This is the post-print of the following article: 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," in IEEE Transactions on Network and Service Management, vol. 16, no. 2, pp. 661-674, June 2019 Article has been published in final form at: ieeexplore.ieee.org/document/8645682. doi: 10.1109/TNSM.2019.2900274

IoT Services Allocation at the Edge via Named Data Networking: From Optimal Bounds to Practical Design

Marica Amadeo, Giuseppe Ruggeri, Claudia Campolo, Antonella Molinaro
University "Mediterranea" of Reggio Calabria - DIIES Department
Email: {name.surname}@unirc.it

***Abstract*—Edge computing is a key paradigm to offload the core network and effectively process massive Internet of Things (IoT) raw data without sending them to the cloud. This paradigm normally relies on a set of purpose-built and pre-planned servers, which host storage and processing resources to provide IoT services close to the data sources, thus saving core network resources and offloading the remote cloud infrastructure.**

In this paper, we propose to turn the network edge into a dynamic, distributed computing environment that supports the provisioning of IoT services, by exploiting the recent evolution of Named Data Networking (NDN), supporting both name-based data retrieval and computation. Specific name structure and novel NDN forwarding mechanisms are designed; a distributed strategy is also engineered to select the service executor among edge nodes, with the objectives to *(i)* limit the raw IoT data traffic crossing the network, and *(ii)* allocate the service execution according to the nodes' available processing resources. Numerical analysis shows that the performance of the proposed framework approaches the one of the optimal solution of a formulated Integer Linear Programming problem. System-level ndnSIM simulations confirm that the proposal also outperforms the considered state-of-the-art benchmark solutions in terms of service provisioning time.

***Index Terms*—Internet of Things, Named Data Networking, Mobile Edge Computing, Information-Centric Networking**

I. Introduction

The future Internet will be dominated by the big amount of Internet of Things (IoT) data traffic generated by billions of heterogeneous devices serving different applications, including smart building, smart transportation and health care. IoT services typically rely on the processing of such – sometimes raw and redundant – data, and may span from simple computations, e.g., identifying the average temperature over a set of monitored values, up to computing-intensive image compression, video transcoding or data analytics. The dominant strategy is to place the application logic for data processing in a cloud server [1], thus guaranteeing a common interface to different players and easy application deployment. This approach, however, might require carrying a big amount of raw IoT data across the network to the remote data center, where cloud resources are available. Placing the IoT service execution in the cloud may cause network congestion and burden the cloud infrastructure, getting inadequate to sustain the ever growing demands for massive IoT processing services.

Mobile edge computing [2] can solve the mentioned issues by bringing computing resources closer to where IoT data are produced. However, making this concept viable implies addressing and solving some key challenges, including: *(i)* deploying computing servers in ad-hoc places close to the network edge, and *(ii)* retrieving data from IoT sources, managing service allocation to the best edge server, and securing all operations.

Information Centric Networking (ICN) and, in particular, its prominent Named Data Networking (NDN) instantiation, have been recently re-engineered to enable in-network processing [3]–[8], besides content dissemination in the future Internet. In the original ICN design, a consumer requests a content "by name", and the producer, or any in-network node storing a valid copy of that content, may answer with a named Data packet. This simple logic has been extended to support named function execution: a consumer requests a named function to be applied over a content (e.g., video compression), and the network uses advanced routing-by-name mechanisms to find the node executing that function and returning the processed content. Unlike host-centric approaches, ICN decouples the function from the identity of the node able to execute it. Hence, there is no need to know in advance the IP address of the executor. This revolutionary paradigm is still at a very early stage of development, and deserves further investigation.

In this paper, we extend NDN to support IoT data processing services at the network edge, i.e., the execution of a named function over a named IoT dataset. The reference services for our study are of two types: "data-intensive", which involve a large set of raw IoT data to be collected and processed, and "compute-intensive", where the function requires high processing capability to be executed. The type of processing function applied over the collected IoT dataset can be as simple as finding the maximum value in a set of sensed data or more complex such as compressing a set of images.

We design an effective and efficient strategy to select the NDN edge node among those capable of executing a specific processing service over a set of IoT data, retrieved through legacy NDN procedures. The selection of such node is driven by the following objectives: *(i)* to limit the raw data traffic across the network, by moving the computation closer to the IoT data sources, and *(ii)* to consider the edge nodes'

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).

processing capability when allocating the service execution function. This paper provides the following contributions:
1. We design a framework, referred to as *IoT-Named Computation Networking* (IoT-NCN), where the retrieval of IoT data and their processing at the network edge is distributedly orchestrated through a re-engineered NDN solution. The proposal extends the NDN naming scheme, the packets format and the forwarding strategy to support the IoT data collection and service execution.
2. We formulate the optimal service allocation through a simple Integer Linear Programming (ILP) problem, and enclose the aforementioned objectives in a weighted cost function to be minimized, in order to identify the set of candidate service executors at the network edge. The cost function considers both the latency in the IoT data retrieval by the executor and the processing footprint for function execution. The problem solution is used as a upper bound for performance comparison, since it represents what is achievable through a centralized scheme, not implemented in practical realization.
3. We design a practically feasible distributed service allocation strategy, which is included in the forwarding fabric of IoT-NCN and is based on the same cost function specified for the ILP. It allows edge nodes to compute the cost function locally and to make on line decisions independently, and orchestrates the service allocation in a distributed manner.
4. We evaluate the performance of the distributed strategy against the formulated optimal solution through Matlab simulations, and we study the impact of the service popularity and the effect of the weighting factor in the cost function. Numerical results demonstrate that the proposed distributed solution exhibits performance close to the optimal centralized one under a wide range of network and data settings, and correctly trades off the two contributions in the cost function.
5. Finally, we implement IoT-NCN as an extension to the NDN protocol stack, and conduct system-level simulations in ndnSIM [9], the official ns-3-based NDN simulator. Achieved results confirm the numerical ones and give more insights about the behaviour of the proposed framework, when considering a realistic NDN design and the comparison with alternative state-of-the-art solutions.

The remainder of the paper is organized as follows. Section II provides NDN basics, scans the closest related works in the field of in-network processing via NDN and identifies the motivations of our work. The main pillars of the proposed distributed IoT-NCN framework are summarised in Section III. The system model, problem statement and ILP formulation are reported in Section IV. Section V specifies the details of IoT-NCN for the edge orchestration of IoT data processing services. The performance evaluation methodology and the main settings are discussed in Section VI. Numerical and ndnSIM simulation results are, respectively, presented in Section VII and Section VIII. Open research directions are discussed in Section IX, and Section X concludes the paper.

II. Background and motivations

A. NDN pillars

The NDN architecture has evolved over the years, and it has been enriched with additional components to support routing and advanced forwarding mechanisms. In the following, we refer to the latest reference implementation in [10] that considers named content retrieval in the Internet.

Packets and data structures. NDN communication is receiver-driven and is based on the exchange of Interest packets requesting contents, and Data packets carrying them. They both include hierarchical content names. Security is performed per packet: producers create digital signatures on their Data packets; the signature binds the Data name with the generated content, so that its authenticity can be verified at any time, even if the Data is provided by an intermediate in-network node caching the content.

Each NDN node implements the model in Figure 1 that consists of the following data structures.

The *Content Store* (CS) caches Data packets traversing the node. The *Pending Interest Table* (PIT) records all incoming unsatisfied Interests forwarded by the node and tracks the related arrival time. There is a one-to-one matching between the Interest and the requested Data. Data received without a matching Interest, or Data arriving after the Interest lifetime expiration, are discarded. The *Forwarding Information Base* (FIB) is used to route Interests. Each entry records a name prefix with a collection of outgoing node's faces towards a potential content provider and the related routing cost. There are network faces (i.e., lower-level network interfaces) and application faces. In the latter case, the request is forwarded to the Application layer of the node.

The *Routing Information Base* (RIB) records routing information, which are registered and updated by different parties, e.g., the routing protocol and the application services. The RIB Manager handles the RIB and updates the FIB when needed.

The *Strategy Choice Table* (SCT) collects a set of named forwarding strategies associated with namespaces, since different names may require different forwarding semantics. The so called *Strategy Choice (SC) Manager* is the software component in charge of the SCT management, i.e., to set/unset strategies. It can also store measurement information regarding a name prefix in the so-called *Measurements Table* so to improve the forwarding strategy decisions.

Packet processing. At the Interest reception, an NDN content router first looks into the CS for the matching Data packet to send back. If the content is not found, it looks in the PIT. If there is a matching entry, the node updates the PIT with the incoming Interest's face and discards the request. Vice versa, it looks into the FIB for an outgoing face to send the Interest. Data packets simply follow the chain of the PIT entries back to the requester(s).

NDN for IoT data retrieval. Recent works showed that NDN is particularly suited to IoT data retrieval [11], even from low-powered devices [12]. Indeed, NDN names can be used to address heterogeneous things (e.g., sensors, cameras) and their data, directly at the network layer. NDN names have been proposed for different IoT contexts, like smart home [13] and lighting control [14]. Names can reflect application semantics and their scope, but they are not tied to a specific location, like for IP addresses.

