Lightweight Named Object: an ICN-based Abstraction for IoT Device Programming and Management

Lorenzo Bracciale, Pierpaolo Loreti, Andrea Detti, Riccardo Paolillo and Nicola Blefari Melazzi

***Abstract*—The expected dramatic growth of connected things raises the issue of how to efficiently organize them, in order to monitor and manage functions and interactions. Information Centric Networking (ICN) is a communication paradigm that provides content-oriented functionality in the network and at the network level, including content routing, caching, multicast, mobility, data-centric security and a flexible namespace. Thus, it is a viable solution for supporting IoT services without requiring any centralized entity. In this work we introduce the *Lightweight Named Object* solution: a convenient way to represent physical IoT objects in a derived name space, exploiting ICN. We show that this abstraction can: i) increase the programming simplicity; ii) offer extended functionality, such as augmentation and upgrading, to cope with the "software erosion", and iii) implement a common interaction logic involving mutual function invocation. We present some proof-of-concept implementations of the proposed abstraction dealing with challenging IoT test cases; we also carry out a performance evaluation in a simulated network scenario.**

***Index Terms*—IoT, ICN, NDN, Distributed Objects, Naming, Augmentation, Programming Language, Data-Centric Security**

I. Introduction

The Internet of Things (IoT) concept is transforming many businesses, driving the digital transformation of industries and creating an immeasurable social value. As digital things become increasingly cheaper, more and more devices are connected to the global network, moving the number of connected hosts up of one order of magnitude. Sensors and actuators are widespread in a myriad of different and heterogeneous environments, from homes (smart homes) to whole cities (smart cities), enabling pervasive smart services and deeply transforming the way we perceive, access and manage the environment around us [1].

A necessary condition for building such smart environments is that "things" communicate and interact with each other according to a specific application logic, which can be provided both by system designers and by final users. Nowadays this task is often entirely entrusted to cloud IoT platforms that collect and distribute data at a large scale, offering sophisticated tools for IoT system programming and data analytics [2], [3].

Cloud platforms typically work at the application level; however, a more efficient and higher-performance solution would be to address at the network level many challenging tasks that need to be accomplished (also) with physical/local support, such as device discovery, software/firmware update, local device composition, local programmability and build-in data security and access management [4].

While standardization bodies offer several protocol solutions [3] for device communication (e.g. MQTT, CoAP, AMQP) discovery (e.g. Physical Web, mDNS) identification (e.g. uCode) and infrastructure building (e.g. 6LowPan), maintainability and business issues arose, demanding new ways for troubleshooting local problems as well as upgrading an already deployed infrastructure to support either software or hardware changes.

Information Centric Networking (ICN) is a paradigm emerged to overcome some intrinsic limitations of the IP protocol. In ICN, the network provides users with access to content by names, instead of providing communication channels between hosts. The idea is to provide access to named data as the fundamental network service. This means that all content (e.g. a document, a picture) is given a name that does not include references to its location; then, users requests for a specific content are routed toward the closest copy of such a content, which could be stored in a server, in a cache contained in a network node or even in another users device; finally the content is delivered to the requesting user by the network. As a consequence, applications can refer to data through a name that can potentially contains semantic. In this way network and applications are allowed to share the same namespace, enabling simpler edge networking and boosting innovative IoT architectures, for instance through appropriate naming conventions. Moreover, ICN uses data centric security rather than secure communication channels (e.g. HTTPS), defining a name-based trust schema for distributed authentication and authorization [5].

