Fog Computing in IoT Smart Environments via Named Data Networking: A Study on Service Orchestration Mechanisms

Marica Amadeo, Giuseppe Ruggeri, Claudia Campolo, Antonella Molinaro, Valeria Loscrì, Carlos Tavares Calafate

future internet

MDPI

Article

Fog Computing in IoT Smart Environments via Named Data Networking: A Study on Service Orchestration Mechanisms

Marica Amadeo [1,*], Giuseppe Ruggeri [1], Claudia Campolo [1], Antonella Molinaro [1,2], Valeria Loscrí [3] and Carlos T. Calafate [4]

1 DIIES Department, University Mediterranea of Reggio Calabria, Via Graziella, Loc. Feo di Vito, 89100 Reggio Calabria, Italy; giuseppe.ruggeri@unirc.it (G.R.); claudia.campolo@unirc.it (C.C.); antonella.molinaro@unirc.it (A.M.)

2 Laboratoire des Signaux et Systémes (L2S), CentraleSupélec, Université Paris-Saclay, 91190 Gif-sur-Yvette, France

3 Inria Lille-Nord Europe/FUN, 59650 Villeneuve D'Ascq, France; valeria.loscri@inria.fr

4 Department of Computer Engineering (DISCA), Universitat Politècnica de València, 46022 València, Spain; calafate@disca.upv.es

* Correspondence: marica.amadeo@unirc.it; Tel.:+39-0965-1693276

Received: 24 September 2019; Accepted: 21 October 2019; Published: 24 October 2019

Abstract: By offering low-latency and context-aware services, fog computing will have a peculiar role in the deployment of Internet of Things (IoT) applications for smart environments. Unlike the conventional remote cloud, for which consolidated architectures and deployment options exist, many design and implementation aspects remain open when considering the latest fog computing paradigm. In this paper, we focus on the problems of dynamically discovering the processing and storage resources distributed among fog nodes and, accordingly, orchestrating them for the provisioning of IoT services for smart environments. In particular, we show how these functionalities can be effectively supported by the revolutionary Named Data Networking (NDN) paradigm. Originally conceived to support named content delivery, NDN can be extended to request and provide named computation services, with NDN nodes acting as both content routers and in-network service executors. To substantiate our analysis, we present an NDN fog computing framework with focus on a smart campus scenario, where the execution of IoT services is dynamically orchestrated and performed by NDN nodes in a distributed fashion. A simulation campaign in ndnSIM, the reference network simulator of the NDN research community, is also presented to assess the performance of our proposal against state-of-the-art solutions. Results confirm the superiority of the proposal in terms of service provisioning time, paid at the expenses of a slightly higher amount of traffic exchanged among fog nodes.

Keywords: Internet of Things; named data networking; information centric networking; fog computing; smart environments; service orchestration

1. Introduction

Internet of Things (IoT) promotes the effective integration of the real world and the digital world by allowing objects of everyday life to be connected everywhere at anytime, to interact with each other, and to exchange data and knowledge [1]. Thanks to this paradigm, the environments where we live, work, and play will turn into smart user-friendly ecosystems, where billions of heterogeneous devices, including monitoring sensors, actuators, home appliances, vehicles, and smartphones, are seamlessly integrated [2].

Smart home, smart building, smart transportation, and smart energy are just a few examples of IoT application domains. In these contexts, the cloud and fog computing paradigms will play a crucial role in supporting the variety of expected services (e.g., data aggregation, filtering, manipulation for monitoring, and surveillance purposes) and in fulfilling their requirements in the presence of a huge number of heterogeneous connected devices. According to References [3,4], fog and cloud computing will complement each other to form a service continuum between the cloud and the IoT devices by providing computing, storage, control, and networking resources.

On the one side, by leveraging remote data centers and centralized control, the cloud becomes the perfect candidate for managing long-term data storage and for executing complex analytics algorithms. On the other side, the fog brings computing and storage services close to the end-users, where data are generated and/or consumed. The advantages are clear: enabling interactive and delay-sensitive applications, reducing the amount of traffic reaching the core network segment, and guaranteeing high performance also in the presence of limited network bandwidth and intermittent connectivity. In the most ambitious scenario, network edge routers and hosts, ranging from access points, backhaul nodes, and Points of Presence (PoPs) to IoT gateways and user devices like smartphones and tablets, will turn into fog elements that execute services on demand.

An enabler of such revolutionary vision that integrates computing and networking functionalities is Named Data Networking [5]. Originally conceived as an architecture for the future Internet, NDN connects heterogeneous devices, ranging from IoT sensors to cloud servers, by naming data bits at the network layer instead of using IP addresses. A consequence of this design choice is that NDN can also name computation services [6,7] and implement in-network caching. As a result, clients (the so-called consumers) can request a service by name and the network can find an in-network service executor or even a cached result if the same service has already been computed. By doing so, NDN decouples the service from the identity of the node able to execute it, turning the network edge into a provider-agnostic in-network computing framework.

A few works have been already published that extend NDN to support in-network processing [6–8], and the interest in this research field is steadily growing. Although existing approaches differ in the implementation details, their basic idea is to let potentially any NDN node download and execute a *named processing function*, i.e., a software that takes as input a content, processes it, and returns a result. Contents can be already available in the network or generated on demand by servers, by IoT sources, or by the clients themselves. According to the consumers' demands and the nodes' capabilities, functions can be dynamically migrated in the network or removed if they are not required. Of course, a major challenge in this scenario is the design of effective orchestration mechanisms for allocating the services in the best available nodes.

Different decision criteria can be explored in this context. For instance, an intuitive mechanism could be to execute the service as close as possible to the place where data are produced in order to reduce the data retrieval latency [6]. Another option could be to execute the service in the less-loaded node in order to reduce the processing latency [9,10]. An alternative strategy could be to select the executor according to the service requirements [7,8], e.g., compute-intensive services could be deployed in more powerful nodes. With the decisions taken in a distributed way by nodes with limited and time-varying capabilities, implementing even the simplest decision criterion is not trivial; thus, the impact on the final performance could be huge.

In this paper, we explore the NDN fog computing paradigm in a smart environment scenario, with a special focus on the deployment of service orchestration routines. We consider an edge network domain deployed in a smart campus, where a set of NDN IoT devices are installed that produces environmental data, e.g., temperature, humidity, and luminance values from various sensors, and images and videos taken from cameras. NDN fog nodes can retrieve and process IoT data according to the requests they receive from consumers.

In this context, the main original contributions of this work can be summarized as follows:

- We extend the NDN architecture to support service provisioning in the campus network and propose a new forwarding strategy that supports the decision of whether the IoT data processing has to be performed into fog nodes or remotely in the cloud in a fully distributed manner. Hence, the strategy is aligned with the fog computing concept.
- We conceive a service executor strategy, leveraging a bidding procedure that aims at minimizing the overall service provisioning time, accounting for the retrieval of input IoT data to be processed and for the execution of this processing.
- We implement the devised strategy in ndnSIM [11] and compare its performance through valuable metrics against a set of benchmark orchestration mechanisms available in the literature. The evaluation shows that our proposal outperforms the alternative benchmarking schemes by better scaling as the number of service requests increases. The achieved improvements come at the expense of a slightly higher amount of traffic exchanged among fog nodes.

The rest of the paper is organized as follows. Section 2 provides an overview of the NDN paradigm, notably its suitability for IoT content retrieval in smart environments. The potential of NDN as a key enabler of fog computing is discussed in Section 3, whereas the relevant literature is discussed in Section 4. The proposed framework is presented in Section 5 and evaluated in Section 6. Finally, conclusions are presented in Section 7.

2. Named Data Networking and Its Role in IoT Smart Environments

2.1. NDN in a Nutshell

NDN is an Information Centric Networking (ICN) architecture that turns content into a first-class network citizen [5,12,13]. Unlike the current Internet, where communication is based on named hosts, in NDN, each content is uniquely named and secured at the network layer. Routing and forwarding operations are directly based on content names, and in-network caching is natively enabled. Together with other innovative paradigms, like edge and fog computing, NDN has been recognized as a fundamental pillar of the future Internet [14] that can offer highly scalable and efficient content dissemination in different network scenarios, ranging from the wired Internet to wireless ad hoc environments and IoT systems. Indeed, NDN defines a connectionless communication model extremely suited to IoT environments.

Basically, NDN communication leverages only two packet types: the *Interest*, used to request contents, and the *Data*, used to carry the content. Both packets carry a hierarchical name that uniquely identifies a piece of data. Names are in the form of Uniform Resource Locators (URLs), and they can be human-readable and user-friendly. Name prefixes are distributed by the network routing protocols to enable the reachability of contents.

Three main communication actors can be identified: (i) the *consumer*, which sends Interests to request contents; (ii) the original *producer*, which originates the data packets; and (iii) the *content router*, which forwards interests and data packets and may cache the latter to satisfy future requests. Data packets are signed at the time of creation by their producers. This allows to decouple the trust in the data from the trust in the nodes that may cache them. In the following, producers and cachers of contents are both referred to as providers.

