Caching Popular Transient IoT Contents in an SDN-based Edge Infrastructure

Giuseppe Ruggeri, Marica Amadeo, Claudia Campolo, Antonella Molinaro, Antonio Iera

***Abstract*—With more than 75 billions of objects connected by 2025, Internet of Things (IoT) is the catalyst for the digital revolution, contributing to the generation of big amounts of (transient) data, which calls into question the storage and processing performance of the conventional cloud. Moving storage resources at the edge can reduce the data retrieval latency and save core network resources, albeit the actual performance depends on the selected caching policy. Existing edge caching strategies mainly account for the content popularity as crucial decision metric and do not consider the transient feature of IoT data. In this paper, we design a caching orchestration mechanism, deployed as a network application on top of a software-defined networking Controller in charge of the edge infrastructure, which accounts for the nodes' storage capabilities, the network links' available bandwidth, and the IoT data lifetime and popularity. The policy decides *which IoT contents* have to be cached and *in which node* of a distributed edge deployment with limited storage resources, with the ultimate aim of minimizing the data retrieval latency. We formulate the optimal content placement through an Integer Linear Programming (ILP) problem and propose a heuristic algorithm to solve it. Results show that the proposal outperforms the considered benchmark solutions in terms of latency and cache hit probability, under all the considered simulation settings.**

***Index Terms*—Internet of Things, Edge Computing, Caching, Transient Contents, Software Defined Networking**

I. Introduction

Internet of Things (IoT) envisions billions of massively connected devices able to sense the surrounding physical world. Data generated by IoT sources (producers) can be cached, processed and used by a plethora of applications (consumers) for different vertical markets, such as automotive, e-health, smart energy, industry 4.0. The conventional approach is to send such a big amount of IoT data to the remote cloud where they are stored and processed [1]. The cloud is the perfect candidate for managing long-term data storage and for executing complex analytics algorithms, thanks to its virtually unlimited storage and processing capabilities. This solution also relieves the IoT producers from the burden to answer the requests from multiple consumers, but it charges the core network with the task of allocating network resources for carrying data requests and replies from/to multiple consumers. Edge computing solutions can tackle such issue [2] by bringing computing and storage services close to the end-users, where data are typically consumed.

According to this groundbreaking paradigm, in-network processing and storage capabilities can be allocated not only in purpose-built servers, but also across heterogeneous network edge nodes [3]. They range from network equipment, such as base stations, backhaul nodes and points of presence, to resource-constrained equipment, such as IoT gateways and end-user devices like smartphones and tablets. As a result, edge solutions can easily support interactive and time-critical applications, while reducing the amount of traffic crossing the core network segment. Notably, such technologies are not intended as a replacement of the remote cloud, but as a complement. They seek to enable storage and computing services anywhere, from the cloud to the things, and to realize the *cloud-to-things continuum*.

Caching IoT contents at the network edge has been recognized as an effective solution to reduce both the content retrieval delay [4], [5] and the data redundancy in the core network, caused by multiple deliveries of the same content (such as the smart meter's measured values or the location of a tracked vehicle) to multiple interested consumers [6]. Nonetheless, deciding *which IoT content* is to be cached and *in which edge node* entails peculiar challenges, not yet fully addressed in the related literature.

First, most of the existing caching policies are designed for traditional Internet contents like multimedia files [7]. These solutions cannot be straightforwardly applied to IoT contents that are typically *transient*, i.e., they expire after a short time period since they have been generated at the source. This is the case of samples of a sensed phenomenon, such as a patient's blood pressure/heart rate values, smart meter readings, instantaneous road traffic conditions, which are regularly collected by IoT monitoring applications. Each generated sample has a limited validity (lifetime) until a new (fresher) sample is generated by the producer. Intuitively, it may not be convenient to cache popular contents which are ready to expire, since they could serve a few consumers only.

Second, resources heterogeneity in the distributed edge domain further challenges the caching decision. The few works considering edge caching of transient contents focus on a single edge node [5]. However, misplacing cached data in an edge domain with multiple nodes may increase the retrieval latency, especially when the same data is requested by multiple consumers and leveraged in different contexts [4].

Centralized orchestration can deal with the aforementioned challenges [8], [9]. In particular, a Software-defined Network-

Giuseppe Ruggeri, Marica Amadeo, Claudia Campolo, and Antonella Molinaro are with the DIIES Department, University Mediterranea of Reggio Calabria, Italy (corresponding author's e-mail: claudia.campolo@unirc.it). Antonella Molinaro is also with CentraleSupélec/L2S, Université Paris-Saclay, France. Antonio Iera is with the DIMES Department, Università della Calabria, Arcavacata di Rende, Italy. All authors are also with Consorzio Nazionale Interuniversitario per le Telecomunicazioni (CNIT).

This work has been partially supported by the "A COGnItive dynamic sysTem to allOw buildings to learn and adapt' (COGITO) project, funded by the Italian Government (PON ARS01_00836).

This is the post-print of the following article: G. Ruggeri, M. Amadeo, C. Campolo, A. Molinaro and A. Iera, "Caching Popular Transient IoT Contents in an SDN-based Edge Infrastructure," in IEEE Transactions on Network and Service Management. Article has been published in final form at: ieeexplore.ieee.org/abstract/document/9345760. DOI: 10.1109/TNSM.2021.3056891

ing (SDN) Controller can enforce effective caching policies in a given edge domain, and hide the complexity of the heterogeneous edge environment to end-users. In general, a Controller can count on its knowledge of the edge network's topology and available resources, i.e., nodes' storage capabilities and links' capacity, to make judicious caching decisions.

Our study intends to advance the state of the art, by designing a novel caching policy to be implemented at the SDN Controller in charge of a given edge domain. Such a policy identifies *the IoT contents* to be cached and *the cachers*, by *jointly* accounting for the *IoT content popularity and lifetime*, and targets the following main objectives: *(i)* to prioritize the cache hit ratio for those most popular contents which have a longer lifetime expectancy; *(ii)* to reduce the IoT content retrieval latency; *(iii)* to maximize the content diversity available at the edge domain. More specifically, in this paper the following main contributions are provided:

- We define the reference architecture supporting the proposed caching policy, implemented as a *network application* running on top of the SDN Controller.
- We formulate the optimal content placement through an Integer Linear Programming (ILP) problem, which encloses the aforementioned objectives, in order to identify the set of candidate cachers in the edge infrastructure. To the best of our knowledge, this is the first contribution towards caching optimization in a distributed edge domain taking into account both the popularity and the lifetime of IoT contents.
- We design and implement a heuristic algorithm aimed to approximate the optimal solution by solving the problem considerably faster, also under large-scale problem instances.
- We evaluate the proposal under different IoT content lifetime settings, differently sized edge domains with different caching capabilities, content popularity and request patterns. The solution is also compared with a baseline cloud storage solution and with a state-of-the-art policy not considering lifetime in the caching decision.

The remainder of the paper is organized as follows. Section II provides background material about caching in IoT as well as about SDN-aided caching strategies. The system design is reported in Section III, where the main entities are described as well as their functionalities. The formulated optimization problem is presented in Section IV, whereas the proposed heuristic algorithm is reported in Section V. Practical considerations concerning the retrieval of information feeding the proposed solution are reported in Section VI. Section VII reports the conducted evaluation study, before concluding in Section VIII with hints on future works.

II. Background and motivations

The large-scale deployment of IoT systems in crucial application domains, such as smart cities and logistics, has led to high demands on data storage and processing resources, which are traditionally hosted in cloud-based data centers.

The cloud offers long-term storage of *all* historical IoT data samples, e.g., environmental data periodically captured from sensors embedded in everyday objects. Starting from such big amount of data samples, cloud services can aggregate them and/or apply data analytics algorithms to feed intelligent applications and facilitate information access and sharing to end-users [10]. To limit the network traffic and the retrieval delay for consumers interested in IoT contents, storage resources can be moved at the edge, closer to the end-users.

Caching and replica placement have been largely investigated in past literature in the context of content delivery networks (CDN) [11], [12], distributed clouds [13], Information Centric Networking (ICN) [14] and web caching systems [15]. More recently, edge caching has been also considered, with decision strategies typically based on the content popularity [16]: the higher the popularity of a content the higher the number of requests that can be locally satisfied if the content is cached. For instance, the work in [17] has proposed an optimized replica placement algorithm that selects what contents to store at the edge by trading off between the storage cost, the retrieval latency and the content popularity. However, the focus of such studies has been mainly devoted to Internet contents like multimedia files which are non-transient [18]. A Youtube video or a movie on a streaming platform, once available online, does not change over time: they can be accessed by users even after years, unless the content provider does not explicitly delete them. Conversely, the majority of IoT contents are transient, with a varying lifetime that can range from a few seconds to hours or days. For instance, data collected by smart grid applications can have a longer lifetime than data needed for e-health monitoring applications. Therefore, ad-hoc designed caching strategies are required that are able to recognize the transiency of IoT data and take decisions accordingly.

A. Caching transient IoT contents

Although the considered IoT contents may expire after a certain amount of time, tens of works have indicated that caching IoT data of public interest is still extremely useful to improve the network performance [5], [6], [19]–[21]. A few distributed caching strategies for transient contents have been proposed in wireless sensor networks [22], [23], with the main target of reducing the data retrieval latency and the energy consumption in the network. There, the content lifetime becomes a crucial input in the caching decision of each sensor node: the higher the lifetime the higher the caching probability.

With focus on an edge infrastructure, the proposal in [6] defines a caching model that takes decision according to the interplay between data lifetime and multihop communications. Specifically, a cost function is defined and implemented at each potential cacher that trades off between two conflicting factors: the freshness cost, which is minimum when the data is fetched from its producer, and the communication cost, which is instead maximum in the same condition. A communication coefficient is defined to capture the relative importance that a consumer application can give to the communication cost, i.e., a high communication coefficient value indicates that the application prefers retrieving data from a close router and can tolerate less fresh results. Specific values of the aforementioned coefficient are not provided; they must be determined based on a service level agreement between IoT application providers and network operators.

Unlike our proposal, the aforementioned solutions rely on a distributed approach where decisions are taken autonomously according to locally estimated content and network parameters.

The decision metrics in [6] have been considered also in [5], but the authors focus on a single edge node only, e.g., a base station or a gateway. The conceived strategy trades off between freshness and communications costs, with the communication cost referring to the retrieval of the content from the single edge node or from the original producer. Deep reinforcement learning is used to predict the environment conditions and select the contents to cache, without the knowledge of future IoT data popularity. In [21], the authors propose a caching policy for a single edge node acting as gateway for a set of IoT producers. The strategy decides if caching a content based on its popularity and residual lifetime.

Unlike previous works [5], [21], our proposal focuses on a *distributed* edge infrastructure, where the potential cachers are network elements without stringent energy constraints, but with limited storage capabilities compared to the cloud.