ICN has been already proposed to cope with IoT challenges, contributing to simplify the IoT landscape, particularly providing solutions to devices organization [6], [7], [8], in-network data processing [6], location-based routing [9], improvement of system performance [10] and embedding of authorization

Lorenzo Bracciale and Pierpaolo Loreti are with the Department of Electrical Engineering, University of Rome Tor Vergata
E-mails: name.surname@uniroma2.it
Riccardo Paolillo is a researcher of the Consorzio Nazionale Interuniversitario per le Telecomunicazioni CNIT,
E-mail: riccardo.paolillo@gmail.com

Andrea Detti and Nicola Blefari Melazzi is with the Department of Electrical Engineering, University of Rome Tor Vergata and with CNIT
E-mail: name.surname@uniroma2.it

This work has been partly funded by the EU-JP H2020 ICN2020 and Fed4IoT projects.

and data security in the network, [11], [12]; ICN solutions have also been the object of standardization work [13].

In this work we present an ICN framework designed to enable a more flexible and easier management of IoT devices. To this aim, we introduce the *Lightweight Named Object (LNO)* solution, an ICN-based abstraction able to provide a convenient "things" interface for IoT management and programmability. LNO does not require any "bindings" (logic connections supported by persistent communication channels) among all the remote parts composing an object, making it suitable for dynamic IoT environment. A lightweight object is identified by a simple, shared name prefix on the global namespace natively provided by ICN technology.

We also show how the introduction of this abstraction brings about several advantages for programming patterns, service composition, device upgrading and object augmentation, which can be used for device-to-device interactions and/or exploited by existing IoT cloud platforms.

It is a matter of fact that several IoT programming solutions, including IBM Flow Programming and the Micro Service architecture, are pushing towards a new service/data organization to cope with the dynamic and complex nature of IoT scenarios. LNO uses the ICN namespace organization to express the semantic of the new network interface, reproducing the effectiveness of the Object Oriented Programming.

In what follows we describe the LNO concept and present several examples to see how it makes more easy to program IoT. The paper is organized as follows. Section II introduces the Lightweight Named Object, presenting the main motivations and advantages of the proposed architecture. In sections III and IV we present how the LNO and its security mechanisms can be implemented by using ICN. In section V we show some examples of service implementation and how these services can be programmed using LNO. In section VI we report a preliminary performance evaluation. Finally, we review the related work in the field and we draw the conclusions.

II. LNO Concept

The *Lightweight Named Object* (LNO) concept consists in using ICN hierarchical names and their inherent security, organizing the namespaces provided by the network with aim of addressing specific IoT tasks. In this section we give a definition as well as practical examples on how is possible by using LNO to achieve an easier and more clear programming of intelligent functionality in smart environments (building, houses, cities, etc.).

Definition: a *Lightweight Named Object* is a set of named data and functions published by the ICN network under the same network prefix, with the goal of exposing a management interface for real "things".

To clarify the concept we use the example of a video camera, model MG21 and Serial Number 12345, exploiting ICN communication and deployed in a smart environment. The manufacturer or the installer has to assign to the camera a unique ICN name prefix such as `/videocamera`, `/VC/MG21/12345`, `/SN12345`. ICN security rules allow to reserve a name prefix, allowing only authorized entities to publish a name, as explained in section IV. In the following, for the sake of simplicity, we use the simple name prefix `/camera1` and some named functions and data published under the prefix as reported in figure 1.

Assuming that the camera adopts the LNO concept, it exposes some interfaces under the prefix `/camera1`: these interfaces allow to access data (e.g. status, stream) or to execute functions (e.g. on, off, detect). All interfaces occupy a place in the namespace such as `/camera1/on`.

The LNO semantically associates all functions and data to the "camera1" object, by sharing the same name prefix, even if they are not directly implemented by the camera object and are exposed by other network nodes; in fact ICN "hides" how and where the interface is implemented: for instance the detection function can be provided by an external controller that uses the video stream coming from the camera. The LNO defines an *object-level namespace that offers a semantic link for all the functions that refer to the same objects*. The link is provided at the network layer by ICN, given that all the functions are published under the same name prefix and thus preventing the so called *namespace pollution*, a typical problem of many programming languages. As an example, the `detect` function may assume different meanings if applied to a camera or to an anti-intrusion alarm and the detection functionality can also be physically implemented on different nodes such as the camera controller and the alarm central device. This is a fundamental difference between our solution and the Named Function Networking (NFN) approach (see section VII for a more detailed analysis).

