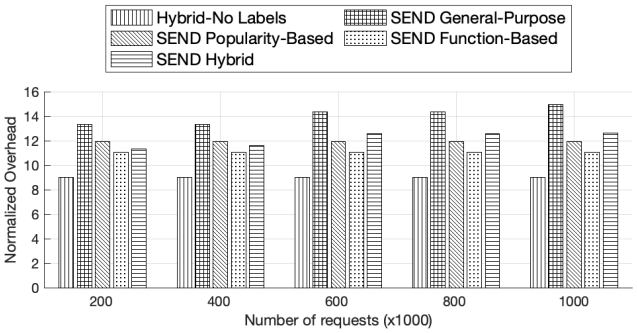


(c) Satisfaction rate for the Google cloud dataset.

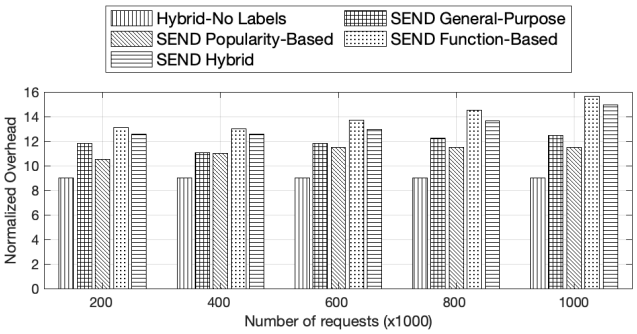
Fig. 6: Request satisfaction rates for different datasets.

In Figure 7, we present the overhead results for different datasets and numbers of user requests. For our synthetic dataset (Figure 7a), 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 are 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 6a). For the Azure dataset (Figure 7b), the function-based placement strategy incurs the highest overhead, since the majority of the requests in this dataset are requests for functions.

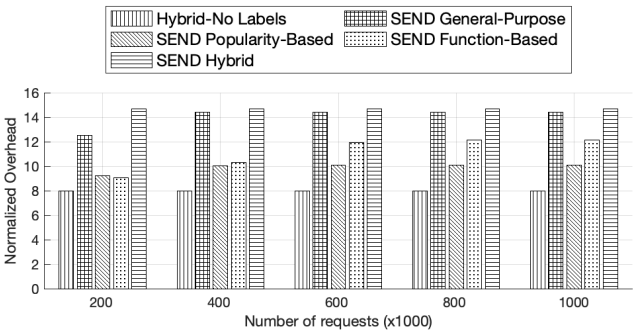
Finally, for the Google cloud dataset (Figure 7c), the SEND general-purpose and hybrid strategies result in the highest overheads. This dataset contains fewer labels compared to the previous datasets and a relatively balanced mix of requests for raw and processed data. To this end, the general-purpose strategy replicates the majority of data in this dataset, since data with the same label are typically already present in the



(a) Normalized overhead for the synthetic dataset.



(b) Normalized overhead for the Microsoft Azure dataset.



(c) Normalized overhead for the Google cloud dataset.

Fig. 7: Normalized overhead for different datasets.

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.

VI. Discussion

In this section, we discuss various extensions and considerations of the SEND design.

A. SEND Performance Optimizations

A number of different approaches can be applied to optimize the performance of SEND as a whole.

Optimal allocation of computing and storage resources: As we mentioned in Section IV, EDRs keep statistics about the data generated by users at the edge as well as the data attributes and the requested functions. In addition to being sent to the EDR management entity of the edge network, such statistics can also be sent to the cloud. On the cloud, the statistics can be combined with historical usage data of multiple edge networks maintained by the EDR providers. The EDR providers may utilize different models (based on machine

learning and other optimization methods) to determine the optimal allocation of computing and storage resources across different edge networks based on the projected user demand.

Computation reuse: The concept of computation reuse refers to reusing/sharing computation (results of function execution) among users for the execution of incoming function requests [39], [40]. As a result, computation reuse can reduce the function execution times and the usage of computing resources. To realize computation reuse, results of previously executed functions will be stored by EDRs, which will determine whether these results can satisfy incoming requests without the need to execute incoming requests from scratch. We will explore this direction in our future work.