Being a networking solution, NDN operates at two levels: the data plane and the control plane, as detailed in the following subsections.

2.1.1. Data Plane

As shown in Figure 1, each NDN node maintains three tables at the forwarding plane (also known as, the data plane).

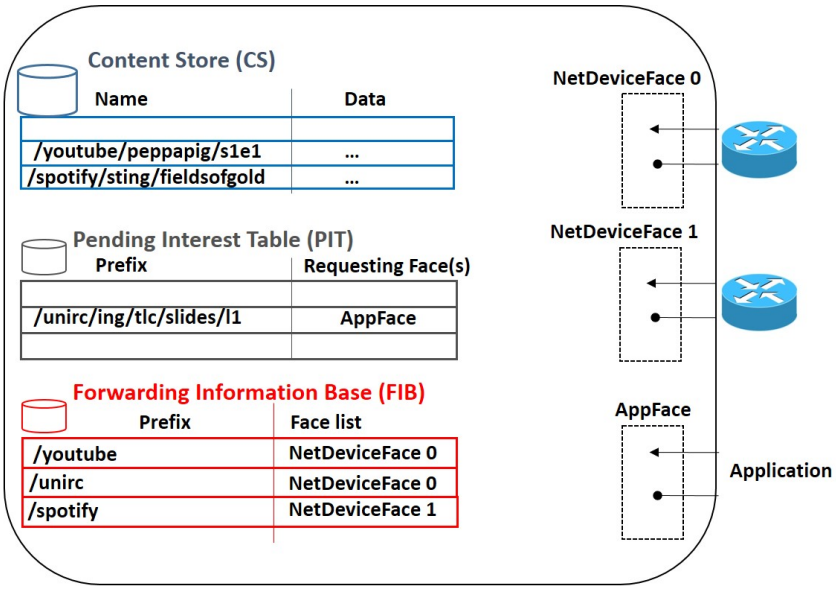


Figure 1. Named Data Networking (NDN) node's data plane.

The Content Store (CS) is used for caching incoming data packets. Depending on the scenario and the available storage resources at the node, several decision policies can be implemented, ranging from the simple caching-everything-everywhere strategy to the probabilistic caching and the more advanced popularity-based autonomous or cooperative mechanisms [15]. The caching policy is always coupled with a replacement strategy that is applied when the storage space is full and a new item must be cached. The default replacement policy in NDN is Least Recently Used (LRU).

The Pending Interest Table (PIT) tracks the transmitted interests together with the incoming/outgoing interfaces. This forwarding state is maintained until the data packets consume the interests, a lifetime expires, or a negative acknowledgement (NACK) is received to advertise an error that occurred along the path.

The Forwarding Information Base (FIB) maintains the outgoing interface(s) per named prefix. Multiple interfaces may exist per prefix, ranked by routing preferences. Interfaces are called *faces* in the NDN context. In particular, two face abstractions are identified: the *NetDeviceFace*, which enables communication with the underlying physical networks, and the *AppFace*, which enables communication with the applications.

2.1.2. Forwarding Fabric

Only interests are routed in NDN [16]. Specifically, at the interest reception (from the application layer or from the underlying access network layer), each node runs the following processing:

- First, the node looks in the CS. If a matching is found, the data packet is immediately sent over the same interface that the interest arrives from.
- Otherwise, the node looks in the PIT. If a match is found, it means that the same request has already been forwarded, the node is waiting for the data, and there is no need to transmit it again. As a result, the interest is discarded and the node only updates the PIT entry with the incoming interface of the received interest. Such a feature, typically referred to as request aggregation, reduces the number of accesses to the original content source.
- If also the PIT matching fails, the node looks in the FIB to forward the interest over an outgoing interface. In case of failure, a NACK can be sent back and the interest is discarded.

Data packets are forwarded over the same interfaces that the interests were received, according to the PIT entry.

2.1.3. Control Plane

The NDN control plane is in charge of the routing decisions and includes the *Routing Information Base* (RIB) and the *Strategy Choice Table* (SCT). RIB entries are originated by manual configuration from administrators, applications, and routing protocols and are used to update the FIB. The SCT, instead, identifies the forwarding strategy per namespace; it can be updated by management protocols or administrators, and it is supposed to be quite stable. The forwarding strategy is responsible for deciding if it should, on which face to, and when to forward NDN packets.

2.2. NDN in Smart Environments

The potential of NDN in IoT smart environments has been widely recognized in the literature [17,18]. Indeed, NDN well matches the pattern of many IoT applications that focus on the data itself (e.g., the temperature in a room and the average speed of vehicles in a road segment) rather than on a specific data producer. The NDN interest/data exchange can be used to support a variety of monitoring services by directly using application-level names [19]: requests for monitored parameters can be carried in interest packets; at the reception of the interest, a provider can send the data back. In-network caching and request aggregation natively reduce the traffic load and the number of accesses to the original producer; this is especially useful when providers are resource-constrained IoT devices.

NDN communication can also support actuation services, e.g., to switch on/off lights [20,21]: in this case, specific named commands can be carried in the interest packets that are forwarded towards the things in charge of executing them. A data packet can be sent back to acknowledge the command execution.

According to References [17,19], NDN names for smart environments can be conveniently structured in three main parts. First, a main prefix should be identified that specifically refers to the IoT domain, e.g., */unirc*. The main prefix can be advertised by the routing protocol and maintained in the FIBs of network core nodes to allow the reachability of the IoT domain from remote consumers. If the IoT domain is large, e.g., a university campus, other subcomponents of the domain name can be specified, e.g., */eng/buildingA/telcolab* identifies a specific laboratory in a given building of a certain faculty. Second, a set of middle components can be defined to identify the IoT data (e.g., */temperature*, the thing, or the actuation command (e.g., */light/on*). Finally, a last optional component can identify the specific instance of the data, e.g., /2019031430. Therefore a name like */unirc/eng/buildingA/telcolab/temperature*/2019031430 uniquely identifies a temperature value sensed on 3 March 2019 at 14:40 in the Telco Laboratory of the Engineering Faculty at the University of Reggio Calabria.

Last but not least, NDN communication in smart environments can leverage a strong per-packet authentication and integrity support involving both interest and data packets. Data packets are always signed by their original producers, and in the case of sensitive actuation services, the interest packets can be signed by the consumers to ensure that certain actions are requested only by authorized authenticated entities [20]. Public key cryptography is usually the reference mechanism for signing NDN packets [22]. However, mechanisms based on symmetric cryptography are preferred for resource-constrained IoT devices [23].

To ensure data confidentiality and access control when dealing with sensitive IoT services, NDN uses encryption and requires an automated key management system to distribute the session encryption and decryption keys to the involved entities [24].

3. NDN: The Networking Paradigm for Fog Computing

3.1. Fog Computing: Key Pillars and Issues

The fog computing paradigm has been initially proposed in the pioneering work of Bonomi et al. [25], was highly promoted by Cisco [26], and was further refined by the industrial and academic communities [4] to enable computing, storage, and networking close to the end-user.

End-user devices or near-user edge devices can be leveraged to carry out communication and computation in different IoT domains, such as industrial plants [27] and vehicular environments [28,29].

Other similar paradigms to fog computing such as edge computing, mist computing, and cloudlets have been proposed by the research community with similar purposes [30]. Despite the similarities in terms of targeted objectives, the OpenFog consortium [4], bringing together industry companies and academic institutions across the world for the sake of standardization and promotion of fog computing, recognizes a more comprehensive scope and flexibility of the fog in that it seeks to enable computing services anywhere from cloud to things, as data typically moves from the IoT devices to the processing nodes. This is a clear departure from many existing solutions that treat network edges as isolated computing platforms and are not inclusive of the cloud.

Such a kind of deployment particularly challenges service addressing; discovery and provisioning; and management aspects, like orchestration of services and network resources [30,31].

Requests for a computing service (i.e., the execution of a given processing function over a given content) are issued regardless of the position (address) of the executor node, of which the identity is not a priori known to the requester. A service may reside on a number of local nodes, or it may partially reside on local nodes and on the cloud. Conventional addressing schemes are not suited to match the service-centric nature of the targeted context.

Discovery procedures are typically performed with the help of an application-layer broker mechanism [32,33] that matches the service request with the best available node that is able to execute the service based on specified requirements and discovered capabilities. These broker entities, however, further complicate the reference scenario.

3.2. NDN as Enabler of Fog Computing

Recently, NDN has been considered an enabler of edge and fog computing because it can support provider-agnostic in-network computing and caching services [34,35]. According to this new vision, interest packets can be overhauled to let a consumer request by name not only a content but also a specific computation service. Such named service requests can be processed directly at the network layer. Services can be executed by any node in the network that owns the required capabilities. The named service can process contents already available in-network (i.e., in the CS of a router) or produced on demand by another source or even produced by the consumer itself. Data packets can be overhauled to carry computation results that can be stored in the CS of NDN nodes to satisfy future requests. This is a radical departure from IP-based fog computing approaches, which usually rely on specific entities to resolve an application-layer request into the IP address of a node able to perform the computation [34].