The transient nature of IoT contents has also been captured by literature works focusing on the Age of Information (AoI) metric, which identifies the time elapsed since a cached content was created [24], [25]. AoI reflects how much outdated the responses from the caches can be: ideally, each consumer should not receive stale information but only the current version of the content. Most of the AoI related literature has provided decision strategies for optimal cache updates in scenarios characterized by strict network bandwidth limitations or intermittent connectivity, e.g., [24], [26]. Other works like [25] have investigated optimal packet scheduling algorithms that minimize AoI for resource-constrained IoT devices. Unlike these works, our paper is focused on edge caching of IoT contents that are periodically generated by IoT sources, e.g., for monitoring purposes. The considered contents are characterized by pre-defined application-specific lifetimes and are removed from the cache when their validity expires. Therefore, each cache can only deliver the current version of a content and does not store stale information.

B. SDN-aided caching

SDN is considered as a notable future Internet technology enabling a flexible network management [27]. The network control logic is implemented via a (logically) centralized Controller. Decoupling the control plane from the data plane allows to reduce the complexity of network nodes, which become simple forwarding elements. OpenFlow (OF) [28] is the most deployed southbound interface, i.e., the interface between the control and the data planes. It provides the ability of *programming* the flow tables of SDN switches by letting the Controller inject the forwarding rules. OF is responsible for feeding the Controller with data about the network topology, interface and traffic status. Such information is maintained in the Network Information Base (NIB) and accessed by the Controller to run the network applications, e.g., routing, load balancing, etc. Thanks to the knowledge of network-wide related parameters and the native capability to inject action rules in network nodes, the Controller can also play a crucial role to optimize *caching (re-)placement* decisions.

Related literature on the topic is still at its infancy, but the main trend is to deploy Controller-driven popularity-based caching schemes for non-transient data. In [7], the SDN switches maintain a hash table and a content store to provide, respectively, name-based content query and storage services. They also maintain a counter of the received content requests and report it to the Controller, which can infer the most popular contents of the domain. Based on such information, the Controller periodically makes content placement decisions and updates the switches' flow tables accordingly. An optimal content placement problem is formulated that aims to jointly limit the retrieval latency and the traffic between Internet service providers. In [29], the SDN switches support name-based data delivery and in-network caching via ICN. Content placement is managed by the Controller, which makes decisions for the most popular contents, according to an objective function that minimizes the retrieval latency, under the assumption that each popular content is cached at exactly one node in the domain. Our proposal builds upon the works in [7], [29], but we make a step forward by introducing a novel policy that aims at optimally caching transient IoT data at the edge by jointly considering content popularity and lifetime.

III. System Design

A. Scenario and components

As a reference scenario for our study, we consider a single edge domain (e.g., a metropolitan area network, a campus network) managed by a single operator and supervised by an SDN Controller. The domain is composed of a set of edge nodes connected to each other through wired links. An egress node connects the domain to the remote cloud through a core network segment, as graphically sketched in Fig. 1.

The cloud hosts IoT contents of public interest identified by an application-level unique *name*, similarly to [7], [30]. Targeted contents, generated by IoT producers such as sensors, cameras, smartphones, are not asynchronously generated, but *periodically refreshed* by the IoT producer, i.e., the result is a time series of content samples [31]. This is the case of samples of a monitored phenomenon like patient's blood pressure/heart rate values, pollution level indicators, road traffic conditions. Each content is transient, hence it is characterized by a *lifetime*, set by the IoT producer, which represents the duration for which a specific content item remains valid after its generation instant. At the expiration of the lifetime, the producer uploads the most recent (freshest) content item to the cloud.

Consumer applications are interested in the above mentioned transient IoT contents. Our design does not target a single IoT application but the class of monitoring applications (e.g., air/water monitoring, smart mobility, smart grid, etc.) that leverage the generation of IoT data with lifetimes ranging from (hence, cacheable for) a few seconds to some minutes or hours. For instance, many air quality monitoring applications or weather forecasting applications [32], [33] retrieve every minute information about air quality, temperature, humidity, wind direction and speed, through sensors installed in certain areas, and deliver it to environmental control agencies and to a variety of interested users. Sometimes, webcams or drones are also used to provide videos of the sensed areas.

Each application runs over a device connected to a single ingress node of the edge domain. In particular, generic monitoring applications that regularly collect the last updated sample of a given IoT content are considered.

The softwarized edge infrastructure is characterized by two main components:

- **Edge SDN nodes.** Augmented with storage capabilities, they can locally cache the transient IoT data, in order to serve multiple requests within the domain, with no need to contact the remote cloud[1].
- **SDN Controller.** A logically centralized Controller is assumed to oversee the edge SDN nodes and inject forwarding rules in their flow tables, as in legacy SDN deployments. It may belong to a hierarchical control plane architecture, as in [34], for scalability purposes and to avoid the single point of failure issue.

In our design, the Controller instructs SDN nodes to perform caching actions according to the decisions taken, at regular intervals, by a caching policy deployed as a network application, which identifies the cachers and the contents to cache (see Section IV). Given the transient nature of targeted contents, it is worth remarking that content placement decisions apply to a certain set of contents that (potentially multiple) consumer applications are interested to, and not to a single generated sample of a given content. If an SDN node has been elected as cacher for a certain content c_1, it can cache the current (not expired) sample of c_1 until a new content placement decision is taken. When the lifetime of a sample expires, the SDN node removes it from the cache and, when a new content request is received, it autonomously retrieves the freshest copy from the cloud, where it is regularly uploaded by the producer, and it will cache it again. By doing so, the implementation of the SDN node is kept simple, since it just adheres to the periodic Controller's instructions.

SDN, which is not proposed specifically for edge computing, can serve as a key enabler to lower the complexity barriers involved and disclose the full potential of this paradigm [8]. By deploying the edge caching orchestration within the SDN Controller, when taking caching decisions, the proposal can benefit from the built-in SDN routines for network status monitoring and from the native SDN programmability to more effectively build routing paths.

In particular, the SDN Controller has to monitor the caching capabilities of the SDN nodes in its domain and the status of the links interconnecting them. It has also to collect statistics about the IoT content request dynamics in the domain. To these purposes, the Controller leverages the OF southbound interface, properly overhauled as explained in Section VI. When a content request arrives which cannot be locally satisfied by the ingress node of the edge domain, the Controller is contacted which injects the proper forwarding rules in the SDN nodes' flow tables to enable the content retrieval from the cacher, if available. Otherwise, the request is forwarded outside the edge domain. However, it is worth remarking that the conceived caching strategy can be also deployed regardless the SDN implementation, and is applicable in general centrally-managed architectures.

[1]Describing the detailed implementation of SDN nodes is out of the scope of this paper. Either information-centric designs, like the one in [30], or proxy-based designs, like the one in [7], can be considered.

B. Targets

Our study aims at complementing the storage capability of the remote cloud by hosting IoT contents at the network edge, closer to the consumers. To this purpose, we define a caching policy implemented at the SDN Controller, which exploits the storage facilities provided on top of the softwarized edge infrastructure and identifies *which contents need to be cached* and *in which edge node*, by targeting three main objectives.

The first one is *to prioritize the cache hit ratio for those most popular contents which have a longer lifetime expectancy* and, therefore, can be requested a higher number of times during their validity period. Caching at the edge the most popular content that expires too soon may be less useful than caching a less popular content with a longer lifetime. Indeed, the former one could serve a few requests only before expiring and, then, the new freshest content sample must be retrieved from the cloud anyway. Conversely, caching resources could be more judiciously exploited to host a less popular content that satisfies more requests during its lifetime. To intuitively understand this concept, please refer to the following toy example. Let us assume that contents c_1 and c_2 have, respectively, average request rates $\lambda(c_1) = 1 request/s$ and $\lambda(c_2) = 0.7 request/s$, while their lifetimes are $T(c_1) = 2s$ and $T(c_2) = 10s$. Let us assume that, due to storage constraints, only one of the two contents can be cached at the edge. Since c_1 is more requested than c_2, intuitively, caching c_1 might seem more convenient than caching c_2; however the fact that $T(c_1) < T(c_2)$ changes things. Let us consider an observation window, TW, e.g., equal to 20s, it results that if c_1 is cached then c_2 has to be retrieved with rate $\lambda(c_2) = 0.7 request/s$ from the cloud, and also a new fresh copy of c_1 must be retrieved if it is expired during the considered observation window. Therefore, the average number of cache miss events at the edge, corresponding to the number of requests sent to the cloud, can be calculated as: $\lambda(c_2) * TW + (TW/T(c_1)) = 24$. Vice versa, if c_2 is cached then the average number of cache miss events is $\lambda(c_1) * TW + (TW/T(c_2)) = 22$. Therefore, by caching the less requested content, c_2, the number of times requests are sent to the cloud is actually reduced. When considering larger TW, caching c_2 is even more convenient.

The second objective is *to reduce the IoT contents retrieval latency* by placing the selected content copies in the most appropriate edge nodes, while matching their storage capability that is typically smaller compared to the one of the cloud. Although the targeted IoT applications have no strict delay constraints, speeding up the overall content retrieval process is crucial for both consumers and network operators. The latency reduction is measured in terms of gain achieved when retrieving the contents from the potential cachers compared to the case in which they are retrieved from the cloud.

The third objective is *to maximize the content diversity, by storing only one content copy at the edge* managed by a single provider. This avoids intra-domain cache redundancy and frees edge storage resources for caching a larger number

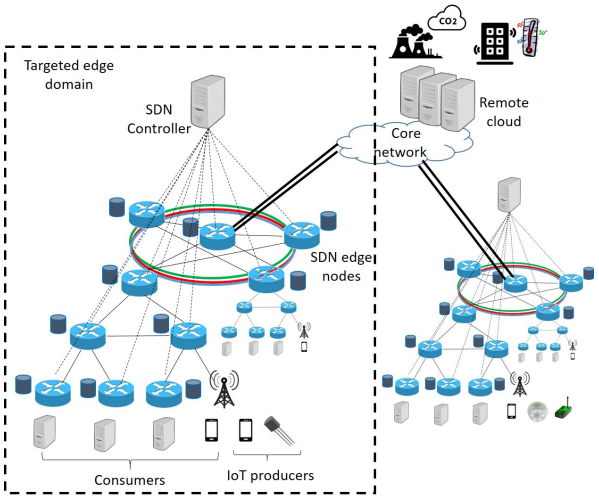


Fig. 1. Reference scenario and main components.

of distinct contents close to the consumers. Such an approach also leads to a reduction of the traffic exiting the domain for retrieving contents from the cloud, with great advantages for the networks operators. Indeed, as reported in [35], [36], links among different Internet Service Providers (ISPs) tend to be the bottlenecks of the Internet and they are also much more costly than ISP-internal links. Therefore, this objective translates into an operator-centric policy that aims at limiting the traffic transit costs.