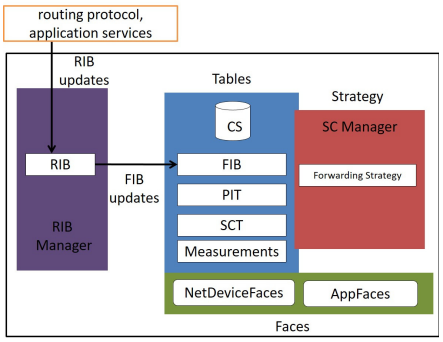


Fig. 1. NDN node model.

B. In-network processing via NDN

NDN has been recently recognized as a potential enabler of distributed in-network computation.

Named Function Networking (NFN) [4] was the pioneering proposal that complemented NDN data access by data processing. In NFN, a name interchangeably represents a mapping to a content, a function for processing the content, or an expression that combines the two. Therefore, the Interest can carry, in the NAME field, expressions involving named data as well as named functions, and the network is in-charge of discovering the node executing the function, by interlacing expression-resolution with name-based forwarding. For this purpose, the NDN forwarding machine is augmented with a $\lambda-$expression resolution engine, which processes all Interests that have the postfix name component */NFN*. The NFN strategy has been recently adapted for targeting IoT services and avoiding function execution at the data owner (i.e., the IoT node or the data center) by forcing execution at the edge [15].

Another work customizing NFN in the IoT domain is Programmable IoT (PIoT) [6], an application layer solution, *statically* installed in the most capable network nodes. It also includes a centralized management functionality that performs the service allocation in the IoT domain.

Closer to NFN, Named Function as a Service (NFaaS) has been deployed as a general purpose framework for in-network function execution [5]. Instead of using $\lambda-$functions that have expressiveness limits, NFaaS supports more sophisticated processing with lightweight virtual machines in the form of named unikernels. A Kernel Store in the node keeps the function codes, computes statistics and makes decisions on which functions to download and execute. Functions can migrate in the edge domain based on their demands: functions move closer to the data source to satisfy delay-sensitive applications and go towards the core for bandwidth-hungry applications.

With focus on IoT data processing, the authors in [16] propose a hybrid naming scheme composed of the following parts: *(i)* hierarchical NDN names to be used in the core network to locate a given IoT domain, *(ii)* a function tag, which identifies the function that will process IoT data in the identified IoT domain, *(iii)* a set of unordered tags that describe the relevant IoT data. Although tags can improve the expressiveness of naming, their use separates the IoT domain from the NDN network and requires ad-hoc designed delivery mechanisms, which are restricted to tree topologies.

With a similar objective of IoT data processing, in our previous work in [7], we advocate both data retrieval and processing to occur at an edge node. We propose a simple procedure for the selection of the executor node, with the relevant workflow aimed to process data as close as possible to the IoT sources, in order to limit the amount of raw data traffic across the network and to reduce the data retrieval latency.

C. Contributions

Compared to previous related works, our proposal shows the following distinctive features.

- It extends NDN to support in-network processing over IoT data. Unlike the previous literature on mobile edge computing [17], we target a more holistic solution in that the networking protocol for data retrieval and the computing schemes are designed in a closely-knit manner to make the best of the available edge resources and to satisfy the service demands. IoT data retrieval occurs through the legacy NDN solution, by benefiting from all its well recognized advantages [11].
- Unlike PIoT [6], in our framework in-network processing is distributed and relies on NDN capabilities augmented with the cognition of "named services".
- Being distributed, our proposal shares design principles with NFaaS [5] and NFN [3], although there are also important differences. While NFaaS is a general framework focusing on functions to be executed and migrated according to their requirements, our proposal is specifically tailored for IoT scenarios, where the IoT data retrieval has a key impact in the service provisioning performance. In NFaaS, the name prefix identifies the function first, thus Interests are forwarded, based on this prefix, to an executor. In IoT-NCN, instead, the Interest carries a composite name that identifies first the content and, then, the function. As a result, the Interest is routed towards the data sources and the executor must be located in the path to the data sources.
- The proposed IoT-NCN logic is similar to the one in NFN. What is mainly new w.r.t. NFN is the forwarding process coupled with the executor selection strategy, based on a *multi-criteria cost function*. The function aims to improve the service provisioning performance by accounting for *the node's resource capability*, the *service type*, and the *closeness to the IoT data sources*. This is also the crucial improvement w.r.t. our previous work in [7], which only considers the closeness to data sources for the executor selection and remains oblivious of the other contextual information.
- Unlike [16], which executes tag-named functions only inside the IoT domain, and other edge-based solutions where computation is demanded to purpose-built nodes [17], in IoT-NCN any node at the network edge can execute a service, according to its capabilities. This increases the flexibility of the framework and the overall performance, while better distributing the load across network nodes.
- Similar to what happens in NDN for contents requested by multiple users, in the proposed IoT-NCN the same service can be also requested by multiple consumers. In this context, *the same service means the same function applied to the same*

input data. If this happens, an IoT-NCN node executes the function only once over the same dataset and, with the cached results, it serves different consumers [18] in several contexts. This is what we call *service reuse*. These could be the cases of, for instance, augmented reality services required by users attending the same event (e.g., in a museum, in a concert); requests of the same analytics over a given road traffic status information, by both the municipality and the road authority. Caching computing results comes for free thanks to the NDN name-based communications, which allows each network node to detect different requests for the same (service) name. On the contrary, this is less viable when a data-agnostic networking protocol like IP is implemented, and the service request must be inspected at the application layer. This limits the service execution to purpose-built nodes in an ossified network edge deployment [18], [19]. Certainly, in IoT-NCN the service reuse cannot be applied to those services that rely on personal information that cannot be cached, or on volatile data, whose validity would expire rather quickly [3].

III. The IoT-NCN framework: an overview

IoT-NCN provides IoT named services by leveraging the NDN communication architecture. The conceived framework is intended to entail minor modifications of the legacy NDN to keep backward compatibility with it. To this purpose, we use a naming scheme that identifies IoT contents and functions without affecting the NDN routing logic. A few novel fields in the legacy Interest/Data packets, and a couple of new modules in NDN nodes are added to enable the distributed edge service orchestration, as detailed in the following. The interaction workflows are also specified which involve the edge nodes with the aim to identify the best executor – i.e., the one minimizing the conceived cost function – on the path between the consumer and the IoT data to be processed.

A. Reference scenario and service assumptions

The reference IoT services for our study can be *data-intensive*, here meaning that they involve a large amount of raw IoT data to be collected and processed; and *compute-intensive*, i.e., requiring high processing capability. Executing such services at the network edge would offload the core network, with the additional benefit of reducing the latency incurred by consumers to retrieve the service output. This is because it is quite common in IoT contexts to process massive data (e.g., from a multitude of sensors) in order to extract meaningful information, with a computed result which typically has a smaller size w.r.t. the input raw data.

The framework is intentionally conceived to be quite general for what concerns the types of supported processing functions. They may imply simple computation such as finding the maximum, minimum, or mean value over a set of sensed IoT data (e.g., temperature, CO_2 emissions, vehicle speed); more complex data aggregation, spatial/temporal correlation or filtering operations on the raw data; compute-intensive image compression, video transcoding or analytics from different surveillance cameras; augmented reality creation of a scene (e.g., in a museum, a concert, an autonomous vehicle).

Named services are requested by consumer end nodes, and executed by in-network (edge) IoT-NCN nodes, in the following referred to as executors. All the nodes, including consumers and IoT sources, communicate through the NDN Interest/Data exchange. IoT-NCN can work alongside the cloud, which is used for long-term data analysis and storage, and when no edge node is available for the requested processing.

We refer to a general network topology composed of a *core network*, with high-speed content routers, that is interfaced through Border Router(s) to multiple edge domains. These latter separate the core from the IoT sources and include multiple nodes acting as potential executors. Each edge domain may resemble the backhaul segment of a cellular network.

The following main assumptions hold:

- Multiple consumers can issue requests for *heterogeneous IoT services*. Consumers do *not* execute services, but they only request service executions with a named Interest packet. In the following, we assume the more general case in which consumers are remote w.r.t. the IoT data sources, therefore service requests traverse the core network and reach the edge domain through the Border Router.
- Core routers may cache incoming Data packets, as in the traditional NDN implementation, but they do *not* execute services to maintain a light and fast core network.
- A service can be executed in any IoT-NCN edge node on the path between the consumer and the IoT data source(s). It is not executable in an IoT data source, because of the limited computational capacity and because, in the most general case, a service requires data collection from multiple IoT sources.
- Services are atomic in that they cannot be split in multiple sub-tasks, so each service is allocated to a single node.
- A service is uniquely named and characterized by a self-consistent function code with well-defined data input parameters and information about the computing resources requested for its execution. We pose no restrictions about the IoT function implementation; functions could be implemented as unikernels like in [5], as containers like in [6], as Python scripts [19], which are quite common in IoT applications. Identifying the best implementation approach is out of the scope of the current paper.

B. IoT services naming

Like in [7], IoT-NCN leverages an ad-hoc specified naming scheme that identifies *both* the IoT data and the processing function that will be applied over them. Specifically, both data and functions are hierarchically named, with the function name added *after* the data name, in order to not affect the legacy NDN forwarding. The prefix *NCN* is used to delimit the function name from the data name. If the NCN tag is not included, the Interest is treated as a legacy NDN content request by IoT-NCN nodes.