Besides overhauling interest and data packets, further modifications are required to let NDN support in-network computation. So far, the related literature has focused on the following main design aspects:

- *Naming schemes* for identifying both contents and services [6,36,37].
- *Discovery schemes* for allowing NDN nodes to identify services and resources supported by their neighbours, e.g., CPU, energy, and storage [8,34].
- *Service orchestration* strategies to identify the best in-network service executors [6–10].

- *Security support* in order (i) to authenticate the consumers asking for a service and (ii) to allow the executor to sign the computation result to make it verifiable by the consumer [37,38].
- *Privacy-preserving schemes* ensuring that results of computation inherit the same confidentiality of the input contents [39].
- *Forwarding schemes* for managing long and/or interactive computation [37] and for managing live stream processing [40].

In this paper, we consider a much less investigated topic, which is the problem of IoT service orchestration via NDN in a fog context. Basically, given a service request with known requirements, such as the associated CPU demand and processing deadline, the network has to find an NDN node that can perform the service matching its requirements. This is a challenging objective since NDN nodes have usually limited and rapidly changing computing and storage resources and the allocation decision should be taken dynamically. Ad hoc designed mechanisms, closely tied to the NDN forwarding fabric, are therefore required that are different from the traditional cloud resource orchestration strategies conceived so far [41].

4. IoT Service Orchestration via NDN

In this section, we review the IoT service orchestration mechanisms via NDN available in the literature by identifying their features together with their pros and cons. In order to match the NDN mechanisms and to make the best of them, the majority of the proposals include a distributed service orchestration, according to which service allocation decision is taken in the network. However, there are also a few examples of centralized designs, which we report for the sake of completeness. Scanned literature works are summarized in Table 1.

Table 1. IoT service orchestration mechanisms in NDN.

Mechanism	Type	Short description
NFN FoX [6]	Distributed	First, it tries to find a cached result on the path towards the data source. In case of failure, the computation is performed as close as possible to the data source or it is pushed into a node/server off-path.
NFN EdgeFox [10]	Distributed	Customizes FoX for edge domains, where nodes advertise the availability of the functions they provide, and they are queried for the result.
NFN FaX [10]	Distributed	The computation is started in the first available edge node, and it is stopped if a cached result is found in the meanwhile.
IoT-NCN [7]	Distributed	The cost of IoT service execution is computed by on-path nodes, according to a weighted function that takes as input the closeness cost and the processing cost. The node with the lowest cost is selected as the executor.
NDN edge computing [34]	Distributed	The less congested node is selected as the executor.
CS-Man [42,43]	Centralized	A service manager is in charge of identifying the executors of IoT services according to their capabilities.
ICN-isapiens [44,45]	Centralized	A software component called deployer is in charge of loading the IoT services in specific fog nodes depending on the physical IoT devices they control.

4.1. Distributed Mechanisms

A pioneering work extending NDN to support distributed in-network computations, with a special focus on IoT at the edge, is Named Function Networking (NFN) [6,10,46]. In NFN, a name can represent a content, a function capable of processing the content, or an expression combining both.

The NDN forwarding engine is augmented with a λ-expression resolution system, which processes all interests that have the postfix name component /NFN. Interest packets that do not carry the /NFN tag in the name field are treated as legacy requests for contents. A generic NFN name consists of the following components:

$$[ccn : nfn| \quad /name - of - content| \quad /name - of - function]$$

For instance, a client requesting a video transcoded in a certain format can send an interest with the following NFN name [6]:

$$[ccn : nfn| \quad /name - of - video| \quad /name - of - transconder]$$

The default NFN service orchestration strategy is called Find-or-Execute (FoX). First, the network tries to find a cached copy of the result, i.e., the transcoded video in the above example. Therefore, the request is forwarded towards the data producer, on path, according to the component $/name - of - media$. If this search fails, FoX tries to compute the result in the NFN node n closer to the data producer. If not locally available, this node has to retrieve the content and the function code by sending separate interests for $/name - of - media$ and $/name - of - transconder$. If n is not able to perform the service, it can push the computation towards another on-path node or even towards an off-path node, e.g., a cloud facility. In Reference [10], FoX is extended for orchestrating IoT services at the edge. The resulting EdgeFoX strategy assumes that nodes at the network edge are NFN capable and that they may advertise the name of the functions they are able to execute. Similarly to FoX, the interest is first forwarded towards the data source to find a cached result. In case of failure, the interest is forwarded to the NFN node owning the function to check for the result there. If this search also fails, then the result is computed in the first available edge node, which should retrieve data and function codes.

The main problem with FoX and EdgeFox is a potential long service provisioning time due to the different searching steps before starting the computation. This could negatively affect latency-sensitive services. Therefore, for the latter ones, to speed up the provisioning at the expenses of a larger overhead, FoX is enhanced with the Find-and-Execute (FaX) strategy. In FaX, the computation is started immediately in the first available edge node and, in parallel, the interest is forwarded towards the data source to find a cached result. If the search is successful, the computation is stopped and the result is forwarded to the consumer.

Although extremely promising in terms of targeted objectives and envisioned workflows, the mentioned orchestration strategies in NFN have been only theoretically discussed so far and a comprehensive evaluation of their performance in a realistic network topology is still work in progress. A free software implementation of NFN, with Berkeley-style licence, is under development at the University of Basel and it is available at www.named-function.net.

The work in Reference [34] shows the opportunities of enabling edge computing over NDN. It assumes that fog nodes advertise the services they support and their resource utilization (e.g., CPU, GPU, memory, storage, and energy). Such information can be shared periodically in a proactive way by using an NDN routing protocol like Named-Data Link State Routing Protocol (NLSR) [47] or in a reactive way at the reception of the request. The node with the lowest resource utilization can be selected as executor. Such an approach targets well the fair distribution of computation efforts among nodes in the edge domain. However, it accounts neither for the closeness to the raw data to be processed nor for the time needed to collect them.

In Reference [7], a more sophisticated IoT service orchestration strategy called IoT Named Computation Networking (IoT-NCN) is proposed. There, service execution is assigned to the edge nodes in order (i) to limit the raw IoT data traffic crossing the network and (ii) to not exceed their available processing resources. A weighted function is defined to compute the cost of service execution at the edge nodes on the path towards the data source, which well balances the aforementioned objectives. Two costs are included in the function: the closeness cost, expressing the distance of the potential executor to the data source, and the processing cost, expressing the processing capability of

the potential executor with respect to the computing load required by the service. There, the candidate service executors are limited to the on-path nodes, and the one offering the lowest cost is selected. The possibility to resort to the cloud is not investigated.

4.2. Centralized Mechanisms

In Reference [42], a centralized solution for service orchestration in NFN is provided that is called Computation Service Management (CS-Man). The authors consider a static IoT network where a service manager assigns tasks to other nodes according to their capabilities, as recorded in a database called Service Repository. Despite the advantage of leveraging a centralized knowledge of the nodes' capabilities, in such a design, the service manager represents a single point of failure and a potential bottleneck for the provisioning of services, as it is queried at every service request.

In References [44,45], a 3-level framework called ICN-isapiens is presented, where a fog layer, consisting of smart home servers (HSs), is introduced between the physical world and the remote cloud to support real-time services and to hide the heterogeneity of IoT devices. In ICN-isapiens, HSs are installed in houses and provided with a multi-agent software application to manage predefined low-latency monitoring and actuation services. Long-term storage and complex analysis are, instead, performed in the cloud. Each HS leverages NDN primitives to control a predefined set of IoT devices, while communication with the cloud is performed via IP. The application developers define the association between HS, IoT devices, and agents. Then, a system component called deployer is in charge to load the agents in the designed HSs and to start the system. As a result, ICN-isapiens depend on a centralized component that assigns the agents and, therefore, the services to specific nodes and in a static fashion. Hence, it may lack the flexibility required to properly manage a dynamic fog environment.

4.3. Contribution

In this paper, we consider an NDN architecture for fog services provisioning in a smart campus scenario. Such a choice is to provide insights into the benefits of the proposal on a small scale but also a highly representative IoT smart environment, where the collection and processing of measurements provided by heterogeneous entities, such as consumer devices and building facilities, may be frequent. Unlike the works in References [42–45], which promoted a centralized service orchestration mechanism, we deploy a fully distributed strategy, more aligned with the NDN forwarding fabric.

Moreover, unlike the previously discussed works on distributed service orchestration [6,7,10,34], which mainly focused on an NDN edge scenario, here, we consider the interactions between the edge nodes and the cloud computing facilities, in alignment with the cloud-to-things continuum concept of fog computing. In the conceived framework, NDN nodes first try to execute computing services along the path towards the data providers and, in case of failure, they delegate the services to the cloud or to a node in the path towards it in a transparent manner for the consumer. A novel forwarding strategy is defined to support such a process, which requires minor modifications in the NDN node's architecture. The service executor selection policy targets the minimization of the service provisioning time, accounting for the data retrieval time and the processing time.