To cope against possible intra-domain congestion issues deriving from the choice of caching a single content copy, we recall that thanks to built-in monitoring routines, leveraging the OF southbound interface, SDN nodes report to the Controller their status and make it aware of network links conditions of the overseen domain. This information is exploited by the conceived caching policy, which can identify the *best content placement that limits the retrieval delay* and, therefore, *the intra-domain traffic congestion*. Content placement is performed periodically and hence, if the traffic demand or the network conditions change, the Controller can perform a different allocation scheme in the next step. Additionally, the Controller can realize if a delivery path becomes congested and properly modify the routing tables of the SDN nodes to balance the load.

IV. Optimization Problem

The caching decision policy is modeled through an optimization problem, whose main settings and assumptions are reported below before formulating it. A list of symbols used in this paper is summarized in Table I.

A. Preliminaries

1) Edge nodes settings: We assume that the edge domain is characterized by N SDN nodes, which are linked in an arbitrary topology and are logically divided in the following subsets: *(i)* $I \subset N$ includes all the ingress nodes, receiving content requests from the consumers; *(ii)* $F = \{N - I\} \subset N$ includes all the intermediate nodes and the egress node linking the edge domain to the cloud. Each node $j \in N$ is equipped with a cache that can store up to S_j packets, and the overall storage capability of the edge domain is $S = \sum_{j \in N} S_j$.

TABLE I
Summary of the main notations.

Symbol	Description
C	catalog of IoT contents
c_k	generic content in the catalog
$T(c_k)$	lifetime of content c_k
$TS(c_k)$	generation timestamp of content c_k
$E[T^{res}(c_k)]$	average residual lifetime of content c_k
$B(c_k)$	size of content c_k, in terms of number of packets
d	remote cloud platform hosting the IoT contents
N	set of candidate cachers in the edge topology
j	candidate cacher node
S_j	number of packets that can be stored at edge node j
I	set of ingress nodes in the topology ($I \subset N$)
S	overall storage capacity in the domain
$X_j(c_k)$	binary variable taking the value 1 if c_k is available at node $j \in N \cup d$ and 0, otherwise
$G^L_{i,j}(c_k)$	latency gain for node i in retrieving content c_k from node j
$\lambda_i^{req}(c_k)$	request rate for content c_k at node $i \in N$
$\hat{T}_i^{req}(c_k)$	average request interval for content c_k at node $i \in N$
$T_{i,m}^{req}(c_k)$	request interval at interval $m \in M$ for content c_k at node $i \in N$
$R_{i,j}$	packet data rate between node i and j (in pkt/s)
$L_{i,j}(c_k)$	delay for retrieving content c_k at node i from node j
$RTT_{i,j}$	round trip time between node i and node j
$p^l_{i,j}$	packet loss probability over the link between node i and node j
$p^f(c_k)$	probability that the copy of content c_k cached at the edge is not outdated
$\lambda_j^{ref}(c_k)$	caching refreshing rate of content c_k at node j
$T_j^{ref}(c_k)$	time period between successive caching refreshing events for content c_k at node j
W	AIMD transmission window
β	AIMD multiplicative decrease factor
η	window increment per RTT

2) Content settings: We define the catalog of IoT contents as $C = \{c_1, ..., c_K\}$, where each content c_k is made of $B(c_k)$ packets. For the sake of simplicity, we assume that all content packets have the same size. We denote the lifetime and the generation timestamp of a content c_k as $T(c_k)$ and $TS(c_k)$, respectively. The same values are set in each of the $B(c_k)$ packets composing the content c_k. Every $T(c_k)$, a fresh sample of content c_k is stored in the remote cloud, in the following simply referred to as node d.

3) Content arrival rate: We define $\lambda_i^{req}(c_k)$ as the average arrival request rate for content c_k at ingress node $i \in I$. This rate can be practically derived by each ingress node by observing the inter-arrival times of the requests for that content. This will be described in Section VI-C.

4) Content retrieval latency: Several works in the literature, e.g., [6], [7], modelled the content retrieval latency in terms of the hop count between two end-points. However, this metric does not reflect the real latency over the considered path. Indeed, due to traffic congestion, two paths with the same number of hops can experience different latencies. This is why in our model we take into account the latency experienced by the requested content c_k over the path interconnecting data consumer and provider (i.e., an edge cacher or the remote

cloud). We assume that the Controller is able to estimate the latency, $L_{i,j}(c_k)$, experienced at ingress node i for retrieving content c_k from provider $j \in N \cup d$. The practical way this delay can be estimated is reported in Section VI-D.

5) Latency Gain: We define the latency gain $G^L_{i,j}(c_k)$ achieved if caching content c_k at node j as the saved retrieval delay by ingress node i to get c_k from node j rather than from the cloud d:

$$G^L_{i,j}(c_k) = \lambda^{req}_i(c_k) \cdot [L_{i,d}(c_k) - L_{i,j}(c_k)], \tag{1}$$

with $i \in I$, $j \in \{N \cup d\}$, and $L_{i,d}(c_k)$ the latency measured from ingress node i to retrieve content c_k from cloud d. When $j = d$, the latency gain is obviously zero, while if $j = i$, i.e., the cacher is the ingress node the consumer is connected to, then the gain is maximized. Intuitively, to be rapidly accessed by the consumers, all the contents should be cached at the ingress nodes. However, being the edge storage capacity limited and distributed among N nodes, the caching strategy has to select the most popular contents and the cacher that maximize the related latency gain.

6) Freshness probability: Let us define the probability, $p^f(c_k)$, that a content c_k cached at the edge is fresh, i.e., its lifetime $T(c_k)$ has not expired yet. In other words, the following condition holds for the residual lifetime of c_k, denoted as $T^{res}(c_k)$, measured at the current time instant t:

$$T^{res}(c_k) = TS(c_k) + T(c_k) - t > 0. \tag{2}$$

We assume that every time an edge node is selected as the cacher for content c_k, it stores a fresh copy of c_k until its lifetime expires. Then, if the node is still the current cacher (i.e., the Controller has not decided a new content placement) but the cached content has expired, whenever the cacher is reached by a new request for content c_k and a cache miss occurs, it retrieves the new freshest copy from the remote cloud. This happens with probability $(1 - p^f(c_k))$. Hence, $p^f(c_k)$ represents the probability that, when a new request for content c_k arrives, there is a fresh copy of it on the cacher. Therefore, similarly to [6], the expected caching refreshing rate $\lambda^{ref}(c_k)$ of content c_k can be derived as:

$$\lambda^{ref}(c_k) = (1 - p^f(c_k)) \cdot \lambda^{req}(c_k), \tag{3}$$

where $\lambda^{req}(c_k)$ is the arrival rate of requests for content c_k in the domain, and is obtained as the sum of the arrival rate for that content at all ingress nodes:

$$\lambda^{req}(c_k) = \sum_{i \in I} \lambda^{req}_i(c_k). \tag{4}$$

Indeed, since there is always only one cacher in the edge domain, all the incoming requests of c_k in the domain will be re-directed towards it.

Given the caching refreshing rate $\lambda^{ref}(c_k)$, we define $T^{ref}(c_k) = \frac{1}{\lambda^{ref}(c_k)}$ as the average time interval between two successive caching events for content c_k at the edge, as graphically sketched in Fig. 2. When a copy of c_k is updated on the cacher, i.e., a new content copy is retrieved from the cloud upon a request which cannot be locally satisfied with a fresh copy, it has an average residual lifetime denoted as $E[T^{res}(c_k)]$ (where $E[\cdot]$ represents the average operator).

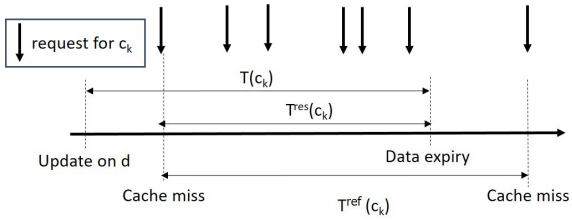


Fig. 2. Time period, $T^{ref}(c_k)$, between successive caching events for content c_k with lifetime $T(c_k)$.

In our model a cacher refreshes its copy of c_k only after the current copy is expired and a new request for c_k arrives. Hence, under the hypothesis at hand, $T^{ref}(c_k)$ is larger than $E[T^{res}(c_k)]$. In other words, the cached copy of c_k remains valid on average for $E[T^{res}(c_k)]$ seconds every $T^{ref}(c_k)$ seconds. Therefore, $p^f(c_k)$ represents the fraction of time that a fresh (not expired) content copy exists in the node's cache, and can be expressed as the ratio between the average residual lifetime of c_k over the refreshing period of c_k:

$$p^f(c_k) = \frac{E[T^{res}(c_k)]}{T^{ref}(c_k)} = E[T^{res}(c_k)]\lambda^{ref}(c_k). \tag{5}$$

$E[T^{res}(c_k)]$ is updated every time the cacher has a cache miss and retrieves the content from the cloud. In absence of historical values, $E[T^{res}(c_k)]$ is estimated as $T(c_k)/2$ under the assumption that the time between the refresh of content c_k at the remote cloud d and the request of a fresh content from the cacher j is uniformly distributed in $[0, T(c_k)]$.

By combining Eq. (3) and Eq. (5), the freshness probability can be derived as follows:

$$p^f(c_k) = \frac{E[T^{res}(c_k)]\lambda^{req}(c_k)}{1 + E[T^{res}(c_k)]\lambda^{req}(c_k)}. \tag{6}$$

B. Problem formulation

Given the system model, the caching strategy aims to maximize the latency gain defined in Eq. (1), by preferably caching at the edge those contents with a larger expected number of requests in their lifetime. The problem can be formulated as an Integer (binary, 0 or 1) LP problem, as follows:

$$\max \sum_{c_k \in C} p^f(c_k) \sum_{i \in I} \sum_{j \in N \cup d} G^L_{i,j}(c_k) X_j(c_k) \tag{7}$$

s.t.

$$\sum_{c_k \in C} B(c_k) X_j(c_k) \leq S_j, \ \forall j \in N \tag{8}$$

$$\sum_{j \in N \cup d} X_j(c_k) = 1, \ \forall i \in I, \ \forall c_k \in C \tag{9}$$

$$X_j(c_k) = \{0, 1\}, \forall j \in N \cup d, \forall c_k \in C \tag{10}$$

Constraint in Eq. (8) limits the placement of contents on edge nodes according to their available storage resources. Constraint in Eq. (9) states that content requests are satisfied by a single node: either a cacher in the local domain or the remote cloud; the position $X_d(c_k) = 1$ means that the content c_k is only

stored at the cloud d and not in the edge domain. Finally, the constraint in Eq. (10) restricts the decision variables, $X_j(c_k)$, to binary values. As a result, the strategy selects the contents that are expected to be more popular during their lifetime and places a copy of them at the cachers that guarantee the maximum latency gain. This also implies that the selected placement tries to reduce the overall content retrieval delay and copes against possible adverse effects of traffic congestion.