The service name can also include a limited set of parameters as input for the execution. For instance, an Interest with name */unirc/buildingA/temp/NCN/avg/*$\{p_1$*=10*$\}$ requests the execution of an average operation over the last 10 temperature values collected from building A of the University of Reggio Calabria.

Nodes without computing capabilities maintain information only on the data name prefix (e.g., /unirc) and route the packet according to it. This guarantees backward compatibility with legacy NDN implementations.

C. Service Provisioning

At high level, service provisioning in IoT-NCN consists of the following steps.

Interest forwarding with executor selection. IoT-NCN uses persistent Interests, a.k.a. Long-Lived Interests [20], to request a service. A persistent Interest is used to maintain active the Interest in the PIT for the entire service provisioning time – since the Interest transmission by the consumer to the reception of the response including the output of the computation – and to retrieve more Data packets if the processing result needs to be conveyed in multiple packets.

To prevent parallel execution of the same service, broadcast/multicast forwarding is forbidden in IoT-NCN: the persistent Interest is sent only on a single outgoing face, the one with the lowest cost in the FIB. The Interest is forwarded towards the IoT sources according to the data name component(s) only. When the request reaches the edge domain, the service name component is considered and a distributed process selects the edge node with the lowest *service execution cost* as the IoT service executor. Information about the decision engine and the cost calculation will be given in the following sections.

Service Execution. Service execution is split into two phases: *Data collection* and *Computation*. According to the data name component and the service input parameters, the selected executor first collects the IoT Data using legacy NDN Interest-Data exchange, and then processes them. Data collection generally implies the retrieval of multiple Data packets, either from the same producer (*single-source*) e.g., generating a video stream, or from multiple producers (*multi-sources*) e.g., measuring a set of sensed values in a specific region. Data provided by devices equipped with the same type of sensors (e.g., for measuring temperature, traffic conditions) usually share a common name prefix, but differ in the last component(s), e.g., *unirc/buildingA/temp/room1* and *unirc/buildingA/temp/room2* are used to collect temperature values from two different rooms of the same building.

In legacy NDN, several strategies can be used to collect more Data under the same prefix [21]. The classic approach is based on the so-called *exclusion-filters*: *Exclude* selectors in the Interest header specify a list and/or ranges of name components that must not appear as a continuation of the name prefix in the responding Data packet[1]. By applying this technique to the example of temperature values collection, if the first Interest *unirc/buildingA/temp/* retrieves the Data *unirc/buildingA/temp/room1* from *room1*, then a second Interest must be issued with *room1* in the Exclude field in order to retrieve Data from *room2*.

Result delivery. The computed result is sent back to the consumer into one or more Data packets, which carry the same name of the original persistent Interest. Results that include shareable contents can be cached to satisfy future requests.

[1]https://named-data.net/doc/ndn-tlv/interest.html

IV. Optimal services allocation

In this section, we formulate the optimal IoT service allocation problem in IoT-NCN. The optimization problem solution is based on the ideal availability of a centralized service orchestration, which conflicts with the NDN distributed philosophy. Therefore, this solution will be taken as an upper bound for the performance comparison shown in Section VII.

Plenty of literature works have studied the optimal services allocation problem – sometimes referred to as task allocation problem – being it crucial in cloud computing and its recent edge evolutions. In the context of mobile edge computing [17], the main solutions can be classified as: *(i) single-user* edge systems, where the decision is on whether a particular task should be (partially, if split in sub-tasks) offloaded for edge execution or instead computed locally on the mobile device; *(ii) multi-user* edge systems comprising multiple mobile devices that share one edge server; and *(iii)* systems with *multiple edge servers*, where the issue is to determine the destination server of the computation offloading.

The problem of service allocation in IoT-NCN especially resembles the third category. The main difference lies in the fact that nodes which can act as *executors are not just purpose-built edge servers, but also in-network (edge) NDN content routers.* This is viable thanks to *(i)* the gradual shift of ICN network nodes from simple forwarders to more capable computing and storage platforms; and *(ii)* the network function virtualization (NFV) paradigm, whereby services can be flexibly hosted at any network location that has virtualized resources, e.g., provided by commodity servers [22]. Overall, this makes the problem different from existing literature and quite unique, being tightly coupled with the NDN philosophy.

In the following, we will consider the *service execution cost* as the cost function to be minimized in the defined optimization problem; the same cost function will be used to define the practical distributed strategy for service allocation detailed in Section V.

A. System model

The reference network edge topology for our study in Figure 2 is a metropolitan area network (MAN) built as a ring

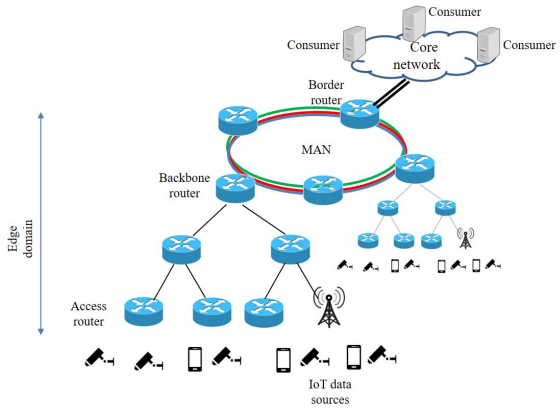


Fig. 2. Reference topology.

TABLE I
SUMMARY OF THE MAIN NOTATIONS

Symbol	Description
S	catalog of services ($\|S\| = s$)
N	set of nodes in the topology ($\|N\| = n$)
R	set of service requests ($\|R\| = r$)
s_k	a generic service in the catalog
C_{s_k}	set of consumers requesting service s_k ($\|C_{s_k}\| = c$)
α_{s_k}	weight for the service execution cost function related to s_k
r_k^j	j-th request of service s_k (with $j \in C_{s_k}$)
n_i	a generic IoT-NCN node in the edge domain
$\pi_{r_k^j}$	set of nodes in the consumer-source path for r_k^j
$aCPU_{n_i}$	available computation capability (i.e., CPU cycles per second) at node n_i
CPU_{s_k}	number of CPU cycles required for executing service s_k
$c_{r_k^j}$	the consumer issuing service request r_k^j for service s_k
AR_{s_k}	access router to which is(are) connected the IoT sources(s) which own(s) the input data for service s_k
SEC_{n_i,s_k}	cost for executing service s_k at node n_i
CC_{n_i,s_k}	closeness cost for executing service s_k at node n_i
PC_{n_i,s_k}	processing cost for executing service s_k at node n_i
$D_{n_i,AR_{s_k}}$	latency measured over the path connecting n_i and AR_{s_k}
D	maximum diameter of the topology
X_{n_i,s_k}	binary variable taking the value 1 if s_k is executed by node n_i and 0, otherwise

backbone, with 4 backbone routers connected through a 3-layered tree topology to *leaf nodes*, which are the Access Routers (ARs) towards the IoT domains. ARs may operate as base stations to which the IoT devices are *directly connected*, e.g., in case of Low Power Wide Area Network (LPWAN) or Narrowband-IoT cellular solutions. IoT devices may be also connected to the ARs *through short-range connectivity with IoT gateways*.

We assume that a Border Router, which links the MAN to the core network, receives the service requests from the remote consumers. The core network, being not involved in the service allocation performed at the edge, is simply modeled as a link with a fixed delay.

All ARs, intermediate edge nodes and backbone nodes, including the Border Router, implement IoT-NCN, therefore they have computing capabilities and participate in the service allocation process.

We refer to N as the set of IoT-NCN nodes in the topology, thus excluding consumers, IoT data sources and gateways. Each node $n_i \in N$ has $aCPU_{n_i}$ computing resources available to perform service(s) execution (expressed in CPU cycles per unit time). We reasonably assume that nodes closer to the backbone ring have more available computing resources and they could better manage resource-intensive tasks.

We consider a catalog of services $s_1, ..., s_s \in S$. Each service s_k is characterized by *(i)* a name that, as previously described, identifies the function, its parameters and the input data; *(ii)* a CPU_{s_k} value, which is the amount of computing resources required to execute the service (i.e., the number of CPU cycles).

IoT devices are typically deployed on the ground to monitor/measure physical phenomena in a given area, e.g., within a smart building, a smart city. For example, different surveillance cameras capture multiple angles of the same city neighbourhood; multiple smart meters in the same building transmit their regular measurements. Due to this co-locality of IoT data sources, it is reasonable to assume that they are under the coverage of the same AR, which provides connectivity to the area over which monitoring is enforced. When the set of IoT data is spread over more ARs, the paths from the Border Router towards these ARs will cross in a common upper-level node that we call Branching Node (BN). Without loss of generality, we assume that the input IoT data for a given service s_k are reachable through a common IoT entry point, which is the AR, denoted as AR_{s_k}, when the IoT sources are all under the coverage of the same AR, and is the BN, denoted as BN_{s_k}, when the IoT sources are distributed under the coverage of different ARs. For the sake of simplicity, in the remainder of this paper we will use the term AR_{s_k} to refer to both situations.

Each service s_k in the catalog can be requested by different consumers in the set C_{s_k}. When different consumers require the same service, an IoT-NCN node may serve different requests by executing the same function over the same set of data only once. We refer to R as the set of distinct service requests by different consumers. A specific request for a service s_k by consumer j in C_{s_k} is denoted as r_k^j, with $j = 1, ..., c$ and $k = 1, ..., s$.