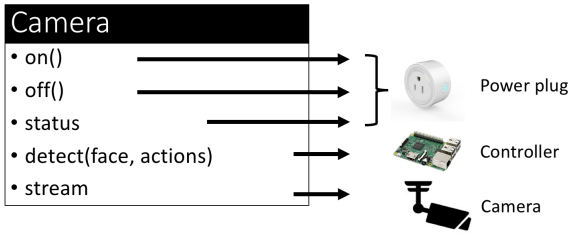

Fig. 1. Camera methods can be implemented by different networking elements

With reference to an IoT service scenario, the usage of LNO can lead to several advantages for managing things and programming their relations, which we are going to briefly describe hereafter.

An easier way to implement program logic: The ICN technology has been proved to be an effective solution to implement network services [14] and network functions [15] to support Future Internet programming paradigms. Indeed IoT applications need to express relations among objects in an easy way, in order to build complex and predictable behaviors. For instance, it could be needed to implement the automatic opening of a door when a machine learning software recognizes from a camera that an employee of a company stands in front of the door. In this case, LNO allows expressing such complex behavior by simply executing a named function and passing some data as parameters. In more details, we need to

invoke `camera1.detect`) with the following parameters: i) named data corresponding to the "face" of an employee (e.g. `/faces/alice`) and ii) named function corresponding to the action to perform after confirming the detection (e.g. call `/door1/open`). LNO provides a convenient abstraction to interact with "things", resembling *intentional programming* [16] but applied to real objects, where we can express our intent with expressions such as:

```
camera1.detect(faces.alice, door1.open)
```

This expression offers an high-level vision of the program behavior and a convenient way to express a relation among IoT services, objects and data, not originally built to work together. Differently from the use of sophisticated but centralized IoT cloud platforms, this approach operates at the network level, offering the extra advantage of easing the management of network functions (e.g., the setup of a firewall to regulate the access to IoT appliances).

Network object augmentation: Object augmentation is a common practice in prototype-based, object-oriented programming languages. Augmentation is an approach consisting in extending the object functionality without the need to change the original object implementation.

With LNO it is possible to augment objects by providing an extended set of functions in addition to the set of "internal" functions that they are already shipped with. The networking layer provides the ability to extend objects by simply publishing a name under the object network name prefix. This enable the re-purposing of existing hardware by adding new software functionality. Let us consider for instance the case of a surveillance camera that exports only the `stream` name to access the video stream. With object augmentation we can re-purpose this object for face recognition using suitable algorithms in order to provide an IAM service (we present the implementation of this use case in section V-B). Even though this is possible by simply applying the face recognition function on top of the video stream (for instance pipelining the stream with the algorithm using lambda calculus [15]), it could be more convenient from an IoT programmer/maintainer point of view to see it as a "native feature" of the camera. This enables the programming of a complex functionality as an easy-to-read operation that can be expressed as: `camera1.detect(faces.alice, door1.open)`.

The `detect` method enriches the basic set of functionality originally shipped with the camera by the hardware vendor, and can be implemented by an extra hardware or through virtual container-based machines and advertised on the network. As an example, in figure 1 there are several methods of camera objects implemented by different networking elements. For instance, the on/off methods can be implemented by smart plugs, the face detection can be implemented by a controller. All these methods can be seen as methods of the "camera1" object but not all of them must be initially shipped with the original hardware.

Easy-to-update object: Updating things is a challenging issue for IoT management platforms. It implies software updates to patch security flaws, correct software bug, improve performance or add new features. In many cases, performing periodic software update is vital to keep things efficient and usable, fighting also the so called *software erosion*[1]. Updating procedures usually require a support by the hardware manufacturer for firmware creation and dissemination, resulting in a potential risk for the maintainability of large scale device deployments. With LNO it is possible to update the object functionality by simply overriding its name on the network namespace and advertise it with a greater priority. The network will automatically route any further request for that function to the new producer, without requiring any software update released by the hardware vendor and preserving the same name hence the same service interface. The network maintainer can therefore manage things and also their upgrade by orchestrating the name advertising on the routing plane.