V. Proposed heuristic algorithm

The formulated ILP problem is equivalent to the maximization of a submodular function under a knapsack constraint. It can be proven that the objective function is sum of monotone nondecreasing submodular functions: *(i)* monotonicity is trivial: any new placement of a content cannot decrease the value of the objective function; *(ii)* submodularity applies since as the set of contents becomes larger, new contents are more likely cached to further edge nodes and ultimately, to the cloud, for which the gain will not increase at all. The problem is well known to be NP-hard, but can be approximated within the factor of (close to) $(1 - \frac{1}{e})$ through greedy algorithms [37]. Hence, we propose a heuristic algorithm that leverages a greedy approach to obtain sub-optimal solutions which are faster compared to the ILP, especially as the problem instances increase. Reasonably, given a set of requested contents C_r, the target of the heuristic is caching the requested contents $c_k \in C_r$ in the cachers $j \in N \cup d$ that provide the highest caching gain, with this latter depending on the freshness probability, the latency gain, and the content size, as follows:

$$G(c_k, j) = \sum_{i \in I} p^f(c_k) \cdot \frac{G^L_{i,j}(c_k)}{B(c_k)}. \tag{11}$$

The proposed heuristic algorithm proceeds iteratively. It starts by considering the sets of requested contents and of potential cachers, and calculates the caching gain of placing each $c_k \in C_r$ in each $j \in N \cup d$ according to Eq. (11). For each $c_k \in C_r$, the "gain and cacher" pairs are included in ascending order into the list of best cachers for c_k, BC_{c_k}, i.e., the first element of the list identifies the best cacher and the corresponding gain, the second element identifies the second best cacher and the corresponding gain, etc. At each iteration, the algorithm selects an edge node j and tries to allocate there the contents that have identified j as the best cacher and have the higher gains. Due to storage constraints, not all the contents could be accommodated in j and other allocation options must be considered according to the information in BC_{c_k}.

As summarized in Algorithm 1, the caching allocation problem is solved with successive iterations by using as input, at each step, the set of contents that are still to be allocated $\hat{C}$, with $\hat{C} \subseteq C_r$, and the set of nodes that still have free storage space, $\hat{N} \subseteq N \cup d$. At the first iteration, $\hat{C} = C_r$ and $\hat{N} = N \cup d$. As shown from line 14 of Algorithm 1, given a node $j \in \hat{N}$, the algorithm creates the list CL_j, which includes the contents that have selected j as their first best cacher and it is sorted in terms of decreasing gain. The contents in CL_j are allocated in j and removed from the list until the available storage capacity of the node is filled, i.e., $s_j = 0$, or CL_j is empty. In the first case, j is removed from $\hat{N}$ and from the lists of best cachers. Also, if some of the contents have not been allocated on j due to the storage space constraint, then the algorithm repeats the process to find the subsequent best cacher for them, if available. The loop continues either until all the contents are allocated or there is no more edge cacher available. Possibly remaining contents are left in the cloud.

If we consider an efficient sorting algorithm like QuickSort, Algorithm 1 complexity is bounded by $O(|N| \cdot |C_r| \cdot \log |C_r|)$.

VI. Practical considerations

In the following we describe how the involved parameters can be practically estimated as well as the devised SDN workflows and monitoring routines.

A. Content placement and workflows

We assume that the Controller stores information about the transient IoT contents, requested and cached in its domain, in a new data structure that we call Content Information Base (CIB) (Fig. 3). For each requested content c_k, the CIB tracks the relevant parameters, e.g., its lifetime $T(c_k)$, its request arrival rate $\lambda^{req}(c_k)$, the number of packets $B(c_k)$ composing it, the timestamp $TS(c_k)$ and the cacher, if available.

The content placement decision is *not taken upon each content request*. Instead, in order to make the deployment more viable, it is *periodically* taken by the Controller implementing the heuristic described in the previous Section. Such a choice is meant to track possible changes in the content popularity profiles, i.e., in the pattern of content request arrivals at the ingress nodes of the edge domain. When this pattern varies, the set of contents to be cached at the edge may vary as well. Therefore, the caching decision period is an important parameter to set. After analyzing an IoT dataset and queries from search engines, the authors in [20] have found that the IoT popularity follows *the Zipf's law [38] with a skewness parameter that can change on a regular basis.* Such variations are due to the fact that the requests for IoT data are related to the people's daily life and needs. The authors demonstrate that a period of 60 minutes well captures the popularity variation of IoT contents and select this value in their deployment. A similar design choice has been also applied to vehicular traffic information in [39]. In agreement with the aforecited works, in our design, we select a caching decision period of 60 minutes.

Whenever the content placement algorithm is executed, an OF PACKET_OUT message is used by the Controller to trigger caching actions in SDN nodes. In particular, a newly defined action type, named *caching*, is injected through the *Experimenter instruction* field [28] into the selected cachers. When an SDN ingress node receives a content request from a consumer, it extracts the application-level name carried in the packet's header, and checks whether a content with that name is locally stored in its cache. If it is the case, then it replies with the cached content. Otherwise, it checks whether a route exists in its flow table towards a cacher for that content. Matching on the content name can be performed thanks to the use of OF eXtensible Match (OXM) [28], included in

Algorithm 1: Heuristic algorithm

input : Set of candidate cachers $N \cup d$;
set of requested contents C_r;
storage capability S_j $\forall j \in N \cup d$;
currently available storage s_j $\forall j \in N \cup d$
output: $X_j(c_k)$ $\forall j \in N \cup d$, $\forall c_k \in C_r$

for ***all*** $j \in N \cup d$ **do**
  for ***all*** $c_k \in C_r$ **do**
    insert $[j, G(c_k, j)]$ in BC_{c_k};
    $X_j(c_k) \longleftarrow 0$;
  end
  $s_j \longleftarrow S_j$;
end
for ***all*** $c_k \in C_r$ **do**
  sort(BC_{c_k}, according to $G(c_k, j)$, non increasing);
end
$\hat{C} \longleftarrow C_r$;
$\hat{N} \longleftarrow N \cup d$;
repeat
  for ***all*** $j \in \hat{N}$ **do**
    for ***all*** $c_k \in \hat{C}$ **do**
      if *the* 1^{st} *item of* BC_{c_k} *contains* j **then**
        insert $[c_k, G(c_k, j)]$ in CL_j
      end
    end
    sort(CL_j, according to $G(c_k, j)$, non increasing);
    while $s_j > 0$ *and* $CL_j \neq \emptyset$ **do**
      $[c_k, G(c_k, j)] \longleftarrow$ ***read***(CL_j,first element);
      if $B(c_k) \leq s_j$ **then**
        $X_j(c_k) \longleftarrow 1$;
        $s_j \longleftarrow s_j - B(c_k)$;
        purge c_k from CL_j;
        purge c_k from $\hat{C}$;
      end
    end
    if $s_j == 0$ **then**
      $\hat{N} \longleftarrow \hat{N} \setminus j$;
      for ***all*** $c_k \in \hat{C}$ **do**
        purge item containing j from BC_{c_k};
      end
    end
    if $CL_j \neq \emptyset$ **then**
      for ***all*** $c_k \in CL_j$ **do**
        purge item containing j from BC_{c_k};
      end
      break
    end
  end
until $\hat{N} == \emptyset$ *or* $\hat{C} == \emptyset$;
return $X_j(c_k)$ $\forall j \in N \cup d$, $\forall c_k \in C_r$

OpenFlow implementations starting from version 1.2. OXM enables the matching over any message header's fields not defined in the OF specifications.

TABLE II
IMPACT OF CACHING DECISION PERIODICITY ON CONTROLLER-TO-NODE SIGNALING.

Periodicity	$\gamma = 0.01$		$\gamma = 0.1$	
	N**=30**	N**=90**	N**=30**	N**=90**
5 minutes	125.86 bps	164.26 bps	1.085 kbps	1.124 kbps
10 minutes	62.93 bps	82.13 bps	542.5 bps	561.96 bps
30 minutes	20.97 bps	27.38 bps	180.83 bbs	187.32 bps
60 minutes	10.48 bps	13.69 bps	90.416 bps	93.66 bps

In case no entry exists in the flow table that matches the incoming packet, the ingress node contacts the Controller by sending an OF PACKET_IN message that encapsulates the request packet's header conveying the content name. Upon receiving it, the Controller looks into the CIB to identify the cacher for that content in its edge domain. If the requested content was selected to be stored by the periodically run heuristic, then a cacher is found. In such a case, after accessing the NIB, it sets up the path from the ingress node to the cacher, i.e., a forwarding action is injected at the on-path nodes, through legacy OF FLOW_MOD messages. Conversely, if the cacher is not available, a fresh copy of the requested content must be retrieved from the cloud. In this case, the Controller uses OF FLOW_MOD messages to set up the path from the ingress node to the cloud, unless this path is already available as a default route in the flow tables of edge nodes. Fig. 4 depicts the aforementioned workflow.

For the sake of completeness, in Table II, we report the signalling overhead in an edge domain of N cachers ($N = 30$ and 90) for an average content name length, denoted as $\bar{L}$, of 40 bytes [40] and a catalog size $|C|$ equal to 10^4, when varying the ratio of the edge caching capacity over the content catalog size, denoted as γ (0.01 and 0.1). We consider the worst case condition by assuming that caching instructions are sent, in an OF PACKET_OUT message, to all SDN nodes for all the contents (identified through a $\bar{L}$ bytes-long name) that can be stored in the edge domain, which is equal to $\gamma \cdot |C|$. As expected, the overhead increases with the edge domain size and as the periodicity of the caching decision decreases. Such a result confirms that the chosen periodicity of 60 minutes incurs a low signaling overhead, as a further beneficial effect.

B. OF monitoring routines and data structures

Topology and interfaces status. The Controller maintains a centralized view of the network by tracking the graph of the edge topology in the NIB, as in legacy SDN deployments. Per-port statistics of each SDN node are periodically reported to the Controller by leveraging legacy OF messages. Starting from this information, the Controller measures the packet loss probability and the link delay on each node interface [41], which allow to estimate the content retrieval latency.

Caching capabilities and content statistics. We assume that each SDN node j, when bootstrapping, notifies the Controller of its *nominal caching capability* S_j through the $storage$ field that we propose to add to the legacy OF FEATURES reply message. This OF message typically carries, upon request by the SDN Controller, the basic capabilities of a switch, such as the number of flow tables' entries, the switch's

buffer size, etc. [28]. We assume that the Controller fills in its CIB by getting information about the IoT content requests entering its domain through the OF FLOW message, which normally retrieves flow entry metrics from an SDN node [28]. In particular, we propose that an ingress node measures the request arrival rate for each content, and periodically reports it to the Controller via the cited OF FLOW message. This is possible thanks to the OF *Experimenter extension* fields [28]. Then, the Controller derives the arrival rate of content c_k in its edge domain by summing up the request arrival rates for that content tracked at all ingress nodes (see Eq. (4)).