For each service request, prior to allocation, the minimum routing path is computed between each consumer $j \in C_{s_k}$ requesting service s_k, $c_{r_k^j}$, and the IoT sources generating the input data. This is done according to the Dijkstra algorithm, which is currently used in NDN routing implementations [9]. Without loss of generality, the latency measured over each link is chosen as the routing metric. Thus, the minimum routing path is the one ensuring the *minimum latency* between the consumer and the IoT data sources.

Once the path has been identified, for each service request, r_k^j, we additionally have: $\pi_{r_k^j} \subset N$, which is the set of IoT-NCN nodes in the forwarding path between consumer *j* requesting service s_k and AR_{s_k}.

B. Optimization problem

Defining the problem. Given the system model, our aim is to compute the service allocation optimally. The problem can be formulated as an integer (binary, 0 or 1) LP problem, as it will be clarified in the following.

We assume that each IoT-NCN node $n_i \in N$ can execute any service in the catalog, according to its available CPU capability, $aCPU_{n_i}$. Not all feasible assignments produce desirable application performance, thus we introduce the *service execution cost* function to be minimized, which analytically defines an order relationship among all feasible solutions. The definition of the service execution cost does not target a single performance metric to be minimized/bounded, instead it is a multi-criteria cost function that reflects our twofold aim of: *(i)* limiting the amount of raw IoT data traversing the network, and *(ii)* allocate service execution to nodes according to their available processing resources. The two goals are conflicting in most cases. Executing services closer to the IoT data sources would relieve the network from the

burden of carrying a big amount of raw IoT data. However, nodes closer to the IoT sources, such as the AR_{s_k}, may have limited computing capabilities compared to a Border Router and could not effectively manage the computing load related to s_k execution.

Thus, we transform a multi-objective optimization problem into a single-objective problem, using the Simple Additive Weighting (SAW) technique [23]. According to SAW, we define the service execution cost as the weighted sum of the normalized *closeness cost* (CC) and the normalized *processing cost* (PC) at each potential executor. The CC indicates the proximity of the potential executor to the IoT sources, while the PC reflects its available computing capabilities w.r.t. the ones requested by the service.

For the sake of the reader's convenience, the main notations of the problem are summarized in Table I.

The Cost function. The cost of executing service s_k at node n_i is defined as:

$$SEC_{n_i,s_k} = (1-\alpha_{s_k})CC_{n_i,s_k} + \alpha_{s_k} PC_{n_i,s_k}. \quad (1)$$

The first contribution, CC_{n_i,s_k}, is the *closeness cost* for node n_i to collect input IoT data for service s_k. Minimizing this contribution limits the injection of raw IoT data into the network. Being AR_{s_k} the closest node to the IoT sources feeding service s_k, the minimum closeness cost (zero) is achieved when AR_{s_k} is chosen as the service executor. In a more general case, the closeness cost depends on the proximity between node n_i (candidate as executor of the service request) and the relevant AR_{s_k}. This value can be measured in terms of the chosen routing metric, that is the *latency* between node n_i and AR_{s_k}. We define the closeness cost as:

$$CC_{n_i,s_k} = \frac{D_{n_i,AR_{s_k}}}{D}, \quad (2)$$

where: $D_{n_i,AR_{s_k}}$ is the latency between the potential executor n_i and AR_{s_k}, which is normalized over D, representing the latency over the path between the Border Router and AR_{s_k}. It is worth observing that the closeness cost reflects the performance in terms of IoT data collection latency. Indeed, the closer the executor to the IoT sources, the shorter the latency of the raw data retrieval by the executor. The latency increases when the executor moves away from the AR especially if the amount of raw data to be transferred towards the executor is huge.

The second contribution, the *processing cost* PC_{n_i,s_k} depends on the total number of CPU cycles, CPU_{s_k}, required to execute service s_k, and the computation capability available at node n_i when the service allocation procedure is executed, $aCPU_{n_i}$. It is defined as:

$$PC_{n_i,s_k} = \begin{cases} \frac{CPU_{s_k}}{aCPU_{n_i}} & \text{if } CPU_{s_k} \leq aCPU_{n_i} \\ \infty & \text{if } CPU_{s_k} > aCPU_{n_i} \end{cases} \quad (3)$$

Nodes with higher computing capabilities have a lower processing cost and they are preferred to execute CPU-intensive services. Since service preemption is not included in our framework, PC_{n_i,s_k} is set to ∞ if the node cannot execute the computation due to lack of currently available processing resources.

The weight parameter α_{s_k}. This parameter ($\alpha_{s_k} \in [0,1]$) has a main role in weighting the two cost contributions in Equation (1). It is *service-specific*, in that it characterizes a service and, thus, is the same for different requests, r_k^j, of the same service s_k, whichever the consumer j. We define α_{s_k} in a systematic way, in order to capture the requirements of heterogeneous services, namely CPU-intensive and data-intensive services. In the literature, e.g., [24], it is common to distinguish such services by defining the ratio between the computation cost and the communication cost, namely the Computation-to-Communication Ratio (CCR). Intuitively, in a data-intensive service the data to be retrieved outweigh the computational requirements for processing them. Services with a high CCR are CPU-intensive in nature, vice versa they are data-intensive. We compute the CCR of a service s_k as the ratio between the execution time and content retrieval time. More in detail: $CCR_{s_k} = \frac{CPU_{s_k}}{CPU_{ref}} / \frac{D_{s_k}}{B_{ref}}$, where CPU_{ref} and B_{ref} are, respectively, the minimum computation capability and data rate measured in the domain and D_{s_k} is the size of the input content. CCR_{s_k} can be computed per each service s_k in the catalog S. According to the above formula, a service is CPU-intensive if $CCR_{s_k} > 1$ and data-intensive, otherwise. Then, a feature scaling operation is performed on all services in the domain's catalog S to normalize the independent CCR_{s_k} in the range [0,1] in order to obtain α_{s_k}[2]:

$$\alpha_{s_k} = \begin{cases} CCR_{s_k} \cdot 0.5 & \text{if } CCR_{s_k} \leq 1 \\ \frac{CCR_{s_k} - 1}{\max\limits_{s_j \in S} CCR_{s_j} - 1} \cdot 0.5 + 0.5 & \text{if } CCR_{s_k} > 1 \end{cases} \quad (4)$$

For data-intensive services, which require the collection of a huge amount of data prior to computation, α_{s_k} is set to values lower than 0.5. The lower α_{s_k} in Equation 1, the closer the executor will be to the IoT data sources. In so doing, the amount of raw data traffic traversing the network is reduced. Vice versa, for α_{s_k} values higher than 0.5, CPU-intensive services are modeled. Values of α_{s_k} close to 1 push the CPU load for service execution to more capable nodes.

Problem formulation. Let X_{n_i,s_k} be a binary variable taking the value 1 if service s_k is executed by node n_i, and the value 0, otherwise. We formulate the problem of the optimal allocation of services $s_k \in S$ to potential node executors $n_i \in N$ as follows:

$$min \sum_{s_k \in S} \sum_{n_i \in N} SEC_{n_i,s_k} X_{n_i,s_k} \quad (5)$$

s.t.

$$\sum_{s_k \in S} CPU_{s_k} X_{n_i,s_k} \leq aCPU_{n_i} \; \forall n_i \in N \quad (6)$$

$$\sum_{n_i \in \pi_{r_k^j}} X_{n_i,s_k} = 1 \; \forall r_k^j \in R \;\; j = 1,...,c; k = 1,...,s \quad (7)$$

$$X_{n_i,s_k} = \{0,1\} \; \forall s_k \in S, \forall n_i \in N \quad (8)$$

[2]In this work, we are considering a fixed set of services S, but in a real scenario the catalog could reasonably vary over time and α_{s_k} values will change with it.

The constraint in Equation (6) limits the allocation of a service on a node according to its available resources. Equation (7) guarantees that each service, s_k, is executed in one and only one node in a given consumer-AR_{s_k} path, $\pi_{r_k^j}$, computed for a service request r_k^j. Such a constraint prevents the same service to be executed multiple times on the same path. In case of two partially overlapping paths, let say $\pi_{r_k^j}$ and $\pi_{r_k^i}$, Equation (7) still allows service s_k: *(i)* to be executed only once on the nodes $n_z \in \pi_{r_k^j} \cap \pi_{r_k^i}$ or *(ii)* to be replicated on the nodes $n_z \notin \pi_{r_k^j} \cap \pi_{r_k^i}$. Indeed, Equation (6) sums all the resources required to execute services $s_k \in S$ and not each distinct service request. Finally, constraint in Equation (8) reminds that we conveniently model the allocation problem with binary variables.

The defined problem will be solved as described in Section VI and related results will be shown in Section VII.

V. Distributed service allocation in IoT-NCN

In this Section, we describe how the distributed service allocation is performed in the proposed IoT-NCN framework. The mechanism allows each node to *autonomously* decide if performing the IoT computation services and collect the relevant data. The decision process takes into account the available processing resources of the node, its closeness to the data sources (the latency routing cost), and the peculiarities of the requested service (through the α_{s_k} parameter).