5. IoT Service Provisioning via NDN in the Smart Campus

5.1. Reference Scenario and Service Naming

Our reference scenario is shown in Figure 2, which depicts four distinct 4-floor buildings composing a university campus. This is a smart environment where multiple NDN IoT devices are installed that sense and monitor the environment and produce a variety of data. They can range from simple environmental data, such as temperature, humidity, movements, and luminance values, to larger and complex information, such as video or audio files captured by cameras in specific areas of the buildings.

Without loss of generality, in the following, we define as IoT service an operation that takes as input the raw data generated by the NDN IoT devices and processes them according to a certain computing function. The output of the computation is generally referred to as service *result*. It can be composed of a single data packet, e.g., a float value indicating the average temperature in a room or a boolean value indicating if there are movements in an area, or multiple data packets, e.g., a compressed video or a dataset indicating the average, maximum, and minimum humidity values in a room.

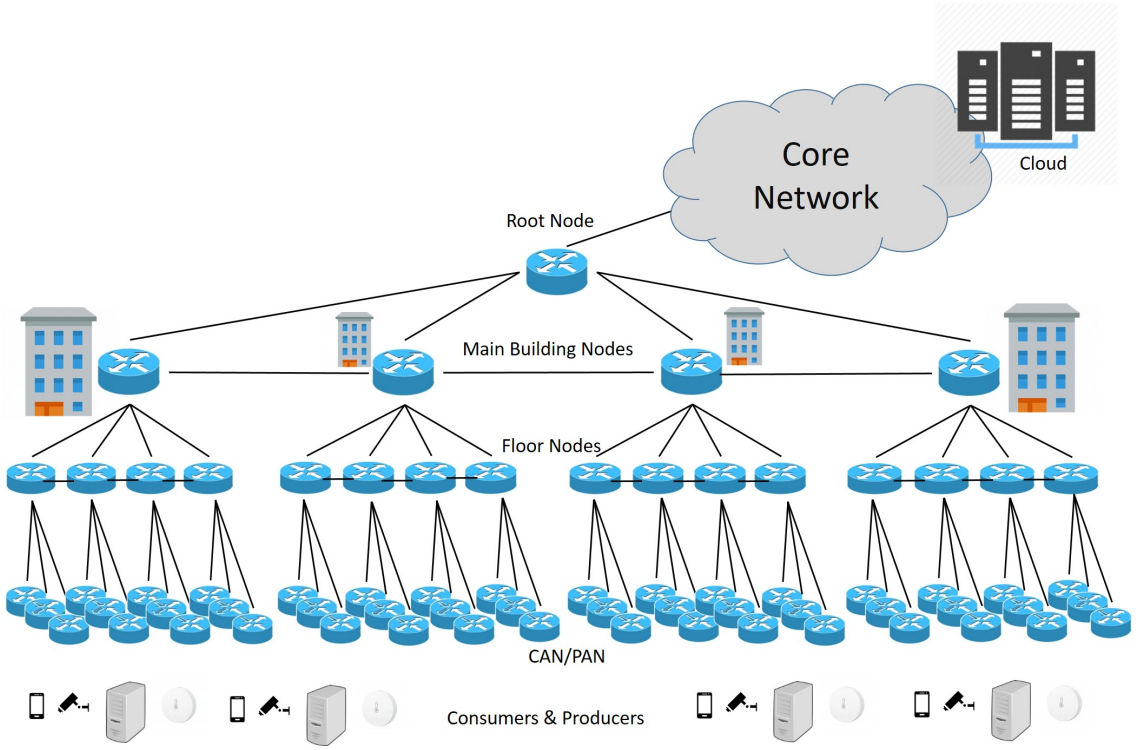


Figure 2. Reference campus network topology.

Generally, there are no restrictions in the way a function must be implemented. According to the literature, functions are generic software programs that can be in the form of unikernel, as in Reference [8], or containers, as in Reference [42], or even λ-expressions. In the following, we assume that a function is a self-consistent code, identified by a unique name and with a well-defined list of optional input parameters, which can be specified by the consumer; otherwise, a default value will be used.

For instance, a simple function computing the average over a set of float values is named *avgF* and takes as input a number of s float samples over which the average is computed. If not specified, the default value $s = 2$ is considered. Instead, a function compressing flac audio files into mp3 files is named *flacToMp3* and takes as input the kilobits per second value, which identifies the quality of the compression (128 kbps, 192 kbps, and 320 kbps). If not specified, the value 128 kbps is considered.

The IoT service is uniquely named in a hierarchical structure with a name composed of three distinct parts: the content name, the function name, and the input parameters. While the first two components are mandatory, the last one is optional. The tag EXEC is used for separating the content and the function names. For instance, the name

campus/building1/floor4/classroomA/temp/EXEC/avgF/{s=5}

identifies a service that computes the average value over 5 temperature measurements collected from sensors installed in classroom A, located on the fourth floor of building 1 of the campus.

Consumers, located in the campus as well request IoT services by sending interest packets that carry the service name. Of course, consumers can also request raw (non-processed) contents by

sending interests that carry only the traditional content name component, i.e., without the EXEC tag. To distinguish the two cases, in the following, we refer to as *service interest* the packet requesting a named service. Vice versa, we call interest a standard NDN packet requesting a content.

According to the standard ISO/IEC 11801 [48], a wired network is deployed in the campus that connects the IoT devices with a set of infrastructure elements acting as NDN content routers and fog nodes, which in the following is referred to as NDN-fog nodes. In addition to the traditional routing and forwarding operations, they are provided with storage and processing capabilities to dynamically offer IoT services. Of course, their resources are limited with respect to the cloud. As shown in Figure 2, the network resembles a fat tree topology, with 4 layers and 69 nodes. The root node connects the edge domain to the core network, which in turn is connected to a remote cloud facility.

Services can be executed inside the campus network, if there are available resources, or remotely in the cloud.

5.2. NDN-Fog Node Design

As shown in Figure 3, NDN-fog nodes implement the legacy NDN data plane, while the control plane is extended with a Service Decision Engine (SDE) that coordinates the following functionalities:

- *Function storing module* decides which functions should be installed in the node and which functions should be evicted. The selection is performed in a proactive way by following a popularity-based approach, similar to Reference [8]. During the network bootstrap, a first manual configuration is performed by the network administrator, who may also decide that specific functions must be installed only in specific locations, e.g., for privacy purposes. When a function is available in the node, a FIB entry is configured that binds the function name with the *AppFace* of the node.
- *Service allocation module* decides, after the service interest reception, if the node can execute the service and manages the signalling related to this operation.

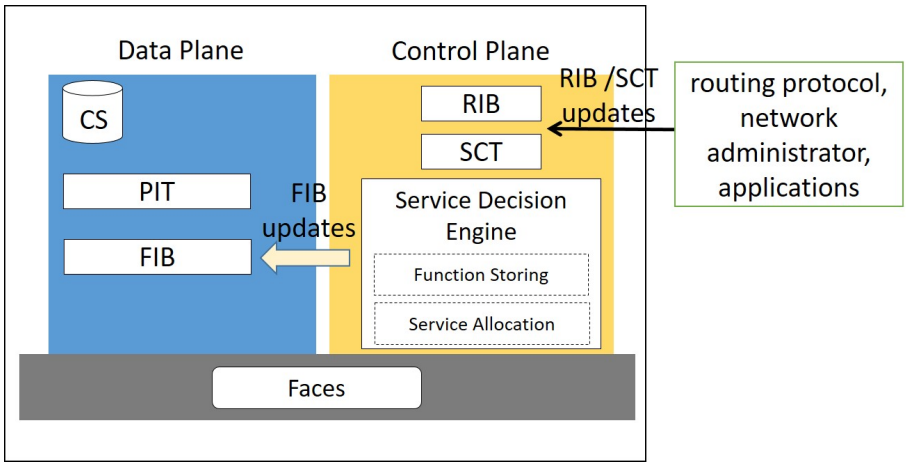


Figure 3. Architecture of NDN-fog nodes.

NDN-fog nodes can dynamically download the function code from the so-called *Function Data Base*, a storage unit located in the campus domain that contains the function catalog. A default route is installed in the FIBs of the NDN-fog nodes towards the Function Data Base; thus, functions can be retrieved by name with the standard NDN interest/data exchange. Of course, if the function code is available in the CS of an on-path node, the latter can directly satisfy the request without further forwarding it.

5.3. Service Provisioning

Service provisioning is performed on demand when service interest packets are sent by consumer applications. The latter are hosted in end-devices (e.g., users' smartphones or laptops, monitoring utilities, etc.) connected to the leaf nodes of the campus network. When receiving the request, each NDN-fog node applies a processing routine to discover if the result is already cached locally or if itself can be a candidate service executor. The service interest processing includes a bidding procedure that aims at executing the service at the edge in the best available NDN-fog node, which is the one offering the *lowest service provisioning time*. The latter is computed as the sum of the estimated processing time and the IoT data collection time.