Accurate data and function placement: In Section IV-D, we mentioned examples of placement strategies. Such strategies can be augmented with reinforcement and machine learning approaches for the accurate prediction of when and what types of data and functions to be placed at different EDRs. This will increase the likelihood of placing the most relevant data and functions as close as possible to where they may be requested by users in the future. The reinforcement and machine learning approaches can be located at the management entity of each edge network or further up at strategic locations of the core network, having visibility across multiple edge networks.

Data naming, indexing, and compression: Mechanisms for data naming/identification and indexing within storage can enable efficient data lookup, insertion, and other operations. Compression techniques may also be able to reduce the overall size of the required storage while preserving throughput [41].

B. Security and Privacy Considerations

A storage environment built around SEND at the edge involves security considerations. A first consideration is related to malicious users. Let us consider a vehicular use-case, where a mobile navigation application is used to find the fastest route to a destination. The application may be reporting its speed and ongoing traffic conditions over time, so that the edge can compute the fastest routes for other drivers that use the same application. However, malicious users may report bogus speeds or traffic conditions. Such data may be stored by EDRs and may also be archived on the cloud in the long run to help the city authorities with traffic planning. Mechanisms involving the use of machine learning or statistical modeling [42] can be utilized to identify and flag potential bogus data and processed results that might involve bogus input data.

Another issue to consider is the existence of malicious EDRs, which can alter stored data as well as the code and results of functions. Verifiable proofs of computation [43] may enable EDRs and users to verify the validity of function execution results. Moreover, data generated by users and stored by an EDR may be logged into a distributed ledger, such as a blockchain, accessible by EDRs to avoid modifications of data as they get replicated, processed, or relocated at the edge.

Finally, storing user data at EDRs allow EDR providers to have access to the user data and their attributes, effectively being able to identify the types of data that a user device may be generating and the application(s) that it might be running. Encryption mechanisms, such as attribute-based encryption [44], can be utilized to "hide" the data and their attributes from EDR providers and enable access to it only by authorized users and/or data providers. However, such mechanisms may come at a cost of sub-optimal management operations, since SEND utilizes data attributes to make efficient management decisions. Alternative approaches may allow access to certain attributes to help SEND efficiently manage the data, but encrypt the actual data and attributes that are considered sensitive.

C. Techno-Economic Aspects

SEND will impact several techno-economic aspects of edge computing ecosystems. First of all, pricing schemes are needed for users and data providers that occupy storage and computing resources at the edge. For example, an intuitive solution might be for SEND to inherit the "pay-as-you-go" model of cloud computing. More sophisticated approaches may involve auctions between users or data providers and EDR providers [45], as well as optimizations for profit maximization [46]. Another challenge is related to enabling collaboration among EDR providers. Collaboration may be needed, for example, in order to exchange data that have been generated by users subscribed to a particular EDR provider but is requested by users subscribed to another EDR provider. Smart contracts can be utilized to provide a consistent definition of the collaboration terms and automate their execution [47].

VII. Conclusion and Future Work

In this paper, we presented an in-network data storage management framework, named SEND. SEND is realized through the deployment of EDRs (1-hop away from users), seamlessly interacting with network traffic originated from user devices. Through a logically centralized management plane, SEND utilizes data attributes to make intelligent decisions about the placement of functions and data among EDRs. We implemented an EDR prototype to evaluate the performance of EDR operations and conducted a network simulation study to evaluate the SEND design as a whole.

While SEND is off to a promising start, we plan to investigate the following directions in the future: (i) mechanisms to enhance the performance of SEND; (ii) mechanisms to enable the collaboration among EDR providers for the exchange of data requested by users subscribed to different providers; (iii) security and privacy challenges related to SEND; and (iv) the deployment of SEND in large-scale settings where it will interact with a variety of real-world applications.

Acknowledgments

This work was partially supported by the EPSRC early career fellowship In-Network Service Provisioning (INSP) under grant agreement number EP/M003787/1, a pilot award from the Center for Research in Human Movement Variability and the NIH (P20GM109090), and the National Science Foundation under award CNS-2016714.

REFERENCES

[1] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu, "Edge computing: Vision and challenges," *IEEE internet of things journal*, vol. 3, no. 5, pp. 637–646, 2016.