Unlike [5], where nodes advertise the functions they can execute, IoT-NCN does not include any publishing procedure, since advertising both functions and content names could overwhelm the network. This is especially true considering that the service execution capability in a given node is dynamic, e.g., because of the time-varying resource availability. So, IoT-NCN adopts an *implicit* service discovery mechanism that runs in piggybacking with the persistent Interest forwarding and involves only IoT-NCN nodes.

A. Basics

The proposed distributed IoT-NCN framework requires the following modifications in the NDN primitives and entities:

- *IoT-NCN persistent Interests and related ACKs* are included that inherit the same formats of NDN Interest and Data packets, respectively, and provide an additional header field, called ServiceExecCost, which carries the currently *lowest* service execution cost.

- The *SC Manager* in each IoT-NCN edge node (see Figure 1) is extended with a *decision engine* that computes the cost of a local service execution, upon the IoT-NCN persistent Interest reception. This information, then piggybacked in the Interest itself, allows service orchestration to be performed in a completely distributed way, thus keeping low both the complexity and the signaling overhead.

- The *Available Service Table* (AST) is added to IoT-NCN nodes for fast checking the availability of the service and its parameters, during the persistent Interest forwarding process. The AST records the name of the functions that can be locally executed, the requested CPU load and their status: *(i) running*, if the function is in execution, or *(ii) inactive*, otherwise.

Each node could install or remove available service code (i.e., function) according to parameters like the service popularity, its local computing resources, for instance by following the ranking engine deployed in [5].

B. The distributed IoT-NCN operation

Interest forwarding and executor selection. The consumer sends the persistent Interest with the ServiceExecCost set to a default maximum value. This Interest is forwarded towards the IoT domain at the network edge, considering only the content name component.

When the Interest reaches the first IoT-NCN node (i.e., the Border Router in our scenario), a novel IoT-NCN forwarding strategy is started. Specifically, at the persistent Interest reception, the IoT-NCN node n_i first looks in the CS for a cached result. If no matching if found, n_i looks in the AST:

- If the service code is available and already *running* locally, the node just adds the arrival face of the request in the PIT and discards the Interest. At the end of the computation, the result will be sent also over the newly added face to reach the other consumer.
- If the service code is available but *inactive* locally, then the decision engine controlled by the SC Manager calculates the local service execution cost (*l-SEC*) and compares it with the one included in the ServiceExecCost field of the received Interest (*SEC*):
  - If *l-SEC* is greater than the current *SEC*, there is a previous more convenient service executor. Therefore, the Interest is simply forwarded according to the content name prefix.
  - If *l-SEC* is lower than the current *SEC*, the SC Manager updates the ServiceExecCost field with the new value before continuing the persistent Interest forwarding towards the IoT source(s) (and the relevant AR), according to the data name component. A PIT entry is created, which includes also the estimated service execution cost as an additional parameter.
  - If *l-SEC* is equal to *SEC*, only a PIT entry is created to record that the node is a candidate executor together with the correspondent cost.
- If the service code is *not available*, then the Interest is simply forwarded according to the content name prefix.

This process continues until the Interest reaches the last IoT-NCN node before the IoT domain, i.e., the AR. Then, the following cases may occur.

If the AR is able to execute the service with the lowest cost, then it will act as the executor. Otherwise, it generates the *IoT-NCN executor acknowledgement* (e-ACK) packet and sends it backward. The e-ACK is a Data packet, with the ContentType field set to the newly defined value IOT-NCN EXECUTOR ACK. It carries the same persistent Interest name, the (lowest) service execution cost as read from the received Interest, and a dummy payload. The e-ACK follows the backward chain of PIT entries, thus each candidate executor can check if the published cost is equal to the one stored in its PIT. The nodes without a matching cost simply delete the Interest from the PIT and continue forwarding the e-ACK

back. The first node with the matching cost finally stops the e-ACK forwarding and acts as the service executor. Thus, the e-ACK serves the purpose of letting the executor know that it has been selected.

After the e-ACK processing, the executor starts the service provisioning and, in parallel, confirms that the service is running to the nodes back up to the consumer, by sending an *IoT-NCN service acknowledgement* (s-ACK). This packet has the same structure of the e-ACK, but the CONTENTTYPE field is set to the newly defined value IOT-NCN SERVICE ACK, which simply notifies that the service has been allocated and the persistent Interest can be maintained in the PIT to allow the forwarding of the result. If the s-ACK is not received within a pre-defined time-to-live interval called s-ACK_{TTL}, e.g., because some errors occur in the network or all the nodes are busy, the persistent Interest is finally removed from the PIT. The consumer can retransmit the Interest when the $s-ACK_{TTL}$ expires for a fixed number of retries and finally give up if no acknowledgement is received. In the latter case, to obtain the service, the consumer may forward the Interest towards the cloud, similarly to e.g., [5].

Data Retrieval and Service Execution. The executor starts the IoT data retrieval by sending legacy NDN Interest(s) with the content name obtained from the IoT-NCN Interest name, as described in Section III. Interests are routed towards the IoT sources according to the standard NDN forwarding mechanism. Thanks to in-network caching, the requested Data could be retrieved without interactions with the original sources, if they are already available in the CS of intermediate nodes or also right at the executor.

When the data collection is completed, the executor performs the requested computation and, eventually, it sends the result to the consumer(s) in one or more Data packets, depending on the output's size. Thanks to the persistent Interest pending in the PIT, the nodes behind the executor, maintaining the persistent Interest in the PIT, can forward back the Data packets. The persistent Interest is finally consumed by the last Data packet that composes the result. According to the legacy NDN specification, the final packet in a sequence is identified by the meta-info field called FINALBLOCKID.

C. Local Service Execution Cost Calculation

The local service execution cost, *l-SEC*, is computed by each IoT-NCN node as in Equation 1.

The closeness cost is computed by reading the routing cost from the FIB entry per the requested content name. It is normalized w.r.t. the maximum routing cost in the edge domain, which is assumed to be advertised by the routing protocol.

The processing cost considers the resource requested for the service computation (CPU_{s_k}) that is recorded in the AST, and the currently available node's resource ($aCPU_{n_i}$) that is provided by a standard CPU monitoring utility available in the node. For what concerns the weight parameter α_{s_k}, it is an attribute of the service name and expressed as a globally recognizable weight factor, similarly to [25].

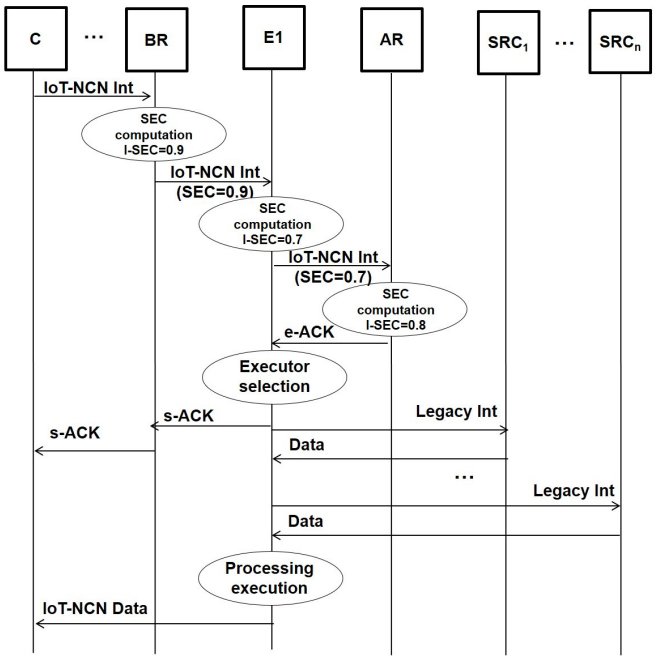


Fig. 3. Main steps of the Interest/Data exchange in IoT-NCN: a consumer C requests a service execution over data provided by IoT data sources SRC_1, ..., SRC_n, and it is executed by the edge node E_1.

D. A toy example

In order to better figure out how the proposed forwarding strategy works in IoT-NCN, let us refer to the toy example reported in Figure 3. A consumer transmits an IoT-NCN persistent Interest to request a named service. First, the Interest traverses the core network routers (not reported in the Figure), where it is forwarded according to the legacy NDN rules, considering only the content name component that drives the Interest towards the IoT domain where data sources are located.

Once the packet reaches the Border Router and enters the IoT-NCN edge domain, each crossed IoT-NCN node computes the service execution cost (SEC) and replaces the current value of the SERVICEEXECCOST field in the forwarded persistent Interest, only if the cost is lower than the one carried in the received Interest. The persistent Interest is then forwarded until it reaches the AR towards the IoT data sources.

In the example, the AR is not able to execute the service with the lowest cost, thus it transmits an e-ACK back. Once the e-ACK packet reaches the edge node that advertised the lowest cost, it realizes to be the executor because it has the matching lowest cost. Therefore, it sends an s-ACK to the consumer and starts the data retrieval from the IoT domain through legacy NDN Interest/Data packets. Once Data are collected, the executor performs the requested computation and sends the result to the consumer with the same name as the original persistent Interest.