The processing time is estimated according to the currently available node's resources, as provided by a standard CPU monitoring utility available in the node, once the CPU requirements of the service are known. The IoT data collection is estimated by reading the routing cost from the FIB entry per the requested content name.

The estimated service processing time is included in a new header field of the service interest that we call SPT. The consumer sets the SPT field in the interest to an upper bound value, which represents the time of executing the service in the remote cloud. The parameter is overwritten by an NDN-fog node offering a shorter delay.

If no cached result is located at the edge and no NDN-fog node is available to satisfy the request, e.g., because there are not enough computing resources, then the interest is forwarded to the cloud. As a result, two different forwarding steps are foreseen.

First, the service interest is forwarded in the campus network towards the IoT producer and the discovery is performed by on-path NDN-fog nodes. If the on-path search fails, then the second step starts and the service interest is forwarded towards the cloud. During this stage, newly traversed NDN-fog nodes still attempt to resolve the request locally. If no cached result or executor are located, finally the root node will forward the request to the cloud.

5.3.1. On-Path Forwarding

In this first stage, the service interest is forwarded from the leaf node connected to the consumer, which in the following is referred to as Consumer Access Node (CAN), and to the leaf node connected to the IoT data producer, which in the following is referred to as Producer Access Node (PAN). In case the consumer and producer have the same access node (CAN = PAN), then the candidate on-path executor is just this single node.

Forwarding is guided by the first component of the hierarchical service name, that is, the content name. Specifically, when the CAN or a subsequent NDN-fog node (let us call it N_i) receives the service interest, the following processing is performed.

- First, N_i looks in the CS for a matching name. If it is found, it means that the same service has been previously computed and the result can be immediately sent back to the consumer.
- If the CS matching fails, it looks in the PIT. If a matching is found, it means that the same service has been requested and it is pending. Therefore, N_i updates the correspondent PIT entry with the incoming service interest face and discards the packet.
- If the PIT matching fails, then the interest is processed by the SDE, which computes the service provisioning time and decides if N_i can candidate itself for the execution. This choice depends on the available computing resources and the implemented service allocation strategy, as it will be clarified in the following section. If the locally computed service provisioning time is lower or equal than the value already included in the SPT field, then N_i becomes the potential service executor: the SDE updates the SPT field and creates a new PIT entry that includes as additional record the service provisioning time. Then, the packet is forwarded to the next node towards the IoT provider (and, therefore, towards the PAN).

After applying the service interest processing, the PAN acts as follows:

1. If no on-path node is available for the execution, it sends back a NACK packet advertising that the service has not been allocated.
2. Otherwise, the node with the lowest service provisioning time is selected as the executor. In particular, if the executor is the PAN itself, then it will take in charge the computation. Otherwise, it sends back a data acknowledgement (ACK) packet carrying the service name and the correspondent SPT. The first node with the correspondent SPT in the PIT will act as the executor.

5.3.2. Forwarding towards the Cloud

This forwarding stage is performed only if the service has not been allocated in the path between CAN and PAN, i.e., the service provisioning time offered by the cloud is shorter than the one offered by on-path NDN-fog nodes.

Specifically, the NACK packet is forwarded hop-by-hop back to the CAN, which starts the forwarding process towards the cloud. It modifies the service interest by adding a well-known prefix, e.g., */cloud*, to the existing service name and resends the request. Thanks to the newly added name prefix, the request is now forwarded towards the cloud and, therefore, it will reach the root node. Each NDN-fog node receiving the service interest applies the previously discussed processing and evaluates the service provisioning time.

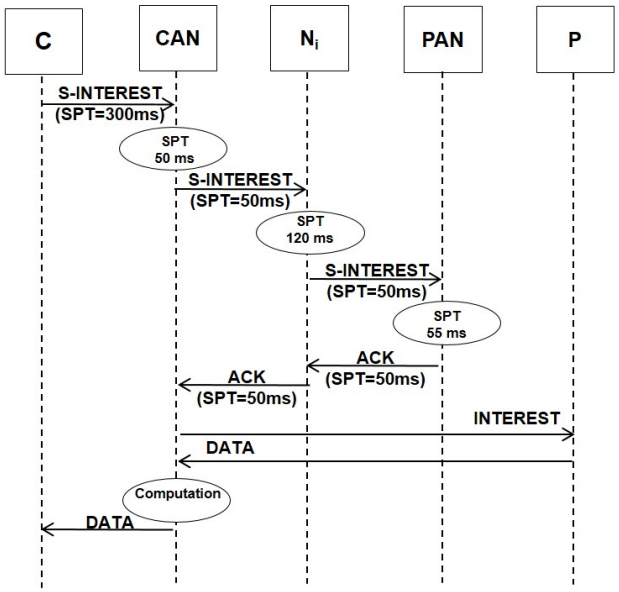
The procedure continues until the interest reaches the root node, which acts in a way slightly similar to the PAN: (i) if no node is available for the execution, the service interest is finally forwarded towards the cloud; (ii) otherwise, the node with the lowest service provisioning time is selected as the executor and a DATA ACK is sent back to notify the decision.

5.4. A Toy Example

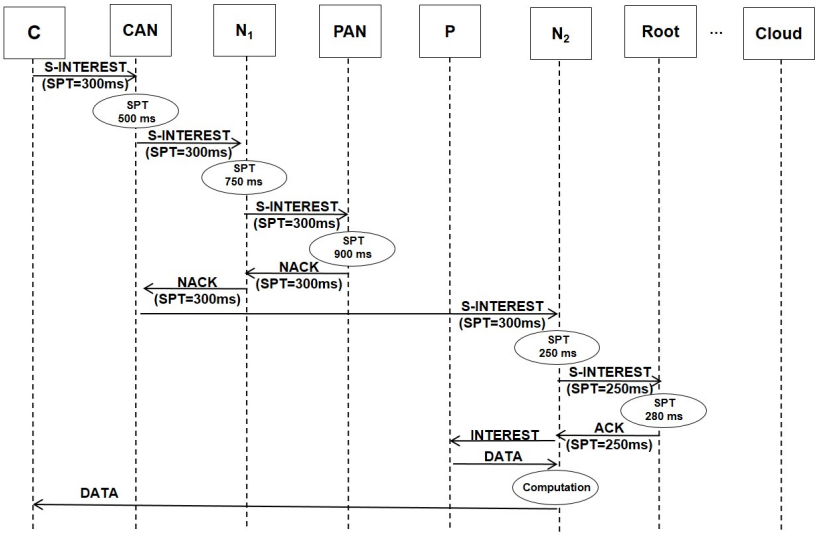
In order to better figure out how the proposed forwarding strategy works, let us refer to the toy example reported in Figure 4.

Service executed on-path: A consumer C transmits a service interest (labeled S-INTEREST in Figure 4a) to request a named service which is routed towards the data provider, P. The issued interest conveys the estimated service provisioning time in case the service is executed in the remote cloud. Each crossed node along the path estimates the service provisioning time and replaces the current value of the SPT field in the forwarded interest only if the time is lower than the one carried in the received interest. The interest is then forwarded until it reaches the PAN. In the example, the PAN is not able to execute the service within the shortest time but a node in the path (the CAN in the figure) is able to execute it. Thus, it transmits an ACK back. Once the ACK packet reaches the node that advertised the lowest SPT, it becomes the executor. Therefore, it starts the data retrieval from the IoT data producer through legacy NDN interest/data packets. Once data are collected, the executor performs the requested computation and sends the result to the consumer using the same name as the original service interest.

Service executed off-path: In such a case (Figure 4b), the NDN-fog nodes of the domain cannot ensure a service provisioning time shorter than the cloud; hence, a NACK is sent back by the on-path nodes. The CAN, then, issues a service interest towards the cloud. All nodes forward the packet by overwriting the SPT field whenever possible. The root node detects the availability of an executor (N_2) and sends an ACK back. N_2 is then in charge of retrieving data from the provider P and of sending the output of the computation to C.



(a) Service executed in the path towards the IoT data producer.



(b) Service executed in the path towards the remote cloud.

Figure 4. Main steps of the interest/data exchange in NDN-fog.

6. Performance Evaluation

6.1. Simulation Settings

The behaviour of the conceived NDN-fog nodes in the campus network topology depicted in Figure 2 is evaluated in a realistic way by means of ndnSIMv2.5 [11], the official discrete-event ns-3-based simulator deployed by the NDN community.

We have modified ndnSIM from its stock installation to implement the features of NDN-fog nodes and the abovementioned allocation strategies.

We consider a catalog of 1000 services, each one requiring a randomly selected amount of CPU resources in the range [25–375] Mcycles. Similarly to Reference [49], since there are no datasets containing traces of the service workloads in real IoT scenarios, we assume that services are requested according to a Zipf distribution with skewness parameter α set to 0.4 and 0.8. The request arrival rate follows the Poisson distribution.

The following benchmarking schemes are considered:

- *Closest executor strategy* (labeled as *CE* in the figures): Inspired by the work in Reference [10], it aims at executing the service as close as possible to the IoT providers.
- *Least loaded executor strategy* (labeled as *LLE* in the figures): It targets at selecting as the service executor the node with the lowest CPU usage, similarly to the work in Reference [34].
- *Remote cloud* (labeled as *Cloud* in the figures): It resembles the case when all services are executed in remote cloud facilities.