Security by design: Un-secure IoT is simply not an option. Thus, we dedicate a whole section (section IV) to explain how it is possible to make secure the Lightweight Named Object solution, by leveraging on the ICN built-in network security capabilities.

Service discovery through common network names: Programming a software (e.g., a mobile application) that interacts with the surrounding things, requires a way for the application to automatically discover and call specific methods of the objects. For instance to turn on the light in the current room, the application must know both the existence of the `light1` object and the availability of a function called `on` that should be called to turn on the light. This knowledge plays also a prominent role for implementing relations between objects.

However, defining a common interfaces for all the similar objects is not an easy task. Instead, with ICN, this agreement can be done on a *name basis*, facilitating the sharing of common interfaces, to allow objects to export their functionality so that other objects can use them. For instance, it is not needed for an object to know the protocol and the encoding required to speak with an automatic door but it suffices to know that "often" doors objects export a function named "open". In this case we can use a *loose naming convention rather then a complex protocol definition*. Similar objects can share similar names, as names implicitly express the interface to communicate with the objects.

Distributed deployment: In contrast to many IoT deployment scenarios, where we have simple objects and "smart" middlebox/gateway, with LNO we can easily distribute functions in the network. It has also be proven that the routing plane of ICN can be implemented in life-size IoT deployments [17] (this work also includes a comparison with common IoT standards such as 6LoWPAN/RPL/UDP).

III. LNO Implementation

A. Naming schema

It is important to define the ICN naming schema to support the LNO concept. We adopt the hierarchical naming structure of the NDN architecture [18] and we select a name composition that is an extension of [14] and [15]. Specifically, we

[1] slow deterioration of software performance over time or its diminishing responsiveness, which can eventually lead to software becoming faulty or unusable

propose an object name composed by four parts: the routable prefix, the name of the object, the specific data or function name and its (optional) parameters (see fig. 2).

The **routable prefix** is used by the network forwarder to restrict the propagation of Interest packets to a specific area in their way to the data location [12].

The **name of the object** part represents the physical (or virtual) device in the namespace.

The **data or function name** represents a way to access the data or to invoke the function execution; it is published under the object scope identified by the routable prefix and the name of the object.

Finally, the **parameters** part is optional; it can be needed by the functions to address a specific problem. Each function may require one or more parameters that are concatenated in the request by the "_" character that acts as a delimiter between the parameters values. Clearly the underscore becomes a reserved symbol for the LNO naming scheme, which, if necessary can be escaped e.g. with the ASCII numerical form.

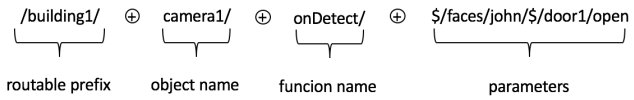

Fig. 2. Proposed named schema

B. Name publishing and advertising

The introduced naming scheme is inherently supported by NDN nodes. However, in the IoT network, devices can come and go, so it is imperative to handle things management in an automatic way. In the ICN context this requirement is fulfilled thanks to the routing protocol that dynamically advertises the names published by the various objects in the network, enforcing all necessary security requirements ad discussed in sec. IV.