C. Content request arrival at ingress node

The request arrival rate for content c_k at the ingress node i can be computed by considering the average interarrival time of the $M+1$ most recent content requests[2] at node i, as explained in the following.

Let us consider a set of request arrival events for content c_k at node i, $r_{i,1}(c_k), r_{i,2}(c_k), ..., r_{i,M+1}(c_k)$, respectively at times $t_1, t_2, ..., t_{M+1}$, where $r_{i,1}(c_k)$ is the most recent arrival, and $r_{i,M+1}(c_k)$ is the oldest one. As shown in Fig. 5, we refer to $T_{i,1}^{req}(c_k)$ as the time interval between the two most recent request arrival times for c_k at node i, t_2 and t_1, to $T_{i,2}^{req}(c_k)$ as the time interval between t_3 and t_2, etc. We also consider $T_{i,0}^{req}(c_k)$ as the time interval between the instant in which the most recent request for c_k arrives at node i and the current instant, t, in which the estimation of the average arrival request rate for c_k is performed, $T_{i,0}^{req}(c_k) = t - t_1$. Value $T_{i,0}^{req}(c_k)$ has to be ignored in the calculation unless it is large enough that including it would increase the average, as in [42].

We define $\hat{T}_i^{req}(c_k)$ as the average time between successive requests for c_k, weighted over the last M time intervals, $T_{i,m}^{req}(c_k)$, at node i:

$$\hat{T}_i^{req}(c_k) = max\left\{\frac{\sum_{m=1}^{M} T_{i,m}^{req}(c_k)}{M}, \frac{\sum_{m=0}^{M-1} T_{i,m}^{req}(c_k)}{M}\right\}. \quad (12)$$

The average arrival request rate for c_k at node i is finally computed as:

$$\lambda_i^{req}(c_k) = \frac{1}{\hat{T}_i^{req}(c_k)}. \quad (13)$$

D. Content retrieval latency

For the sake of simplicity, we do not distinguish the consumer from the ingress node it is attached to, and we denote the round-trip-time (RTT) between an ingress node $i \in I$ and a provider $j \in N \cup d$ as $RTT_{i,j}$. The time needed by the ingress node i to fetch content c_k from a provider j, $L_{i,j}(c_k)$, can be computed by the Controller assuming stationary network conditions, i.e., stationary $RTT_{i,j}$ and packet loss probability $p_{i,j}^l$, whose values are tracked by the Controller, as mentioned before. More specifically, the packet data rate for a generic content retrieval over the path between nodes i and j, $R_{i,j}(t)$,

[2]After a tuning study not reported because of the limited relevance, we set $M = 20$, which guarantees a good accuracy of the estimation.

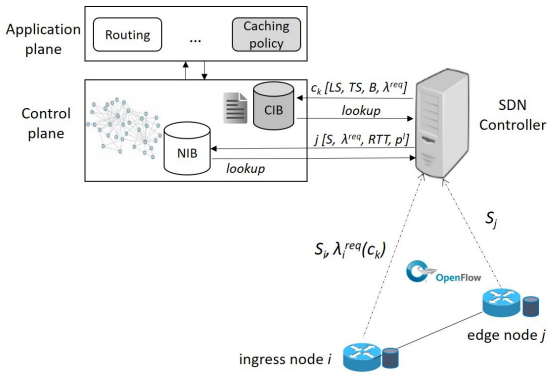


Fig. 3. Newly monitored parameters and added data structures (in grey).

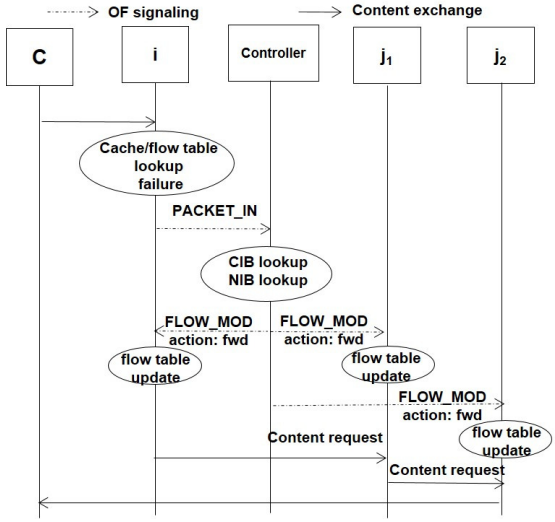


Fig. 4. Workflow for a content request issued by consumer C attached to ingress node i and satisfied by edge node j_2, after that forwarding actions are injected by the Controller.

is estimated by considering a Transmission Control Protocol (TCP)-like Additive Increasing Multiplicative Decreasing (AIMD) algorithm. Although IoT traffic is generally carried over User Datagram Protocol (UDP), here we are considering the transfer of IoT data packets from an edge cacher (or the cloud) to an ingress node, so TCP is a more suited solution in order to guarantee transfer reliability. During the TCP steady state, the provider sends W consecutive data packets, where W is the size (in packets) of the TCP transmission window under stationary conditions. If all the packets are received without any loss and successfully acknowledged, then W is increased

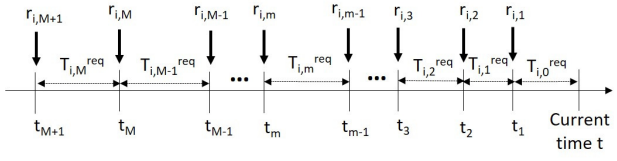


Fig. 5. Interarrival times, $T_{i,m}^{req}(c_k)$, of requests for a given content c_k, contributing to the estimation of $\hat{T}_i^{req}(c_k)$.

TABLE III
MAIN SIMULATION SETTINGS.

Parameter	Value
Number of SDN edge nodes	• 29 (Scenario A) • 91 (Scenarios B, C)
Number of consumers	• 300 (Scenario A) • 4000 (Scenarios B, C)
Content request distribution	Poisson (Scenarios A, B, C)
Content request arrival rate	• 0.5 req/s (Scenarios A, B) • [0.2 0.4 0.6 0.8 1] req/s (Scenario C)
Content catalog size	10^6
Content size (in packets)	[5-1000]
Packet size (in bytes)	1000
Short-lived content lifetime	exponentially distributed with mean 1s
Long-lived content lifetime	exponentially distributed with mean 100s
Content popularity	• Zipf, α=0.8 (Scenarios A; B; C) • uniform distribution (Scenario B)
Edge caching capacity	3%-15% of the content catalog size

of $\eta \geq 1$ packets, otherwise for a packet loss probability $p^l_{i,j} > 0$, then the window size is decreased of a factor β, with $0 \leq \beta < 1$. The factors β and η are related to the packet data rate $R_{i,j}(t)$ according to the following expression, derived from previous works on TCP in [43]:

$$\frac{dR_{i,j}(t)}{dt} = \frac{\eta}{RTT^2_{i,j}} - \beta \cdot R_{i,j}(t) \cdot p^l_{i,j}. \tag{14}$$

Typical values for the AIMD increasing and decreasing factors are, respectively, $\eta = 1pkt/RTT$ and $\beta = 0.5$. Eq. (14) holds under the assumption that $R_{i,j}(t)$ represents a marginal contribution (i.e., it is the rate of a single TPC-like data flow) to the overall traffic exchanged over the path connecting i and j. As a consequence, $RTT_{i,j}$ and $p^l_{i,j}$ are not influenced by $R_{i,j}(t)$. Hence, the latency experienced at ingress node i for retrieving content c_k from a provider $j \in N \cup d$ can be estimated as:

$$L_{i,j}(c_k) = \begin{cases} 0 & if \quad j = i \\ \frac{B(c_k)}{E[R_{i,j}(t)]} & if \quad j \neq i \end{cases} \tag{15}$$

When j coincides with the remote cloud, d, the latency $L_{i,d}(c_k)$ can be estimated in a similar way, with the difference that eastbound/westbound interfaces may be leveraged by the Controller to get information about the latency contributions over the path to the cloud, likely crossing other network domains.

VII. PERFORMANCE EVALUATION

A. Parameter settings

The performances of the proposed policy, when compared with two benchmark solutions, have been derived under the settings detailed below and summarized in Table III.

1) Content settings: To cover a variety of monitoring application scenarios, we consider a catalog of 10^6 transient contents which can be classified into two distinct sets depending on their lifetime values, whose average is exponentially distributed, similarly to [44]–[46]. The first set, denoted as C_S, includes IoT contents we refer to as *short-lived*, with a mean lifetime equal to 1s, which resemble, for instance, body parameters, like heart rate and blood pressure [47]. The second set, denoted as C_L, includes IoT contents we refer to as *long-lived*, with a mean lifetime equal to 100s, which resemble, for instance, parameters (e.g., temperature, moisture, load) requested by less critical applications like environmental monitoring [33] and smart grid systems [48]. The fraction of contents belonging to C_L is a varying parameter in our simulations, denoted as C_L/C.

Content request arrivals follow a Poisson process, with exponentially distributed inter-arrival times. Popularity follows a *Zipf distribution* with skewness parameter α set to 0.8. This setting is coherent with the study in [20], which analyzes IoT dataset and queries from search engines proving that the content popularity follows a Zipf distribution. A *uniform distribution* is also considered to model the case where requests are not concentrated on a few contents, which may resemble some IoT contexts, where all sensors have close probabilities of being solicited, according to [49].

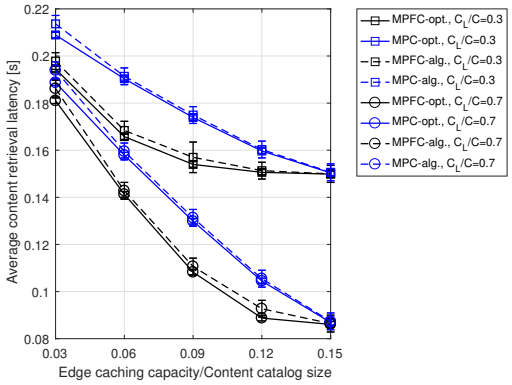
2) Network topologies and content request rates.: We refer to three different scenarios:

- *Scenario A:* it consists of a medium size topology with 29 SDN edge nodes organized as illustrated in Fig. 1, where 4 upper-layer nodes, interconnected in a full meshed topology, are the root of a three-layered fat tree. Leaf nodes at the bottom act as ingress nodes which consumers are connected to. The node in the middle of the mesh topology acts as the egress node, connecting the edge domain to the cloud. Such nodes handle 300 consumers with each consumer requesting a content, according to a Zipf distribution and with an arrival rate which is Poisson-distributed with average 0.5 requests/s.
- *Scenario B:* it foresees a large topology with 91 nodes, organized as a central ring with 6 backbone routers connected through a four-layered fat tree topology to leaf nodes, which are the access routers which 4000 consumers are connected to. Each consumer requests a content with an arrival rate which is Poisson-distributed with average 0.5 requests/s. We consider both a Zipf and a uniform content request distribution.
- *Scenario C:* the same topology as in Scenario B is considered. Each consumer requests a content according to a Zipf distribution and with an arrival rate which is Poisson-distributed and varied in the range [0.2, 0.4, 0.6, 0.8, 1] requests/s [6].