[2] I. Psaras, O. Ascigil, S. Rene, G. Pavlou, A. Afanasyev, and L. Zhang, "Mobile data repositories at the edge," in {*USENIX*} *Workshop on Hot Topics in Edge Computing (HotEdge 18)*, 2018.

[3] M. Satyanarayanan, "The emergence of edge computing," *Computer*, vol. 50, no. 1, pp. 30–39, 2017.

[4] S. Ghemawat, H. Gobioff, and S.-T. Leung, "The google file system," in *Proceedings of the nineteenth ACM symposium on Operating systems principles*, 2003, pp. 29–43.

[5] D. Hitz, J. Lau, and M. A. Malcolm, "File system design for an nfs file server appliance." in *USENIX winter*, vol. 94, 1994.

[6] J. J. Kistler and M. Satyanarayanan, "Disconnected operation in the coda file system," *ACM Transactions on Computer Systems (TOCS)*, 1992.

[7] K. Shvachko, H. Kuang, S. Radia, and R. Chansler, "The hadoop distributed file system," in *2010 IEEE 26th symposium on mass storage systems and technologies (MSST)*. Ieee, 2010, pp. 1–10.

[8] R. Angles and C. Gutierrez, "Survey of graph database models," *ACM Computing Surveys (CSUR)*, vol. 40, no. 1, pp. 1–39, 2008.

[9] X. Jin *et al.*, "Netcache: Balancing key-value stores with fast in-network caching," in *Proceedings of the 26th Symposium on Operating Systems Principles*, 2017, pp. 121–136.

[10] M. Blott, K. Karras, L. Liu, K. Vissers, J. Bär, and Z. 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.

[11] Y. Ye, L. Xiao, I.-L. Yen, and F. Bastani, "Secure, dependable, and high performance cloud storage," in *2010 29th IEEE Symposium on Reliable Distributed Systems*. IEEE, 2010, pp. 194–203.

[12] E. Stefanov and E. Shi, "Oblivistore: High performance oblivious cloud storage," in *2013 IEEE Symposium on Security and Privacy*. IEEE, 2013, pp. 253–267.

[13] B. Mao, S. Wu, and H. Jiang, "Improving storage availability in cloud-of-clouds with hybrid redundant data distribution," in *2015 IEEE International Parallel and Distributed Processing Symposium*. IEEE, 2015, pp. 633–642.

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

[15] D. Harnik, B. Pinkas, and A. Shulman-Peleg, "Side channels in cloud services: Deduplication in cloud storage," *IEEE Security & Privacy*, vol. 8, no. 6, pp. 40–47, 2010.

[16] C. Wang, S. S. Chow, Q. Wang, K. Ren, and W. Lou, "Privacy-preserving public auditing for secure cloud storage," *IEEE transactions on computers*, vol. 62, no. 2, pp. 362–375, 2011.

[17] S. Kamara and K. Lauter, "Cryptographic cloud storage," in *International Conference on Financial Cryptography and Data Security*. Springer, 2010, pp. 136–149.

[18] Q. Liu, G. Wang, and J. Wu, "Secure and privacy preserving keyword searching for cloud storage services," *Journal of network and computer applications*, vol. 35, no. 3, pp. 927–933, 2012.

[19] J. M. A. Calero, N. Edwards, J. Kirschnick, L. Wilcock, and M. Wray, "Toward a multi-tenancy authorization system for cloud services," *IEEE Security & Privacy*, vol. 8, no. 6, pp. 48–55, 2010.

[20] Q. Wei *et al.*, "CDRM: a cost-effective dynamic replication management scheme for cloud storage cluster," in *2010 IEEE international conference on cluster computing*. IEEE, 2010, pp. 188–196.

[21] A. Prahlad *et al.*, "Cloud gateway system for managing data storage to cloud storage sites," Dec. 30 2010, uS Patent App. 12/751,953.

[22] A.-C. Nicolaescu, O. Ascigil, and I. 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*, 2019, p. 41–47.

[23] J. Liu, M. L. Curry, C. Maltzahn, and P. 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.

[24] O. Ascigil *et al.*, "On uncoordinated service placement in edge-clouds," in *2017 IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*. IEEE, 2017, pp. 41–48.