VI. PERFORMANCE EVALUATION: METHODOLOGIES, SCENARIO AND SETTINGS

Performance evaluation of the proposed IoT-NCN architecture is carried out in two steps including both numerical and

TABLE II
MAIN SIMULATION SETTINGS

Parameter	Value
$\|S\|$	1000
$\|R\|$	50
Zipf's parameter (Z)	0; 0.4; 0.8; 1.2; 1.6
α_{s_k}	varying
CPU_{s_k}	uniformly distributed in [25, 375] Mcycles
$aCPU_{n_i}$	[1500-1000-500] MHz at the inner, intermediate and leaf nodes

simulation results.

First, we derive the *optimal* IoT service allocation by solving the problem formulated in Equation (5) with *linprog*, the Matlab linear programming solver. This is aimed to understand to which extent the formulated ILP problem and the cost function in Equation (1) are able to capture the targeted objectives, and to set the upper-bound solution for performance comparison.

A simplified distributed IoT-NCN framework is also implemented in Matlab to get a preliminary insight about the viability of the distributed proposal and compare it against the optimal bound computed by assuming a centralized service orchestration. The distributed allocation is carried out by considering each service request in isolation in a sequential order. For each request, the service execution cost at each candidate executor node along the path is computed according to Equation 1 and the service is allocated to the one that provides the minimum value.

Then, IoT-NCN is evaluated in a realistic way by means of ndnSIMv2.5 [9], the official discrete-event ns-3-based simulator deployed by the NDN community. Being based on the *ndn-cxx* library and the NDN Forwarding Daemon (NFD), ndnSIM enables reliable simulation experiments and their reproducibility in real environments with no need to modify the source code. We have modified ndnSIM from its stock installation to implement the IoT-NCN features and the benchmark schemes chosen for performance comparison (see Section VIII). Specifically, *(i)* the structure of Interest and Data packets has been modified to include the new information for the service allocation procedure; *(ii)* the NFD modules have been extended to accommodate the new devised naming scheme, the novel decision engine, the service execution cost computation, and the service execution at the selected executor nodes.

The reference topology for all evaluations is the same as in Figure 2. It includes: *(i)* 29 edge nodes, including the Border router, 4 Backbone Routers, 8 intermediate nodes and 16 ARs; *(ii)* 100 IoT sources, uniformly attached to the different ARs, *(iii)* 50 (remote) consumers that are connected to the Border Router through the core network, simulated as link with a delay of 50ms. All the edge nodes implement the IoT-NCN architecture; they can execute services according to their processing capabilities, which are assumed to be lower for ARs, and higher as moving towards the Border Router.

We consider a catalog of 100 services, each one requiring a randomly selected amount of CPU resources, in the range [25-375] Mcycles. Since there are no datasets containing traces of the service workloads in real IoT scenarios, similarly to [18], we rely on a synthetic workload as generated through the well-known Zipf's distribution. Results are reported when varying the popularity parameter Z, from 0 to 1.6, in order of increasing popularity. The main simulation settings are reported in Table II, and they hold unless differently stated in the text.

VII. OPTIMAL VS. DISTRIBUTED SOLUTIONS: NUMERICAL RESULTS

Metrics. The following metrics are computed for the purpose of performance comparison.

The *average CPU usage* per node: it is computed as the ratio between the CPU resources required by the allocated services over the available CPU at each node, and this is averaged over all edge nodes. This metric allows measuring to which extent edge nodes are loaded w.r.t. their capabilities. The higher the value the more likely services are executed at the edge, so in less powerful nodes which easily saturate their computing resources. On the contrary, lower values of the metric reflect the case in which services are executed by more powerful nodes farther from the IoT domain.

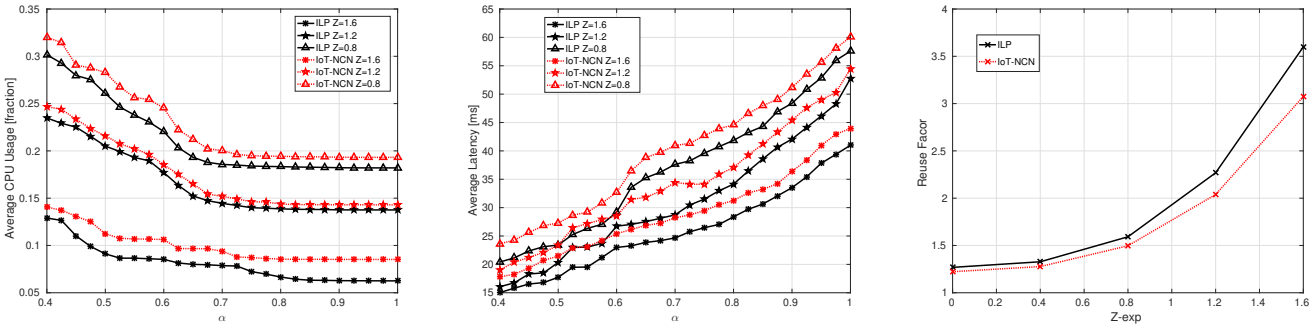
The *average executor-source latency*: it is derived as the latency experienced over the path (set of links) interconnecting the node acting as executor and the IoT data source(s) for each service. Raw data provided by IoT sources are delivered over this path before being processed at the executor. The closer the executor to the data the lower such a latency value. Thus, in case of data-intensive services, it is crucial to choose a low-latency executor-source path.

The *average service reuse factor*: it is defined as the ratio between the overall number of service requests ($|R|$) over the total number of executed services in the IoT-NCN-capable edge nodes. The higher the metric the higher the service reuse, meaning that the same node has served multiple requests from different consumers for the execution of the same service, and it has executed it only once, i.e., the execution always produces the same output given the same input. For a reuse equal to 1, each request entailed a service allocation in a different node, which reflects the low service popularity condition.

Results. Figure 4 compares the optimal (i.e., centralized) and the distributed solutions in terms of the aforementioned metrics.

First, it is evident that the optimal solution represents the best theoretically achievable performance. Differences between the optimal and the distributed solutions can be explained as follows. The distributed solution only aims to allocate a service execution in the node on the consumer-source path which provides the minimum cost function for a given service request. This choice would not necessarily translate into the minimum sum of the cost functions of all considered services.

It is worth remarking that the distributed IoT-NCN approach is practically viable when relying on the plain NDN philosophy, according to which the service orchestration can be performed autonomously by in-network nodes. Furthermore, the distance from the optimal solution performance is accept-



(a) Average CPU usage per node (b) Average executor-source latency (c) Average Reuse Factor ($\alpha = 0.7$)

Fig. 4. Optimal solution (ILP) Vs. the distributed algorithm (IoT-NCN) when varying the α parameter for different Zipf popularity settings (Z).

able for all metrics in Figure 4 and for every value of the α parameter[3].

Looking at the average CPU usage in Figure 4(a), it decreases as α increases; the opposite trend is observed for the executor-source latency in Figure 4(b). Results confirm that the α parameter properly weights the two contributions of the service execution cost in the cost function. In particular, for $\alpha < 0.5$, the computation is moved closer to the IoT domain. This turns into *(i)* higher average CPU usage, since less powerful nodes are involved in the computation; and *(ii)* lower average executor-source latency, being the executor located closer to the IoT sources.

Figure 4 also reports the performance results when varying the service popularity through the Z parameter. In Figure 4(a), the higher the Z parameter the lower the CPU usage per node. Reasonably, the higher service popularity allows the same service to be executed once in a single node, which serves multiple requests. The same behavior is observed in Figure 4(b): the latency decreases as the Z parameter increases, because there is no need to retrieve the IoT data again. As an additional positive side effect, if multiple requests can be served with a single execution then, more resources are available to allocate the services close to the data sources, thus implying a lower latency.

The average service reuse factor shown in Figure 4(c) allows better understanding the impact of service popularity. Results are shown for a fixed α when varying the Z parameter. As the Z parameter increases the reuse factor increases as well, witnessing that a node may satisfy multiple requests (up to 3.5 on average for Z=1.6) for the same popular service by executing it only once.

For more insights into the CPU usage, let us refer to Figure 5 which reports a heatmap of the average CPU usage at each node of the reference network topology, when varying the α parameter for a fixed value of Z. It can be observed that for low α values (Figure 5(a)) the executors are mainly selected among nodes close to the IoT domain, being for them the average CPU usage close to 1 (dark red). As α gets higher (Figure 5(c)), nodes acting as executors are those more powerful and farther from the IoT domain.

[3]Please notice that in the following, the same α applies for all services, thus, α_{s_k} is replaced by α, $\forall s_k \in S$.

VIII. ndnSIM Results

In this Section we use a realistic network deployment thanks to ndnSIM and compare the IoT-NCN framework against two benchmark state-of-the-art solutions. The first one is representative of a legacy mobile edge computing approach (it is labeled as MEC in the plots), according to which all requested services are executed in a *single server* only, which is co-located with the Border Router. The second one is representative of NDN-based approaches in the literature, i.e., [7], [15], according to which services are executed *as close as possible to the IoT data sources*; it is labeled as $aCaP$ (that stands for as close as possible) in the plots.