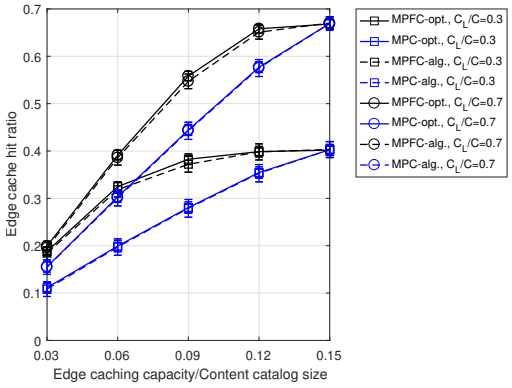
3) Storage settings: For both scenarios, we assume the total caching capacity of the edge domain to be a varying fraction of the content catalog size, according to the study in [50]. This capacity is assumed to be uniformly distributed among *all* SDN nodes in the domain.

B. Benchmark schemes and metrics

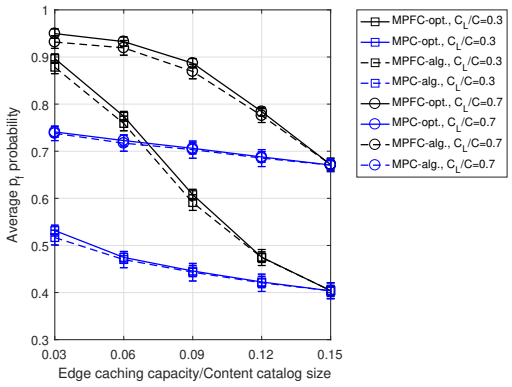
We compare the proposed model against two benchmark solutions. The first one, named here as Most Popular Contents (MPC), is representative of the SDN-based proposals that use the content popularity as the decision criterion for caching at the edge, such as the one in [7]. It jointly optimizes the content retrieval delay and the amount of traffic crossing the domain to reach the cloud/cacher, while satisfying the storage capacity constraints of each edge node. Similarly to our proposal, this objective is achieved by maximizing the



(a) Average content retrieval latency.



(b) Edge cache hit ratio.



(c) Average freshness probability.

Fig. 6. Scenario A: the proposed solution MPFC against a baseline MPC, both when solving the optimization problem (opt.) and the heuristic (alg.): when varying the caching capacity of the domain and the fraction of long-lived contents (Zipf distribution, α=0.8).

latency gain, multiplied by the content request arrival rate. For the sake of a fair comparison, unlike the work in [7], the considered implementation of this benchmark solution models the latency between a consumer and the cacher/cloud not in terms of number of hops, but by considering the actual retrieval latency over each link along the consumer-cacher/cloud path. Unlike our model, the caching decision is completely oblivious of the transient nature of IoT contents.

The second benchmark is a baseline approach according to which IoT contents are only stored into the remote cloud.

The following metrics are derived with results averaged over 100 runs and reported with 99% confidence intervals.

- *Average content retrieval latency*: it is the average time experienced by the consumers to retrieve the requested contents from the providers (either a cacher in the edge domain or the remote cloud).
- *Edge cache hit ratio*: it is the fraction of requests which are satisfied with a fresh cached content provided by edge nodes, rather than from the remote cloud.
- *Average content freshness probability*: it measures the average freshness probability at the edge.
- *Edge-to-cloud traffic*: it is derived as the rate of packets that are retrieved from the cloud to satisfy the content requests, either because a content is not stored in the edge or because a previously cached content has expired.

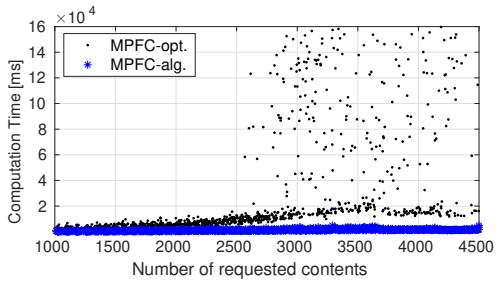
C. Heuristic algorithm Vs. Optimization model: Scenario A

The conducted study is aimed to understand to which extent the proposed caching policy to be deployed by the SDN Controller is able to capture the targeted objectives. Moreover, we evaluate the performance of our proposed heuristic (dashed curves in the plots), by comparing it with the optimal solution of the ILP model formulated in Eq. (7). The latter one is solved with *intlinprog*, the Mixed-integer linear programming (MILP) solver available in the Matlab® optimization toolbox (solid curves in the plots). The heuristic algorithm and the optimal solution have been compared when considering both the proposed policy (labeled as Most Popular Fresh Contents, MPFC, in the plots, black curves) and the MPC benchmark scheme (in the plots, blue curves), when varying the caching capacity of the edge domain, for different fractions of long-lived contents, C_L/C, when considering a Zipf content popularity distribution.

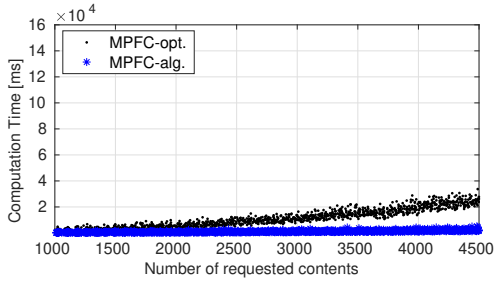
First of all, we observe that the heuristic well approximates the optimal solution for all metrics and under all the considered settings. Fig. 6(a) shows that the content retrieval latency reasonably decreases as the caching capacity of the domain increases, for both MPFC and MPC. This is because there is a higher chance for a consumer to retrieve the requested content from a close edge node caching it, instead that from the remote cloud. For instance, the latency values of MPFC, for a C_L percentage of 70%, pass from 0.19s to 0.09s, when increasing the storage capacity from 3% to 15% of the catalog size. Increasing further the edge caching capacity does not achieve further latency reduction. This holds for both MPFC and MPC. Indeed, with a Zipf parameter of 0.8, many requests are concentrated to a few popular contents, which can be accommodated in the edge nodes' caches.

Differences in the retrieval latency between the two compared schemes, MPC and MPFC, are more remarkable when the caching capacity is below 10% of the catalog size. This is a significant finding since the edge caching capacity is expected to be significantly lower than the content catalog size [50]. With less caching capacity available, the competition among contents is exacerbated, and the correct choice of the contents to cache at the edge becomes more critical. Moreover, the larger the fraction of long-lived contents in the domain the shorter the retrieval latency. Such a trend reflects the fact that edge nodes more likely cache contents

with a longer residual lifetime that would expire less quickly, hence causing more cache hit events, as shown in Fig. 6(b). Similar observations regarding the comparison between MPFC and MPC hold for the edge cache hit ratio metric. Fig. 6(c) reports the average content freshness probability. As expected, the MPFC proposal exhibits higher freshness probability than the benchmark MPC solution. This means that it is more effective in caching contents with a longer lifetime at the edge domain. The benchmark scheme is agnostic about the content freshness, hence it cannot ensure that those contents with a longer lifetime are preferably cached. For both schemes, not surprisingly, the freshness probability gets higher as the fraction of long-lived contents increases.



(a) Edge caching capacity/Content catalog size=0.03



(b) Edge caching capacity/Content catalog size=0.09

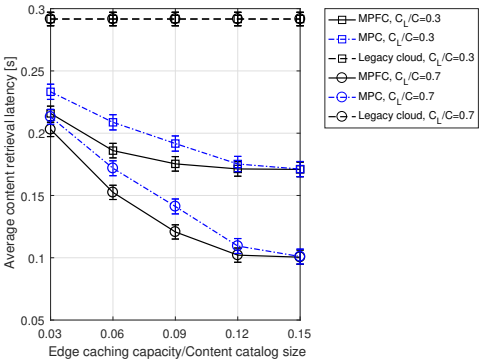
Fig. 7. Computation time of the optimization model (opt.) and of the proposed heuristic (alg.) when varying the number of requested contents (Scenario A).

D. Computation time

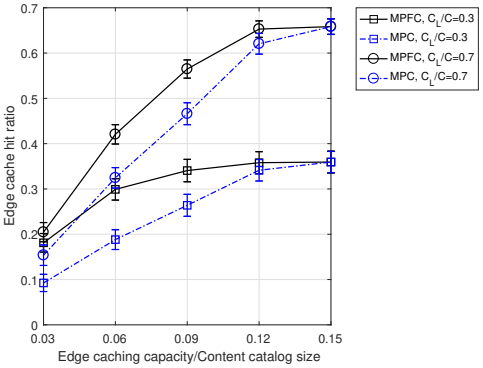
We measure the computation time for solving our formulated ILP problem, in Eq. (7), on a single Intel® Xeon® CPU core with 1.8 GHz as reported in the scatterplot in Fig. 7. We consider the same settings as in Scenario A.

Being a combinatorial problem, the numbers of variables and constraints affect the computation time. Hence, we generate different problem instances of varying sizes in terms of *(i)* ratio between the edge storage capacity and the content catalog size and *(ii)* number of contents requested by the consumers. The former parameter resembles the capacity of the knapsacks in the considered ILP problem. The latter parameter represents the number of contents belonging to the catalog that are candidates to be cached at the edge, i.e., the items to be packed in the knapsacks of the considered ILP problem.

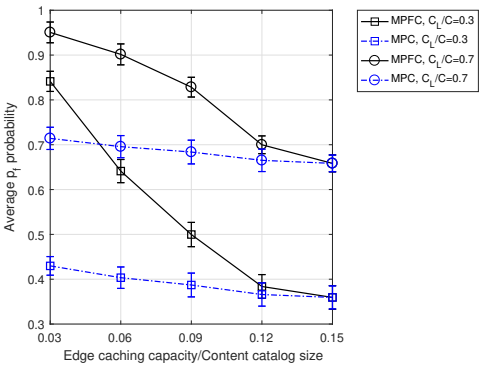
The computation time for the standard optimization problem solver is larger as the number of requested contents increases: in the order of tens of seconds, with higher values when the edge caching capacity over the content catalog size is 0.03.



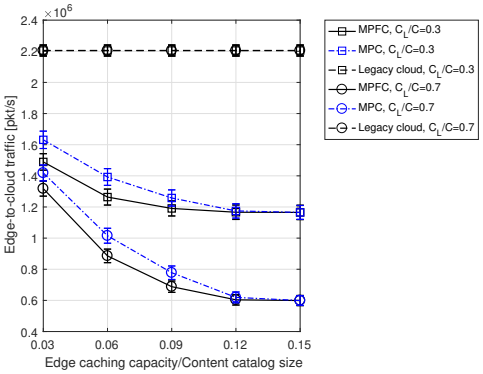
(a) Average content retrieval latency.