[25] T. Ouyang, Z. Zhou, and X. Chen, "Follow me at the edge: Mobility-aware dynamic service placement for mobile edge computing," *IEEE Journal on Selected Areas in Communications*, vol. 36, no. 10, 2018.

[26] M. García-Valls, A. Dubey, and V. Botti, "Introducing the new paradigm of social dispersed computing: applications, technologies and challenges," *Journal of Systems Architecture*, vol. 91, pp. 83–102, 2018.

[27] 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)*, 2017, pp. 344–352.

[28] B. Lin *et al.*, "A time-driven data placement strategy for a scientific workflow combining edge computing and cloud computing," *IEEE Transactions on Industrial Informatics*, vol. 15, no. 7, 2019.

[29] L. Wang, L. Jiao, T. He, J. Li, and M. Mühlhäuser, "Service entity placement for social virtual reality applications in edge computing," in *IEEE INFOCOM 2018-IEEE Conference on Computer Communications*. IEEE, 2018, pp. 468–476.

[30] Q. Luo, C. Li, T. H. Luan, and W. Shi, "Edgevcd: Intelligent algorithm inspired content distribution in vehicular edge computing network," *IEEE Internet of Things Journal*, 2020.

[31] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick, "Microsoft coco: Common objects in context," in *European conference on computer vision*. Springer, 2014.

[32] D. Y. Huang, N. Apthorpe, G. Acar, F. Li, and N. Feamster, "Iot inspector: Crowdsourcing labeled network traffic from smart home devices at scale," *arXiv preprint arXiv:1909.09848*, 2019.

[33] L. Saino, I. Psaras, and G. Pavlou, "Icarus: a caching simulator for information centric networking (icn)," in *SimuTools*. ICST, 2014.

[34] E. 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*, 2017, pp. 153–167.

[35] J. Wilkes, "Yet more Google compute cluster trace data," Google research blog, Mountain View, CA, USA, Apr. 2020, posted at https://ai.googleblog.com/2020/04/yet-more-google-compute-cluster-trace.html.

[36] K. Omer Mahgoub Saied, "Network latency estimation leveraging network path classification," 2018.

[37] "BGP in 2018 — The BGP Table," May 2020, [Online; accessed 29. May 2020]. [Online]. Available: https://blog.apnic.net/2019/01/16/bgp-in-2018-the-bgp-table/

[38] "AWS Architecture Blog: Internet Routing and Traffic Engineering," May 2020. [Online]. Available: https://aws.amazon.com/blogs/architecture/internet-routing-and-traffic-engineering/

[39] P. Guo, B. Hu, R. Li, and W. Hu, "Foggycache: Cross-device approximate computation reuse," in *Proceedings of the 24th Annual International Conference on Mobile Computing and Networking*, 2018.

[40] P. Guo and W. 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*, 2018, pp. 271–284.

[41] B. Nicolae, "High throughput data-compression for cloud storage," in *International Conference on Data Management in Grid and P2P Systems*. Springer, 2010, pp. 1–12.

[42] R. Sommer and V. Paxson, "Outside the closed world: On using machine learning for network intrusion detection," in *2010 IEEE symposium on security and privacy*. IEEE, 2010, pp. 305–316.

[43] B. Parno, J. Howell, C. Gentry, and M. Raykova, "Pinocchio: Nearly practical verifiable computation," in *2013 IEEE Symposium on Security and Privacy*. IEEE, 2013, pp. 238–252.

[44] V. Goyal, O. Pandey, A. Sahai, and B. Waters, "Attribute-based encryption for fine-grained access control of encrypted data," in *Proceedings of the 13th ACM conference on Computer and communications security*, 2006, pp. 89–98.

[45] N. C. Luong, Z. Xiong, P. Wang, and D. Niyato, "Optimal auction for edge computing resource management in mobile blockchain networks: A deep learning approach," in *2018 IEEE International Conference on Communications (ICC)*. IEEE, 2018, pp. 1–6.

[46] A. Kiani and N. Ansari, "Toward hierarchical mobile edge computing: An auction-based profit maximization approach," *IEEE Internet of Things Journal*, vol. 4, no. 6, pp. 2082–2091, 2017.

[47] K. Christidis and M. Devetsikiotis, "Blockchains and smart contracts for the internet of things," *Ieee Access*, vol. 4, pp. 2292–2303, 2016.