The following metrics are computed for the purpose of performance comparison:

- *Service provisioning time* is 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 at the executor).
- *Offloading-to-the-cloud percentage* is derived as the percentage of times the service is executed in the remote cloud. It only applies to our proposal, labeled as *NDN-fog*.
- *Intra-edge exchanged traffic* is computed as the overall number of NDN packets exchanged into the domain to request a given service, to select the executor, to retrieve the input data to be processed, and to return the result to the consumer.

Simulation settings are reported in Table 2, and they hold unless differently stated in the text. Results are averaged over 100 runs and reported with 95% confidence intervals.

Table 2. Main simulation settings.

Parameter	Value
Number of services	1000
Zipf's parameter	0.4, 0.8
Request arrival rate (λ)	Varying (10–80 requests/s)
CPU requests of services	Uniformly distributed in [25, 750] Mcycles
Edge link latency	[5, 10] ms
RTT to the cloud	50 ms, 100 ms, 150 ms
CPU node capabilities	[250–1500] MHz
Content size (σ)	[500, 1000, 2000] 1024 large packets

6.2. Simulation Results

The first set of results, shown in Figure 5, report the service provisioning time for the compared schemes under different workload settings in terms of size of the content to be processed (σ) and arrival rate of service requests (λ), when fixing the Zipf parameter α equal to 0.8.

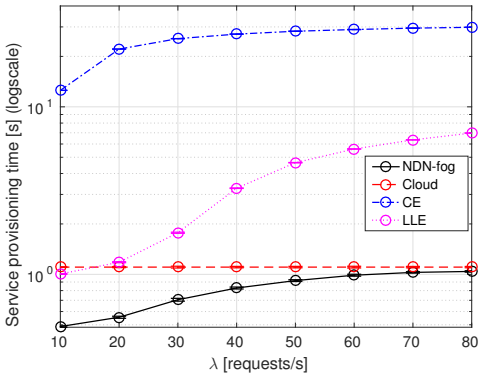
For the cloud solution, the metric is not affected by the parameter λ: the cloud can sustain high processing loads compared to the edge nodes. The processing time contribution is negligible compared to the latency incurred in retrieving the input content. In fact, the larger the content size, the higher the service provisioning time.

The LLE strategy, on the contrary, is highly affected by parameter λ. The considered metric highly increases with λ. The LLE performance is superior to the one experienced by the cloud solution for small arrival rates. This is especially true for large content size settings. As λ increases, instead, the service provisioning time gets higher than the one provided by the cloud and approaches 10 s.

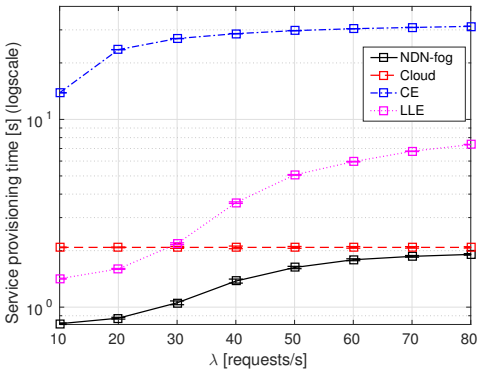
The CE strategy behaves quite poorly because nodes get rapidly loaded for larger values of λ. The metric does not change significantly as the content size increases. This is because the waiting time before the processing, which is inevitably experienced at the nodes close to the producers, is much more significant than the time required to retrieve the input content.

Our proposal outperforms the compared schemes under all the considered settings by scaling better with the size of the content and the arrival rate of service requests. However, due to the limited available processing resources at the edge nodes, its performance approaches the one experienced in the case of the remote cloud for high λ settings.

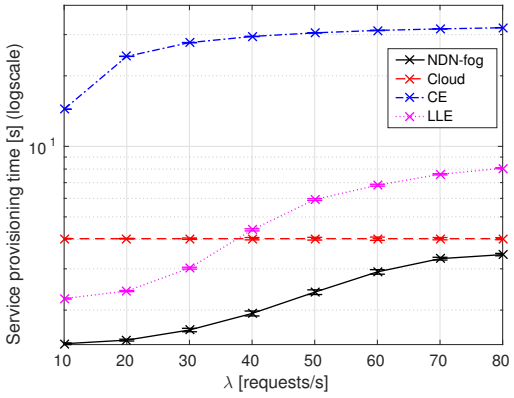
The better performance of the proposal has to be ascribed to the possibility to select as candidate executors also nodes that are off-path, i.e., in the path towards the cloud (and not only in the one towards the data producers) and the remote cloud itself, as the ultimate solution. This is confirmed by results in Figure 6. Notably, only a small percentage of services are offloaded to the cloud and only when the parameter λ is large. The larger the content size, the lower the offloading-to-the-cloud percentage. This is because the latency contribution due to content retrieval by IoT producers gets more significant for the cloud.



(a) Content size = 500 packets



(b) Content size = 1000 packets



(c) Content size = 2000 packets

Figure 5. Service provisioning time when varying the service request arrival rate (λ) for different content size settings (σ), α = 0.8.

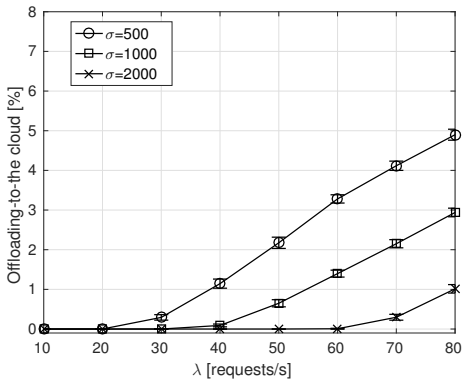


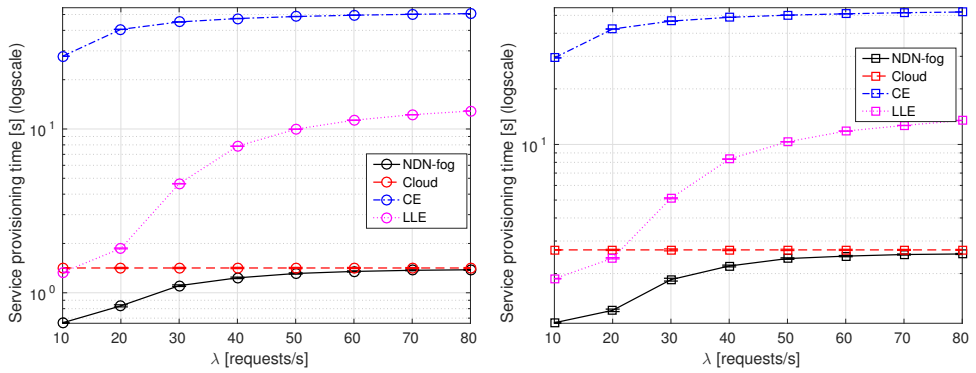
Figure 6. Offloading-to-the-cloud percentage for the NDN-fog solution when varying the service request arrival rate (λ) for different content size settings (σ), α = 0.8.

The supremacy of NDN-fog compared to the benchmark schemes is paid in terms of a larger amount of traffic generated in the domain, as shown in Table 3. For small content size settings, the intra-edge exchanged traffic for NDN-fog approaches values experienced by the cloud solution. As the content size increases, the intra-edge exchanged traffic in NDN-fog gets smaller compared to the cloud, although it is still higher than the CE solution, performing the execution as close as possible to the data producers.

It is interesting to observe that improvements of the proposed solution compared to the considered benchmark schemes are still significant under less favourable settings in terms of service popularity, i.e., $\alpha = 0.4$. The service provisioning time increases since more distinct services need to be executed, wasting processing resources of nodes. For NDN-fog, the metric achieves values are well below the ones provided by the benchmark schemes (see Figure 7).

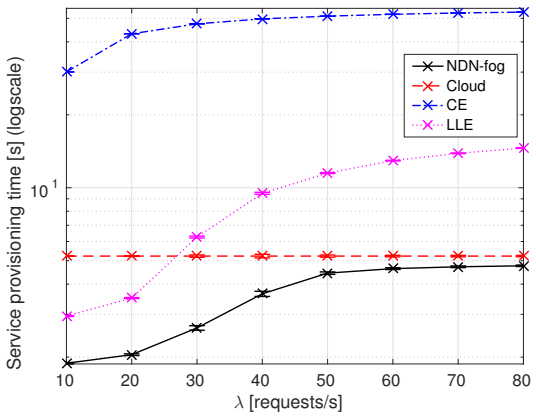
Table 3. Intra-edge exchanged traffic for different content size settings (σ), λ = 80 requests/s.