(b) Edge cache hit ratio.

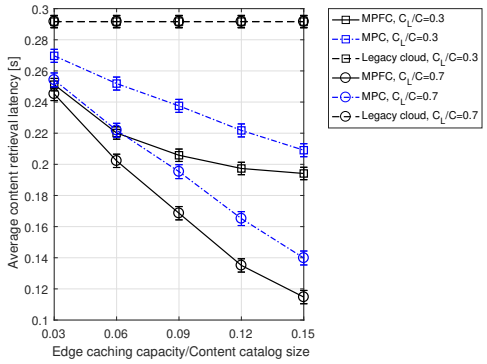


(c) Average freshness probability.

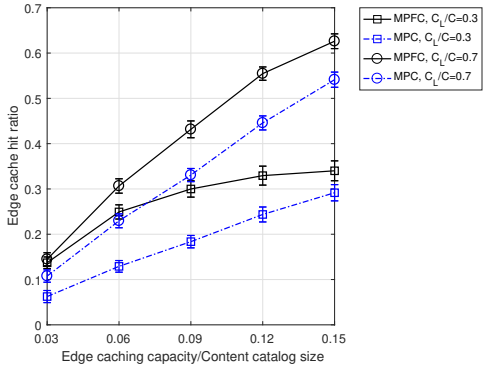


(d) Edge-to-cloud-traffic.

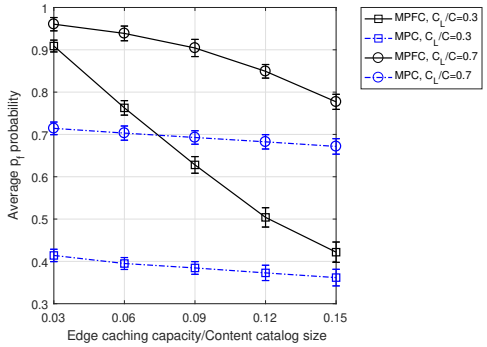
Fig. 8. Scenario B - the proposed heuristic MPFC against the benchmark schemes: MPC and the legacy cloud solution, when varying the caching capacity of the edge domain and the fraction of long-lived contents (Zipf distribution, α=0.8).



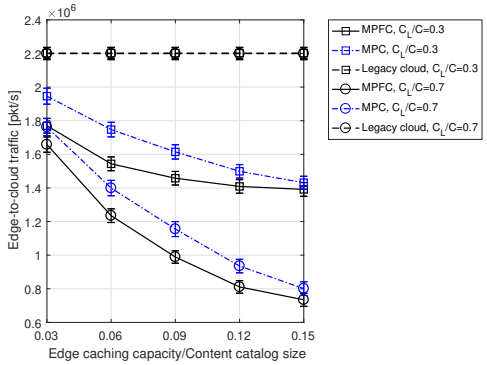
(a) Average content retrieval latency.



(b) Edge cache hit ratio.



(c) Average freshness probability.



(d) Edge-to-cloud-traffic.

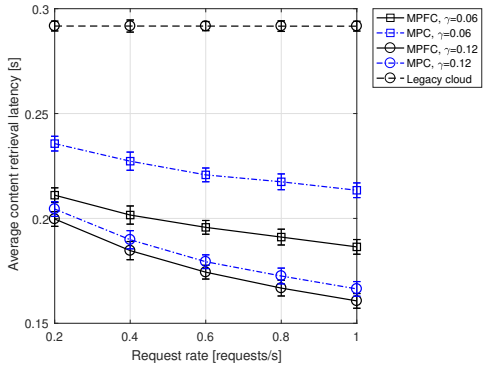
Fig. 9. Scenario B - the proposed heuristic MPFC against the benchmark schemes: MPC and the legacy cloud solution, when varying the caching capacity of the edge domain and the fraction of long-lived contents (uniform distribution).

Values are highly spread in the latter case since it is more challenging to find an optimal solution, and some problem instances may entail the exploration of several solutions. On the other hand, the proposed heuristic scales well with the number of requested contents. Measured values are in the order of a few seconds in the worst case. For the sake of completeness, values up to hundreds of seconds are achieved instead in Scenario B, although results are not reported due to length constraints. Such results confirm the practicality of our proposal: the SDN Controller, expected to be equipped with more powerful processing capabilities than those considered for the experiments, can *periodically* execute the content placement heuristic in acceptable times.

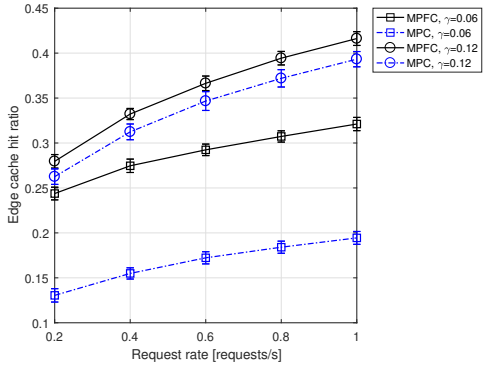
E. Scenario B

Achieved trends in Scenario B are similar to those measured for Scenario A. Fig. 8(a) shows that the content retrieval latency decreases as the caching capacity of the domain increases, for both MPFC and MPC. The cache hit ratio, instead, increases as the capacity increases (Fig. 8(b)). Such trends have to be ascribed to the fact that a consumer has a higher chance to retrieve the requested content from a close edge node caching it, instead that from the remote cloud. Both caching schemes exhibit a significantly lower latency compared to the legacy cloud solution. Similarly to the medium size topology, the differences in the content retrieval latency between the two compared caching schemes are more remarkable when the caching capacity is below 10% of the catalog size. Furthermore, the larger the fraction of long-lived contents in the domain the better the performance. Being more effective in caching contents with a longer lifetime at the edge domain, the MPFC proposal achieves higher freshness probability than the MPC benchmark solution (Fig. 8(c)). Fig. 8(d) shows that MPFC incurs a significantly lower amount of traffic exchange with the cloud compared to the legacy cloud solution. Moreover, it is lighter than the MPC benchmark.

Fig. 9 reports the same metrics discussed above, when considering the content popularity to be uniformly distributed. Compared to previous results, it can be observed that the content retrieval latency is slightly higher (Fig. 9(a)) and the edge cache hit ratio slightly lower (Fig. 9(b)). With uniform content requests distribution, in fact, more requests can be issued for distinct contents, and there is less chance to serve multiple requests by caching only a few contents at the edge. Such a trend also translates in a higher amount of traffic exiting the edge domain, as shown in Fig. 9(d), compared to the Zipf-distributed content requests pattern. It is worth observing that for MPFC the content freshness probability is higher in the uniform case than in the Zipf case (Fig. 9(c)). Indeed, being the popularity of requests uniformly distributed, the only factor deeply influencing the decision in MPFC is the content lifetime. Improvements of the MPFC compared to the MPC benchmark scheme are also more remarkable. Such a trend is ascribed to the fact that contents with a higher freshness probability are preferably stored in MPFC, while MPC is oblivious of the IoT contents' lifetime. As a result, MPFC proves to be effective even when the request pattern does not follow a skewed popularity distribution, a case that could happen in IoT environments.



(a) Average content retrieval latency.



(b) Edge cache hit ratio.

Fig. 10. Scenario C - the proposed heuristic MPFC against the benchmark schemes: MPC and the legacy cloud solution, when varying the request arrival rate per consumer and the caching capacity of the edge domain (Zipf distribution with α=0.8, C_L/C=0.3).

F. Scenario C

The last simulation campaign analyzes the impact of the request rate, for a fixed fraction of long-lived contents, C_L/C=0.3. Results in Fig. 10 show that MPFC outperforms the considered benchmark schemes under all the considered settings, with more remarkable improvements under the most challenging condition of edge caching capacity over catalog size (γ) equal to 0.06. In particular, results show that, as the request arrival rate increases, the content retrieval latency decreases (Fig. 10(a)) and the edge cache hit ratio increases (Fig. 10(b)). This is because, due to the Zipf distribution, a higher number of requests concentrate on the same few contents, the most popular ones, which are the most likely to be cached in the edge domain and, hence, are retrieved with a shorter latency.

VIII. Conclusion and Future Work

In this paper we have proposed a novel caching policy that identifies which transient IoT contents need to be cached and in which edge node of an SDN-based edge deployment. A heuristic is designed which well and efficiently approximates the solution of the formulated ILP problem. Furthermore, the conducted evaluation proves the superiority of the proposal against a baseline approach, oblivious of the freshness of contents. For the sake of completeness, the improvements w.r.t. a cloud solution are also quantified. Differences in the performance get more remarkable under smaller caching capacity values of the edge domain, mimicking realistic and more challenging settings, under which competition among contents to be cached becomes more significant.

As future work, we plan to further optimize the conceived caching strategy by trading-off between two contradictory targets, i.e., content diversity and caching efficiency, for instance by optimally replicating some highly requested content copies in multiple edge nodes and possibly closer to the consumers.

Moreover, since edge infrastructures are evolving towards distributed in-network computing systems, it would be interesting to extend the functionalities at the Controller to support the joint orchestration of caching and processing services in presence of IoT transient contents.

References

[1] X. Xu, S. Fu, L. Qi, X. Zhang, Q. Liu, Q. He, and S. Li, "An IoT-oriented Data Placement Method with Privacy Preservation in Cloud Environment," *Journal of Network and Computer Applications*, vol. 124, pp. 148–157, 2018.

[2] M. Chiang and T. Zhang, "Fog and IoT: An overview of research opportunities," *IEEE IoT Journal*, vol. 3, no. 6, pp. 854–864, 2016.

[3] M. Amadeo, G. Ruggeri, C. Campolo, and A. Molinaro, "IoT Services Allocation at the Edge via Named Data Networking: from Optimal Bounds to Practical Design," *IEEE Transactions on Network and Service Management*, vol. 16, no. 2, pp. 661–674, 2019.

[4] M. I. Naas, P. R. Parvedy, J. Boukhobza, and L. Lemarchand, "iFogStor: an IoT data placement strategy for fog infrastructure," in *IEEE ICFEC*, 2017, pp. 97–104.

[5] H. Zhu, Y. Cao, X. Wei, W. Wang, T. Jiang, and S. Jin, "Caching transient data for internet of things: A deep reinforcement learning approach," *IEEE IoT Journal*, vol. 6, no. 2, pp. 2074–2083, 2018.

[6] S. Vural, N. Wang, P. Navaratnam, and R. Tafazolli, "Caching transient data in internet content routers," *IEEE/ACM Transactions on Networking*, vol. 25, no. 2, pp. 1048–1061, 2016.

[7] Y. Cui, J. Song, M. Li, Q. Ren, Y. Zhang, and X. Cai, "SDN-based big data caching in ISP networks," *IEEE Transactions on Big Data*, vol. 4, no. 3, pp. 356–367, 2017.