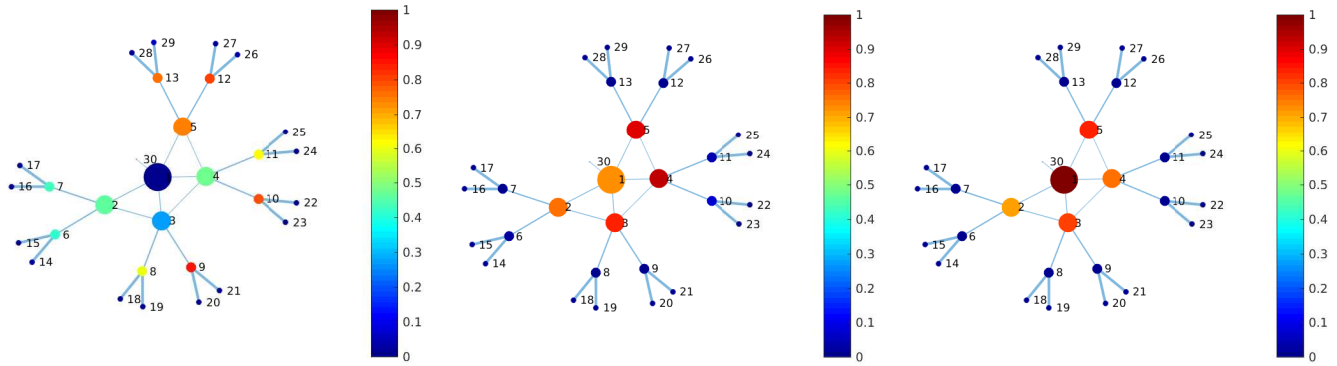
Simulation results are averaged over 20 independent runs and reported with the 95% confidence intervals.

A. Impact of IoT data size

For the first set of results, the Zipf parameter Z is fixed to 0.8, while parameter α and the number of IoT Data packets over which services should be computed vary. Two different workload settings are considered to evaluate the impact of the α parameter and to investigate the capabilities of IoT-NCN to manage services with different requirements. The first one resembles a low workload setting, labeled as LW in the plots, which considers a set of service requests with heterogeneous low CPU demands (i.e., $\alpha \leq 0.5$). The second one mimics a high workload scenario, labeled as HW in the plots, which considers a set of heterogeneous CPU-intensive service requests (i.e., $\alpha > 0.5$).

The following metrics are derived: *(i)* the *average service provisioning time*, computed as the time since the first Interest is transmitted from a consumer to request the service until the output of the service execution is sent back (thus including the computation time); and *(ii)* the *number of transmitted Interest packets* to collect the input raw IoT data for execution of the requested service.

In Figure 6(a) the *average service provisioning time* is reported. Not surprisingly, for all the considered solutions, the metric reasonably increases with the increasing number of Data packets over which the processing should be executed. This is because the latency contribution due to the data retrieval gets higher. This trend is more remarkable when the HW scenario is considered. In such a case, the NDN-based

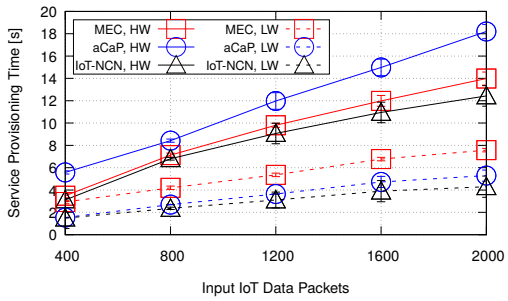


(a) $\alpha = 0.4$ (b) $\alpha = 0.7$ (c) $\alpha = 1$

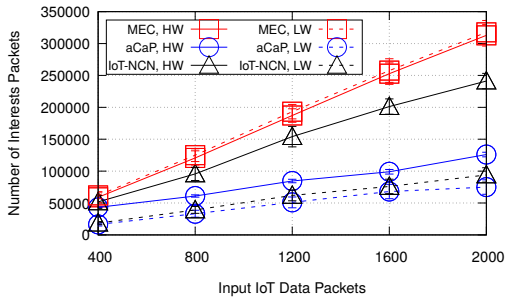
Fig. 5. Heatmap of the average CPU usage at each node when varying the α parameter ($Z = 0.8$).

approaches, aCaP and IoT-NCN, select for service execution more powerful nodes, which are farther from the IoT sources. The proposed scheme outperforms the two benchmarks, under all the considered settings, hence witnessing that the cost function properly reflects the designed trade-off, and the weight parameter flexibly captures heterogeneous service requirements. In particular, the aCaP solution approaches the IoT-NCN proposal under the LW scenario, since they both prefer nodes closer to the data for service execution. The MEC solution, instead, gets close to IoT-NCN under the HW scenario, since CPU-intensive services are better allocated in more powerful nodes, hence close to/into the Border Router.

Figure 6(b) compares the three solutions in terms of number of transmitted Interest packets for IoT data retrieval. Please note that the metric considers the number of transmitted Interests per-link. This means that, to retrieve new (not cached) Data from an IoT source in the considered topology, an Interest sent by the Border Router will be replicated over 4 hops, while an Interest sent by an AR will be transmitted over a single hop. The MEC solution always experiences the highest Interest signaling, since the input content needs to be transferred to the Border Router. The aCaP is the most efficient one, because it places the service execution close to the data sources, hence reducing the number of Interests.



(a) Service provisioning time



(b) Number of transmitted Interest packets

Fig. 6. Performance metrics when varying the number of IoT raw data packets (1kB-large Data packets, 50 service requests).

B. Impact of service popularity

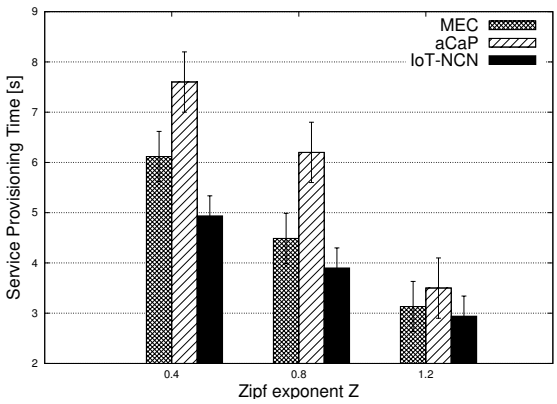
The last simulation campaign is intended to showcase the benefits of IoT-NCN when varying the service popularity, under a heterogeneous workload setting with uniformly distributed CPU-intensive and data-intensive service requests. Specifically, the input content size is uniformly distributed in the range $[200 - 1600]$ packets, the CPU requests of services are uniformly distributed in the range $[25, 375]$ Mcycles and α is computed accordingly.

Results in Figure 7(a) demonstrate that when the service popularity increases, the service provisioning time is reduced. Such a trend can be observed for all the compared schemes. The network traffic load due to Interest packets is also reduced (Figure 7(b)). Indeed, the more popular a service, the more likely this service is already running at the edge, or even already computed; so, there is no need to send Interests to collect novel input IoT raw data for service execution. Under the considered settings, the aCaP scheme exhibits the lowest number of exchanged Interests, which is paid at the expenses of the highest service provisioning time. IoT-NCN always outperforms the benchmarking schemes in terms of service provisioning time, further confirming its capability to adequately treat heterogeneous services.

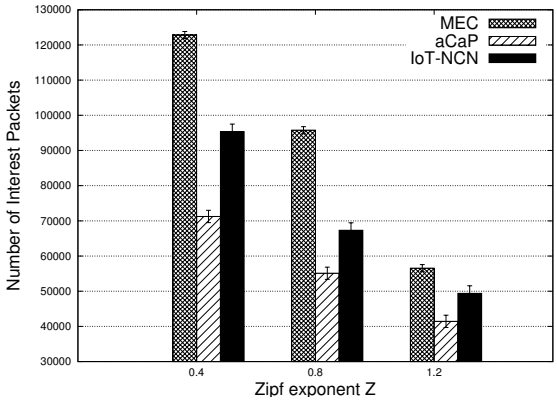
IX. Open Challenges and Research Directions

The topicality of the work and the flexibility of the proposed IoT-NCN framework leave many open challenges as well as options for future research, summarized in the following.

Function migration. Migration strategies could be investigated for future work, which move the service code from a



(a) Service provisioning time



(b) Number of transmitted Interest packets

Fig. 7. Performance metrics when varying the Zipf parameter Z (1kB-large Data packets, 50 service requests).

node to another one in order to cope with changing service demands and the mobility of consumers and data sources.

Beyond IoT services. The proposed framework could be extended to manage more generic services besides the ones related to IoT data processing investigated in this paper. This is viable thanks to the semantically-rich naming and the flexibility of the conceived service allocation strategies.

Service allocation policy. The devised strategy trades-off between the quality perceived by the consumers, i.e., in terms of service provisioning time, and the amount of raw traffic traversing the edge domain. Other policies could be easily integrated in the conceived framework, which could be completely *operator-centric*, for instance if targeting computing load balancing and reduction of the amount of exchanged traffic.

Signalling mechanisms. Alternatives to the use of persistent Interests is also left for future work. Despite the simplicity of the mechanism, persistent Interests need to be stored in the PIT for all the service provisioning time, thus they could impact the PIT scalability. To limit this issue, PIT compression methods could be considered, such as the ones based on Bloom filters [26], or novel signaling mechanisms could be deployed to release the PIT during the service execution time. For instance, the consumer can send a regular Interest to request the service and the selected executor can acknowledge it with a *s-ACK* packet that carries also the estimated service execution time. At the reception of the s-ACK, the intermediate nodes delete the correspondent PIT entry; the consumer, instead, sets up a timer and sends a new Interest to retrieve the result after the execution time.