Content size	NDN-fog		Cloud		CE		LLE	
	$\alpha = 0.4$	$\alpha = 0.8$	$\alpha = 0.4$	$\alpha = 0.8$	$\alpha = 0.4$	$\alpha = 0.8$	$\alpha = 0.4$	$\alpha = 0.8$
500 packets	2012	1567.4	1945	1497.5	496.4	384.54	2018	1517.4
1000 packets	3572	2787.1	3884	2990.7	981.3	757.85	4024	3019.6
2000 packets	6759	5132.6	7764	5977.2	1951	1504.5	8053	6043



(a) Content size = 500 packets

(b) Content size = 1000 packets



(c) Content size = 2000 packets

Figure 7. Service provisioning time when varying the service request arrival rate (λ) for different content size settings (σ), α = 0.4.

The larger processing load at NDN-fog nodes results in a larger percentage of services offloaded to the cloud compared to the previous case with $\alpha = 0.8$; see Figure 8. The impact of different

round-trip-time (RTT) values towards the cloud on the metric is also evaluated. The larger the RTT, the lower the metric: it is more convenient to execute the service at the edge to reduce the service provisioning time. This is especially true as the input content size increases.

For what concerns the intra-edge traffic, the same trends already discussed for $\alpha = 0.8$ can be observed in Table 3. However, a higher traffic is exchanged due to the presence of less popular services.

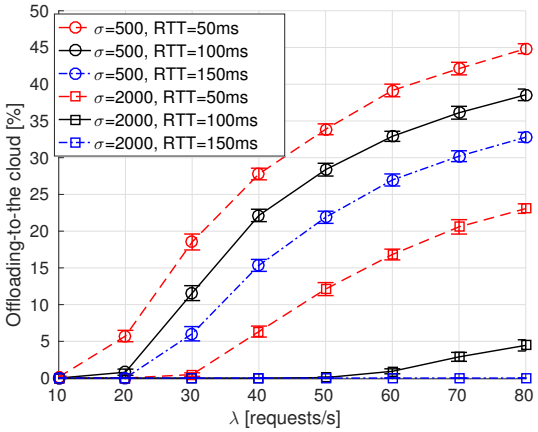


Figure 8. Offloading-to-the cloud percentage for the NDN-fog solution when varying the service request arrival rate (λ) for different content size (σ) and round-trip-time (RTT) settings, $\alpha = 0.4$.

7. Conclusions and Future Work

In this paper, we have discussed the potential of NDN to support fog computing in smart environments. After scanning the relevant literature, our proposal has been presented that is tailored without loss of generality to the representative IoT smart campus scenario. We have proposed a service orchestration mechanism which is fully aligned with the fog computing concept. Indeed, it supports the decision of whether the IoT data processing has to be performed at the edge or remotely in the cloud in a fully distributed manner and targeting the minimization of the service provisioning time.

Achieved simulation results confirm the supremacy of our proposal against literature solutions under different workload settings (i.e., input content size and arrival rate of service requests). Improvements are significant in terms of reduction of the service provisioning time. As a drawback, the proposal incurs a slightly higher amount of traffic exchanged in the edge domain. Such a condition holds when comparing NDN-fog against the conventional cloud solution only for small content sizes. The larger traffic exchanged compared to the CE strategy is widely compensated by the achieved reduction of the service provisioning time, which is well below 10 s for NDN-fog under all the considered settings.

All in all, the proposal adequately targets a user-centric policy which values more how the consumers perceive the services in terms of service provisioning time, in our case, than the incurred traffic in the edge domain.

Future works will be devoted to the design of service executor strategies with different objectives, e.g., targeting computing load balancing and reducting the amount of exchanged traffic while also considering services with different priorities. Moreover, we plan to investigate the integration of the NDN fog computing paradigm with other emerging techniques, such as software-defined networking (SDN) and network function virtualization (NFV).

Author Contributions: In this paper, M.A., G.R., C.C., and A.M. conceived the idea. G.R., M.A., and C.C. conceived and designed the experiments; M.A. and G.R. performed the experiments. All the authors (M.A., G.R., C.C., A.M., V.L., C.T.C.) organized the work; analyzed the experiments; and reviewed the writing of the paper, its structure, and its intellectual content.

Funding: This research was partially funded by the Italian Government under grant PON ARS01_00836 for the COGITO (A COGnItive dynamic sysTem to allOw buildings to learn and adapt) PON Project.

Conflicts of Interest: The authors declare no conflict of interest. The funders had no role in the design of the study; in the collection, analyses, or interpretation of data; in the writing of the manuscript; or in the decision to publish the results.

Abbreviations

The following abbreviations are used in this manuscript:

AppFace	Application Face
CS	Content Store
FIB	Forwarding Information Base
ICN	Information Centric Networking
IEC	International Electrotechnical Commission
IoT	Internet of Things
ISO	International Organization for Standardization
NetDeviceFace	Network Device Face
NCN	Named Computation Networking
NDN	Named Data Networking
NFN	Named Function Networking
PIT	Pending Interest Table
RIB	Routing Information Base
RTT	Round Trip Time
SCT	Strategy Choice Table
SDE	Service Decision Engine

References

1. Lee, I.; Lee, K. The Internet of Things (IoT): Applications, investments, and challenges for enterprises. *Bus. Horizons* **2015**, *58*, 431–440. [CrossRef]
2. Cicirelli, F.; Guerrieri, A.; Spezzano, G.; Vinci, A.; Briante, O.; Iera, A.; Ruggeri, G. Edge computing and social internet of things for large-scale smart environments development. *IEEE Internet Things J.* **2017**, *5*, 2557–2571. [CrossRef]
3. Chiang, M.; Zhang, T. Fog and IoT: An overview of research opportunities. *IEEE Internet Things J.* **2016**, *3*, 854–864. [CrossRef]
4. OFC. Openfog Consortium. Available online: http://www.openfogconsortium.org/ (accessed on 30 September 2019).
5. Zhang, L.; Afanasyev, A.; Burke, J.; Jacobson, V.; Crowley, P.; Papadopoulos, C.; Wang, L.; Zhang, B. Named Data Networking. *ACM SIGCOMM Comput. Commun. Rev.* **2014**, *44*, 66–73. [CrossRef]
6. Tschudin, C.; Sifalakis, M. Named functions and cached computations. In Proceedings of the 2014 IEEE 11th Consumer Communications and Networking Conference (CCNC), Las Vegas, NV, USA, 10–13 Janary 2014 .
7. Amadeo, M.; Ruggeri, G.; Campolo, C.; Molinaro, A. IoT Services Allocation at the Edge via Named Data Networking: From Optimal Bounds to Practical Design. *IEEE Trans. Netw. Serv. Manag.* **2019**, *16*, 661–674. [CrossRef]
8. Król, M.; Psaras, I. NFaaS: Named function as a service. In Proceedings of the 4th ACM Conference on Information-Centric Networking, Berlin, Germany, 26–28 September 2017.
9. Ascigil, O.; Reñé, S.; Xylomenos, G.; Psaras, I.; Pavlou, G. A keyword-based ICN-IoT platform. In Proceedings of the 4th ACM Conference on Information-Centric Networking, Berlin, Germany, 26–28 September 2017.
10. Scherb, C.; Grewe, D.; Wagner, M.; Tschudin, C. Resolution strategies for networking the IoT at the edge via named functions. In Proceedings of the 2018 15th IEEE Annual Consumer Communications & Networking Conference (CCNC), Las Vegas, NV, USA, 12–15 January 2018.
11. Mastorakis, S.; Afanasyev, A.; Moiseenko, I.; Zhang, L. ndnSIM 2.0: A New Version of the NDN Simulator for NS-3. Available online: https://www.researchgate.net/profile/Spyridon_Mastorakis/publication/281652451_ndnSIM_20_A_new_version_of_the_NDN_simulator_for_NS-3/links/5b196020a6fdcca67b63660d/ndnSIM-20-A-new-version-of-the-NDN-simulator-for-NS-3.pdf (accessed on 22 October 2019).