[8] A. C. Baktir, A. Ozgovde, and C. Ersoy, "How can edge computing benefit from software-defined networking: a survey, use cases, and future directions," *IEEE Comm. Surveys & Tutorials*, vol. 19, no. 4, pp. 2359–2391, 2017.

[9] A. Wang, Z. Zha, Y. Guo, and S. Chen, "Software-defined networking enhanced edge computing: A network-centric survey," *Proceedings of the IEEE*, vol. 107, no. 8, pp. 1500–1519, 2019.

[10] H.-L. Truong and S. Dustdar, "Principles for Engineering IoT Cloud Systems," *IEEE Cloud Computing*, vol. 2, no. 2, pp. 68–76, 2015.

[11] S. Bakiras and T. Loukopoulos, "Combining Replica Placement and Caching Techniques in Content Distribution Networks," *Computer Communications*, vol. 28, no. 9, pp. 1062–1073, 2005.

[12] M. Hu, J. Luo, Y. Wang, and B. Veeravalli, "Practical Resource Provisioning and Caching with Dynamic Resilience for Cloud-based Content Distribution Networks," *IEEE Transactions on Parallel and Distributed Systems*, vol. 25, no. 8, pp. 2169–2179, 2013.

[13] Y. Mansouri, A. N. Toosi, and R. Buyya, "Cost Optimization for Dynamic Replication and Migration of Data in Cloud Data Centers," *IEEE Transactions on Cloud Computing*, 2017.

[14] G. Xylomenos *et al.*, "A survey of information-centric networking research," *IEEE Comm. Surveys & Tutorials*, vol. 16, no. 2, pp. 1024–1049, 2014.

[15] J. Wang, "A survey of web caching schemes for the internet," *ACM SIGCOMM Computer Comm. Review*, vol. 29, no. 5, pp. 36–46, 1999.

[16] E. Bastug, M. Bennis, and M. Debbah, "Living on the Edge: the Role of Proactive Caching in 5G Wireless Networks," *IEEE Communications Magazine*, vol. 52, no. 8, pp. 82–89, 2014.

[17] A. Aral and T. Ovatman, "A Decentralized Replica Placement Algorithm for Edge Computing," *IEEE Transactions on Network and Service Management*, vol. 15, no. 2, pp. 516–529, 2018.

[18] M. Zhang, H. Luo, and H. Zhang, "A survey of caching mechanisms in information-centric networking," *IEEE Communications Surveys & Tutorials*, vol. 17, no. 3, pp. 1473–1499, 2015.

[19] O. Hahm, E. Baccelli, T. C. Schmidt, M. Wählisch, C. Adjih, and L. Massoulié, "Low-power internet of things with NDN & cooperative caching," in *ACM ICN*, 2017, pp. 98–108.

[20] B. Chen, L. Liu, M. Sun, and H. Ma, "IoTCache: toward Data-driven Network Caching for Internet of Things," *IEEE IoT Journal*, vol. 6, no. 6, pp. 10 064–10 076, 2019.

[21] S. Fatale, S. Prakash, and S. Moharir, "Caching policies for transient data," in *IEEE National Conf. on Communications*, 2018, pp. 1–6.

[22] M. A. Hail, M. Amadeo, A. Molinaro, and S. Fischer, "Caching in named data networking for the wireless internet of things," in *IEEE RIoT*, 2015, pp. 1–6.

[23] Z. Zhang, C.-H. Lung, I. Lambadaris, and M. St-Hilaire, "IoT data lifetime-based cooperative caching scheme for ICN-IoT networks," in *IEEE ICC*, 2018, pp. 1–7.

[24] B. Zhou and W. Saad, "Joint status sampling and updating for minimizing age of information in the internet of things," *IEEE Transactions on Communications*, vol. 67, no. 11, pp. 7468–7482, 2019.

[25] M. A. Abd-Elmagid, N. Pappas, and H. S. Dhillon, "On the Role of Age of Information in the Internet of Things," *IEEE Communications Magazine*, vol. 57, no. 12, pp. 72–77, 2019.

[26] R. D. Yates, P. Ciblat, A. Yener, and M. Wigger, "Age-optimal Constrained Cache Updating," in *IEEE ISIT*, 2017, pp. 141–145.

[27] D. Kreutz *et al.*, "Software-defined networking: A comprehensive survey," *Proceedings of the IEEE*, vol. 103, no. 1, pp. 14–76, 2015.

[28] "Openflow switch specification - version 1.3.1 Open Networking Foundation (ONF)," September 2012.

[29] X. N. Nguyen, D. Saucez, and T. Turletti, "Efficient caching in content-centric networks using OpenFlow," in *IEEE INFOCOM WKSHPS*, 2013, pp. 67–68.

[30] A. Mahmood, C. Casetti, C. F. Chiasserini, P. Giaccone, and J. Härri, "Efficient caching through stateful SDN in named data networking," *Transactions on Emerging Telecommunications Technologies*, vol. 29, no. 1, 2018.

[31] A. Rao, O. Schelén, and A. Lindgren, "Performance implications for iot over information centric networks," in *Proc. of the 11th ACM workshop on challenged networks*, 2016, pp. 57–62.

[32] J. Dutta, S. Roy, and C. Chowdhury, "Unified framework for iot and smartphone based different smart city related applications," *Microsystem Technologies*, vol. 25, no. 1, pp. 83–96, 2019.

[33] M. Sharaf, M. Abusair, R. Eleiwi, Y. Shanaâ€™a, I. Saleh, and H. Muccini, "Modeling and code generation framework for IoT," in *Int. Conf. on System Analysis and Modeling*. Springer, 2019, pp. 99–115.

[34] S. Gao, Y. Zeng, H. Luo, and H. Zhang, "Scalable control plane for intra-domain communication in software defined information centric networking," *Future Generation Computer Systems*, vol. 56, pp. 110–120, 2016.

[35] J. Li *et al.*, "Popularity-driven coordinated caching in named data networking," in *ACM/IEEE ANCS*, 2012, pp. 15–26.

[36] Y. Yang *et al.*, "Inter-domain routing bottlenecks and their aggravation," *Computer Networks*, vol. 162, p. 106839, 2019.

[37] C.-C. Huang, N. Kakimura, and Y. Yoshida, "Streaming algorithms for maximizing monotone submodular functions under a knapsack constraint," in *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX/RANDOM 2017)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2017.

[38] L. Breslau *et al.*, "Web caching and Zipf-like distributions: Evidence and implications," in *IEEE INFOCOM*, vol. 1, no. 1, 1999, pp. 126–134.

[39] J. Yao and N. Ansari, "Joint content placement and storage allocation in C-RANs for IoT sensing service," *IEEE Internet of Things Journal*, vol. 6, no. 1, pp. 1060–1067, 2018.

[40] A. V. Ventrella, G. Piro, and L. A. Grieco, "Publish-subscribe in mobile information centric networks: Modeling and performance evaluation," *Computer Networks*, vol. 127, pp. 317–339, 2017.

[41] N. L. Van Adrichem, C. Doerr, and F. A. Kuipers, "OpenNetMon: network monitoring in openflow software-defined networks," in *IEEE NOMS*, 2014, pp. 1–8.

[42] S. Floyd, M. Handley, J. Padhye, and J. Widmer, "Equation-based congestion control for unicast applications," *ACM SIGCOMM Computer Communication Review*, vol. 30, no. 4, pp. 43–56, 2000.

[43] J. Padhye, V. Firoiu, D. Towsley, and J. Kurose, "Modeling TCP throughput: A simple model and its empirical validation," *SIGCOMM Comput. Commun. Rev.*, vol. 28, no. 4, pp. 303–314, Oct. 1998.

[44] H. Feng, Z. Chen, and H. Liu, "Performance analysis of push-based converged networks with limited storage," *IEEE Transactions on Wireless Communications*, vol. 15, no. 12, pp. 8154–8168, 2016.

[45] S. Zhang *et al.*, "Air-ground integrated vehicular network slicing with content pushing and caching," *IEEE JSAC*, vol. 36, no. 9, pp. 2114–2127, 2018.

[46] H. Gomaa *et al.*, "Estimating instantaneous cache hit ratio using markov chain analysis," *IEEE/ACM ToN*, vol. 21, no. 5, pp. 1472–1483, 2013.

[47] K. Takata, J. Ma, B. O. Apduhan, R. Huang, and N. Shiratori, "Lifelog image analysis based on activity situation models using contexts from wearable multi sensors," in *IEEE International Conference on Multimedia and Ubiquitous Engineering*, 2008, pp. 160–163.

[48] E. Yaacoub and Z. Dawy, "On using relays with carrier aggregation for planning 5G networks supporting M2M traffic," in *IEEE WiMob 2014*, pp. 124–129.

[49] J. Pfender, A. Valera, and W. K. Seah, "Performance comparison of caching strategies for Information-centric IoT," in *ACM ICN*, 2018, pp. 43–53.

[50] D. Rossi and G. Rossini, "Caching performance of content centric networks under multi-path routing (and more)," *Relatório técnico, Telecom ParisTech*, pp. 1–6, 2011.

Giuseppe Ruggeri received the master degree in electronics engineering in 1998. In 2002 he received the Ph.D. in electronics, computer science and telecommunications engineering. He is currently assistant professor at the University Mediterranea of Reggio Calabria. His current interests include self organizing networks, Internet of Things, Social Internet of Things.

Marica Amadeo is an assistant professor at University Mediterranea of Reggio Calabria. She received a master degree (2008) in telecommunications engineering from the University Mediterranea of Reggio Calabria, and a Ph.D. degree in 2013 from the same University. Her major research interests are in the field of Information-Centric Networking, Internet of Things and edge computing.

Claudia Campolo is an associate professor of telecommunications at the University Mediterranea of Reggio Calabria. She received an master degree (2007) and a Ph.D. degree (2011) in telecommunications engineering from the same university. Before her current appointment, she was an assistant professor at the University Mediterranea of Reggio Calabria (2012-2020). Her main research interests are in the field of vehicular networking, 5G and future Internet architectures.

Antonella Molinaro is an associate professor of telecommunications at the University Mediterranea of Reggio Calabria. Previously, she was an assistant professor with the University of Messina (1998-2001) and the University of Calabria (2001-2004), and a research fellow at the Politecnico di Milano (1997-1998). She was with Siemens, Munich (1994-1995). Her current research focuses on 5G, vehicular networking and future Internet architectures.

Antonio Iera is a full professor of telecommunications at the University of Calabria. He graduated in computer engineering from the University of Calabria in 1991, and received a Master's degree in IT from CEFRIEL/Politecnico di Milano in 1992 and a Ph.D. degree from the University of Calabria in 1996. Since 1997 to 2019 he has been with the University of Reggio Calabria. His research interests include next generation mobile and wireless systems, and the Internet of Things.