Security. NDN natively secures Data at the network layer but, when dealing with in-network computations, other security mechanisms must be considered. First, before the computation, an executor should verify that the service is requested by an authorized and authenticated consumer. To this purpose, signed Interests have been introduced in the literature [14], where requesters sign Interests using their private key. This not only prevents that unauthorized users access computation resources, but generally it could limit denial-of-service attacks. So far, however, signed Interests have been considered in small IoT environments, e.g., in a building automation scenario, and their use on a network domain scale has not been properly investigated. Second, since the computation is performed over an input content, the security property of this latter (e.g., access control) should be extended to the result. Also, if the computation must preserve the user's privacy, input data must be encrypted and the user's information, e.g., its location, must be protected. Which encryption algorithm is more convenient in this case and - generally - how to preserve the user's privacy and the access control are open challenges, although some interesting findings are reported in [27].
Third, since the result of a computation is in practice a new content, its integrity and authenticity must be verified by the user. Being the actual producer of the result the executor node, it could be convenient to assign a private key to each executor and let it sign the result. Alternatively, as proposed in [28], the result could be signed by the "function" itself, providing that it has access to the key of the original function provider/developer. Moreover, if the output is a private content, an encryption mechanism is needed.

X. Conclusions

In this paper, we proposed IoT-NCN, a novel framework for IoT data processing at the network edge, which leverages the NDN paradigm for data retrieval from IoT sources and overhauls it to encompass a distributed service allocation procedure. The designed strategy selects as service executor the edge node which ensures to minimize a weighted cost function balancing between two objectives: *(i)* limiting the raw IoT data traffic across the network, and *(ii)* allocating the execution based on available nodes' processing resources.

Achieved numerical results confirm that the proposed distributed approach is not so far from an optimal solution, formulated through an ILP problem and minimizing the same cost function. Realistic system-level simulations show the superiority of the proposal in terms of service provisioning time, under different workload settings, against state-of-the-art alternative solutions. Such improvement is achieved at the expenses of a higher signaling load to retrieve input Data packets by the service executors.

References

[1] M. Kovatsch, S. Mayer, and B. Ostermaier, "Moving application logic from the firmware to the cloud: Towards the thin server architecture for

the internet of things," in *IEEE Innovative Mobile and Internet Services in Ubiquitous Computing (IMIS)*, Palermo, Italy, 2012, pp. 751–756.
[2] A. G. Tasiopoulos, O. Ascigil, I. Psaras, and G. Pavlou, "Edge-map: Auction markets for edge resource provisioning," in *IEEE 19th International Symposium on A World of Wireless, Mobile and Multimedia Networks(WoWMoM)*, Chania, Greece, 2018, pp. 14–22.
[3] C. Scherb, B. Faludi, and C. Tschudin, "Execution state management in Named Function Networking," in *IFIP Networking Conference (IFIP Networking) and Workshops*, Stockholm, Sweden, 2017, pp. 1–6.
[4] M. Sifalakis, B. Kohler, C. Scherb, and C. Tschudin, "An information centric network for computing the distribution of computations," in *ACM International Conference on Information-Centric Networking (ICN)*, Paris, France, 2014, pp. 137–146.
[5] M. Król and I. Psaras, "NFaaS: named function as a service," in *ACM Conference on Information-Centric Networking (ICN)*, Berlin, Germany, 2017, pp. 134–144.
[6] Q. Wang, B. Lee, N. Murray, and Y. Qiao, "CS-Man: Computation service management for IoT in-network processing," in *IEEE Signals and Systems Conference (ISSC)*, Derry, North. Ireland, 2016, pp. 1–6.
[7] M. Amadeo, C. Campolo, A. Molinaro, and G. Ruggeri, "IoT data processing via Named Data Networking," in *European Wireless conference*, Catania, Italy, 2018.
[8] M. Amadeo, C. Campolo, and A. Molinaro, "NDNe: Enhancing named data networking to support cloudification at the edge," *IEEE Communications Letters*, vol. 20, no. 11, pp. 2264–2267, 2016.
[9] S. Mastorakis, A. Afanasyev, I. Moiseenko, and L. Zhang, "ndnSIM 2.0: A new version of the NDN simulator for NS-3," *NDN, Technical Report NDN-0028*, 2015.
[10] A. Afanasyev *et al.*, "NFD developer's guide," NDN Project, Tech. Rep. Tech. Report NDN-0021, 2016.
[11] M. Amadeo *et al.*, "Information-centric networking for the internet of things: challenges and opportunities," *IEEE Network*, vol. 30, no. 2, pp. 92–100, 2016.
[12] O. Hahm, E. Baccelli, T. Schmidt, M. Wählisch, C. Adjih, and L. Massoulié, "Low-power internet of things with NDN & cooperative caching," in *ACM Conference on Information-Centric Networking (ICN)*, Berlin, Germany, 2017, pp. 98–108.
[13] M. Amadeo, C. Campolo, A. Iera, and A. Molinaro, "Information centric networking in IoT scenarios: The case of a smart home," in *IEEE International Conference on Communications (ICC)*, London, UK, 2015, pp. 648–653.
[14] J. Burke, P. Gasti, N. Nathan, and G. Tsudik, "Securing instrumented environments over content-centric networking: the case of lighting control and ndn," in *IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, Turin, Italy, 2013, pp. 394–398.
[15] C. Scherb, D. Grewe, M. Wagner, and C. Tschudin, "Resolution strategies for networking the iot at the edge via named functions," in *IEEE Conference on Consumer Communications & Networking (CCNC)*, Las Vegas, USA, 2018, pp. 1–6.
[16] O. Ascigil, S. Reñé, G. Xylomenos, I. Psaras, and G. Pavlou, "A keyword-based ICN-IoT platform," in *ACM Conference on Information-Centric Networking (ICN)*, Berlin, Germany, 2017, pp. 22–28.
[17] Y. Mao, C. You, J. Zhang, K. Huang, and K. B. Letaief, "A survey on mobile edge computing: The communication perspective," *IEEE Communications Surveys & Tutorials*, vol. 19, no. 4, pp. 2322–2358, 2017.
[18] M. S. Elbamby, M. Bennis, and W. Saad, "Proactive edge computing in latency-constrained fog networks," in *IEEE European Conference on Networks and Communications (EuCNC)*, Oulu, Finland, 2017, pp. 1–6.
[19] D. Alessandrelli, M. Petraccay, and P. Pagano, "T-res: Enabling reconfigurable in-network processing in IoT-based WSNs," in *IEEE Distributed Computing in Sensor Systems (DCOSS)*, Cambridge, Massachusetts, USA, 2013, pp. 337–344.
[20] P. Moll, D. Posch, and H. Hellwagner, "Investigation of push-based traffic for conversational services in named data networking," in *IEEE International Conference on Multimedia & Expo Workshops (ICMEW)*, Hong Kong, China, 2017, pp. 315–320.
[21] M. Amadeo, C. Campolo, and A. Molinaro, "Multi-source data retrieval in IoT via Named Data Networking," in *ACM International Conference on Information-centric networking (ICN)*, Paris, France, 2014, pp. 67–76.
[22] B. Yang, W. K. Chai, Z. Xu, K. V. Katsaros, and G. Pavlou, "Cost-efficient NFV-enabled mobile edge-cloud for low latency mobile applications," *IEEE Transactions on Network and Service Management*, vol. 15, no. 1, pp. 475–488, 2018.
[23] K. P. Yoon and C.-L. Hwang, *Multiple attribute decision making: an introduction*. Sage publications, 1995, vol. 104.
[24] H. Topcuoglu, S. Hariri, and M.-Y. Wu, "Task scheduling algorithms for heterogeneous processors," in *IEEE Heterogeneous Computing Workshop (HCW)*, San Juan, Puerto Rico, 1999, pp. 3–14.
[25] I. Psaras, L. Saino, M. Arumaithurai, K. Ramakrishnan, and G. Pavlou, "Name-based replication priorities in disaster cases," in *IEEE INFOCOM Workshops*, Toronto, Canada, 2014, pp. 434–439.
[26] Z. Li, J. Bi, S. Wang, and X. Jiang, "Compression of Pending Interest Table with Bloom Filter in content centric network," in *Int. Conference on Future Internet Technologies*, Seoul, Republic of Korea, 2012.
[27] C. Marxer, C. Scherb, and C. Tschudin, "Access-controlled in-network processing of named data," in *ACM Conference on Information-Centric Networking (ICN)*, Kyoto, Japan, 2016, pp. 77–82.
[28] M. Król, M. Claudio, D. Grewe, I. Psaras, and C. Tschudin, "Open security issues for edge named function environments," *IEEE Communications Magazine*, vol. 56, no. 11, pp. 69–75, 2018.

Marica Amadeo is a Postdoc researcher 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 and edge computing.

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.

Claudia Campolo is an assistant 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. 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 Telesoft, Rome (1992-1993) and Siemens, Munich (1994-1995). Her current research focuses on 5G, vehicular networking and future Internet architectures.