12. Ahlgren, B.; Dannewitz, C.; Imbrenda, C.; Kutscher, D.; Ohlman, B. A Survey of Information-centric Networking. *IEEE Commun. Mag.* **2012**, *50*, 26–36. [CrossRef]
13. Afanasyev, A.; Shi, J.; Zhang, B.; Zhang, L.; Moiseenko, I.; Yu, Y.; Shang, W.; Li, Y.; Mastorakis, S.; Huang, Y.; et al. NFD Developer's Guide. Available online: https://named-data.net/wp-content/uploads/2016/03/ndn-0021-diff-5..6-nfd-developer-guide.pdf (accessed on 22 October 2019).
14. Piro, G.; Amadeo, M.; Boggia, G.; Campolo, C.; Grieco, L.A.; Molinaro, A.; Ruggeri, G. Gazing into the crystal ball: When the future internet meets the mobile clouds. *IEEE Trans. Cloud Comput.* **2016**, *7*, 210–223. [CrossRef]
15. Zhang, G.; Li, Y.; Lin, T. Caching in Information Centric Networking: A Survey. *Comput. Netw.* **2013**, *57*, 3128–3141. [CrossRef]
16. Yi, C.; Afanasyev, A.; Moiseenko, I.; Wang, L.; Zhang, B.; Zhang, L. A case for stateful forwarding plane. *Comput. Commun.* **2013**, *36*, 779–791. [CrossRef]
17. Shang, W.; Bannis, A.; Liang, T.; Wang, Z.; Yu, Y.; Afanasyev, A.; Thompson, J.; Burke, J.; Zhang, B.; Zhang, L. Named Data Networking of Things. In Proceedings of the IEEE First International Conference on Internet-of-Things Design and Implementation (IoTDI), Berlin, Germany, 4–8 April 2016.
18. Baccelli, E.; Mehlis, C.; Hahm, O.; Schmidt, T.C.; Wählisch, M. Information centric networking in the IoT: Experiments with NDN in the wild. In Proceedings of the 1st ACM Conference on Information-Centric Networking, Paris, France, 24–26 September 2014.
19. Amadeo, M.; Campolo, C.; Iera, A.; Molinaro, A. Information Centric Networking in IoT scenarios: The case of a smart home. In Proceedings of the IEEE International Conference on Communications (ICC), London, UK, 8–12 June 2015.
20. Burke, J.; Gasti, P.; Nathan, N.; Tsudik, G. Securing instrumented environments over content-centric networking: The case of lighting control and NDN. In Proceedings of the 2013 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS), Turin, Italy, 14–19 April 2013 .
21. Amadeo, M.; Briante, O.; Campolo, C.; Molinaro, A.; Ruggeri, G. Information-centric networking for M2M communications: Design and deployment. *Comput. Commun.* **2016**, *89*, 105–116. [CrossRef]
22. Tourani, R.; Misra, S.; Mick, T.; Panwar, G. Security, privacy, and access control in information-centric networking: A survey. *IEEE Commun. Surv. Tutor.* **2018**, *20*, 566–600. [CrossRef]
23. Shang, W.; Yu, Y.; Liang, T.; Zhang, B.; Zhang, L. Ndn-ace: Access Control for Constrained Environments over Named Data Networking. Available online: http://new.named-data.net/wp-content/uploads/2015/12/ndn-0036-1-ndn-ace.pdf (accessed on 22 October 2019).
24. Zhang, Z.; Yu, Y.; Zhang, H.; Newberry, E.; Mastorakis, S.; Li, Y.; Afanasyev, A.; Zhang, L. An overview of security support in Named Data Networking. *IEEE Commun. Mag.* **2018**, *56*, 62–68. [CrossRef]
25. Bonomi, F.; Milito, R.; Zhu, J.; Addepalli, S. Fog computing and its role in the internet of things. In Proceedings of the First Edition of the MCC Workshop on Mobile Cloud Computing, Helsinki, Finland, 17 August 2012.
26. Fog Computing and the Internet of Things: Extend the Cloud to Where the Things Are. Cisco White Paper. Available online: https://www.cisco.com/c/dam/en_us/solutions/trends/iot/docs/computing-overview.pdf (accessed on 22 October 2019).
27. Aazam, M.; Zeadally, S.; Harras, K.A. Deploying fog computing in industrial internet of things and industry 4.0. *IEEE Trans. Ind. Inform.* **2018**, *14*, 4674–4682. [CrossRef]
28. Hou, X.; Li, Y.; Chen, M.; Wu, D.; Jin, D.; Chen, S. Vehicular fog computing: A viewpoint of vehicles as the infrastructures. *IEEE Trans. Veh. Tech.* **2016**, *65*, 3860–3873. [CrossRef]
29. Menon, V.G.; Prathap, J. Vehicular fog computing: challenges applications and future directions. In *Fog Computing: Breakthroughs in Research and Practice*; IGI Global: Hershey, PA, USA, 2018; pp. 220–229.
30. Yousefpour, A.; Fung, C.; Nguyen, T.; Kadiyala, K.; Jalali, F.; Niakanlahiji, A.; Kong, J.; Jue, J.P. All one needs to know about fog computing and related edge computing paradigms: A complete survey. *J. Syst. Archit.* **2019**, *98*, 289–330. [CrossRef]
31. Baktir, A.C.; Ozgovde, A.; Ersoy, C. How can edge computing benefit from software-defined networking: A survey, use cases, and future directions. *IEEE Commun. Surv. Tutor.* **2017**, *19*, 2359–2391. [CrossRef]
32. Duan, Q.; Yan, Y.; Vasilakos, A.V. A survey on service-oriented network virtualization toward convergence of networking and cloud computing. *IEEE Trans. Netw. Serv. Manag.* **2012**, *9*, 373–392. [CrossRef]

33. Gedeon, J.; Meurisch, C.; Bhat, D.; Stein, M.; Wang, L.; Mühlhäuser, M. Router-based brokering for surrogate discovery in edge computing. In Proceedings of the 2017 IEEE 37th International Conference on Distributed Computing Systems Workshops (ICDCSW), Atlanta, GA, USA, 5–8 June 2017.
34. Mtibaa, A.; Tourani, R.; Misra, S.; Burke, J.; Zhang, L. Towards Edge Computing over Named Data Networking. In Proceedings of the 2018 IEEE International Conference on Edge Computing (EDGE), San Francisco, CA, USA, 2–7 July 2018 .
35. Amadeo, M.; Campolo, C.; Molinaro, A. NDNe: Enhancing Named Data Networking to Support Cloudification at the Edge. *IEEE Commun. Lett.* **2016**, *20*, 2264–2267. [CrossRef]
36. Amadeo, M.; Campolo, C.; Molinaro, A.; Ruggeri, G. IoT data processing at the edge with Named Data Networking. In Proceedings of the 24th European Wireless Conference, Catania, Italy, 2–4 May 2018.
37. Król, M.; Habak, K.; Oran, D.; Kutscher, D.; Psaras, I. Rice: Remote method invocation in icn. In Proceedings of the 5th ACM Conference on Information-Centric Networking, Boston, MA, USA, 21–23 September 2018.
38. Krol, M.; Marxer, C.; Grewe, D.; Psaras, I.; Tschudin, C. Open security issues for edge named function environments. *IEEE Commun. Mag.* **2018**, *56*, 69–75. [CrossRef]
39. Marxer, C.; Scherb, C.; Tschudin, C. Access-controlled in-network processing of named data. In Proceedings of the 3rd ACM Conference on Information-Centric Networking, Kyoto, Japan, 26–28 September 2016.
40. Scherb, C.; Marxer, C.; Schnurrenberger, U.; Tschudin, C. In-network live stream processing with named functions. In Proceedings of the IFIP Networking Conference and Workshops, Stockholm, Sweden, 12–16 June 2017.
41. Liu, C.; Loo, B.T.; Mao, Y. Declarative automated cloud resource orchestration. In Proceedings of the 2nd ACM Symposium on Cloud Computing, Cascais, Portugal, 26–28 October 2011; p. 26.
42. Wang, Q.; Lee, B.; Murray, N.; Qiao, Y. CS-Man: Computation service management for IoT in-network processing. In Proceedings of the 2016 27th Irish Signals and Systems Conference (ISSC), London, UK, 21–22 June 2016.
43. Wang, Q.; Lee, B.; Murray, N.; Qiao, Y. IProIoT: An in-network processing framework for IoT using Information Centric Networking. In Proceedings of the 2017 Ninth International Conference on Ubiquitous and Future Networks (ICUFN), Milan, Italy, 4–7 July 2017.
44. Amadeo, M.; Molinaro, A.; Paratore, S.Y.; Altomare, A.; Giordano, A.; Mastroianni, C. A Cloud of Things framework for smart home services based on Information Centric Networking. In Proceedings of the IEEE 14th International Conference on Networking, Sensing and Control (ICNSC), Calabria, Italy, 16–18 May 2017.
45. Amadeo, M.; Giordano, A.; Mastroianni, C.; Molinaro, A. On the integration of information centric networking and fog computing for smart home services. In *The Internet of Things for Smart Urban Ecosystems*; Springer: Cham, Switzerland, 2019; pp. 75–93.
46. Scherb, C.; Tschudin, C. Smart Execution Strategy Selection for Multi Tier Execution in Named Function Networking. In Proceedings of the 2018 IEEE International Conference on Communications Workshops (ICC Workshops), Kansas City, MO, USA, 20–24 May 2018.
47. Hoque, A.; Amin, S.O.; Alyyan, A.; Zhang, B.; Zhang, L.; Wang, L. NLSR: Named-data link state routing protocol. In Proceedings of the ACM SIGCOMM Workshop on Information-Centric Networking, Hong Kong, China, 12 August 2013; pp. 15–20.
48. ISO/IEC. 11801-2:2017 Information Technology—Generic Cabling for Customer Premises. 2017. Available online: https://www.iso.org/standard/66183.html (accessed on 22 October 2019).
49. Elbamby, M.S.; Bennis, M.; Saad, W. Proactive edge computing in latency-constrained fog networks. In Proceedings of the 2017 European conference on networks and communications (EuCNC), Oulu, Finland, 12–15 June 2017.