More in details, the node publishing a function on the routing information base (producer) is also the one responsible of allowing this function to run and to produce Data Packets for the consumer node. To this aim, the producer leverages the ICN routing daemon such as the NLSR link state routing protocol [19] to advertise the new name towards the other nodes of the network. The selected naming scheme structure allows exploiting the scalability features of the NLSR protocol since it includes the routable prefix part. Similarly to many other routing protocols, NLSR uses a link cost in order to choose the best route for the destination node. This functionality is used in the LNO architecture to give a numerical *priority* to names so that Interest packets will be forwarded to the advertising node having a greater priority. This enables the update feature illustrated in section II: to update a function the new publisher, using the right security credentials, advertises the same name with the routing protocol but with an higher priority than the legacy function. In this way the network layer will route the Interest packet towards the new "updated" function producer. Note that this strategy exposes the network to a name attack: for example, by overloading the "/camera1/stream" name a malicious node could hijack the security camera video flows from the actual captured stream to a recorded tape. Thus, it is important to prevent unauthorized nodes to publish names on the routing information base. We address this requirement by using NDN and NLSR security features as described in section IV.

C. Object synchronous interaction

When a node has to interact with a LNO it sends an Interest packet with a suitable name, according to the specified naming schema. Every router receiving the interest is able to forward the request to the node that is able to perform the requested action. The simpler interaction that can take place in LNO is the synchronous function invocation showed in figure 3. The controller sends an interest containing the name of the object and of the invoked function. In figure 3 it invokes the `/camera1/status` and then the `/light1/on` functions. The invoked objects execute the command and return back the results in a Data packet for instance with a JSON data structure.

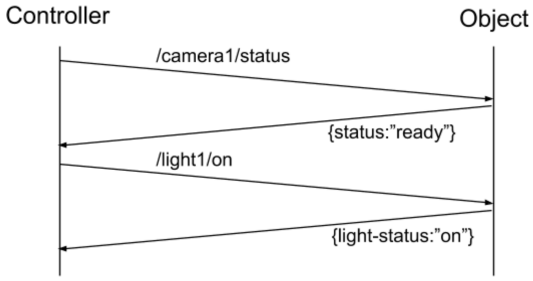

Fig. 3. Message sequence diagram for LNO synchronous function invocation

D. Objects asynchronous interactions and chaining

Very often in IoT systems we need to specify behaviors of the network in reaction to some triggering events. For instance, if we want to open a door only when an employee is recognized to be in front of the door, then we need to set a trigger action for that event. This functional chaining pattern, common in flow programming, requires the implementation of the so called *callbacks*. In the LNO architecture a callback is a name corresponding to a function that has to be called in reaction to a specific event. For instance, as depicted in figure 4, we can decide to call `/door1/open` when `camera1` detects a given face.

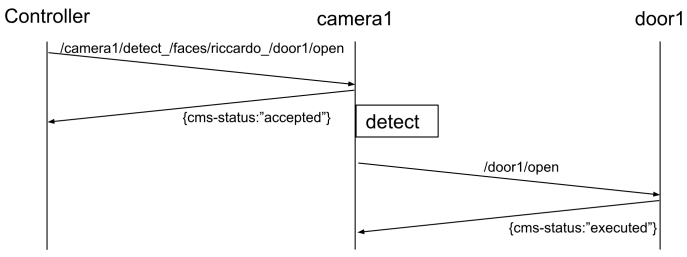

Fig. 4. Message sequence diagram of a LNO asynchronous function invocation and functional chaining

According to the presented named schema, the controller sends an Interest packet with the name `/camera1/detect_/faces/riccardo_/door1/open`.

The returned Data carries an acknowledgment string to notify that the request is taken in charge by the LNO, or may contain application layer error reporting. Each function can receive as parameters many different values, which can correspond to integers, strings or NDN Names. The definition of the parameter and of their types are function-dependent. We remark the possibility of chaining reactions. For instance, we can also program this callback event in addition to the previous one. As showed in figure 4, once a given face is detected by a camera, the door is opened, to let the employee access to the building.

Algorithm 1 Controller side

```
1: function INVOKE OBJECT FUNCTION(o, f, p_1 ... p_n)   ▷ Where o is the object name, f the function name and p_1 ... p_n the function parameters
2:     name = setupName(o, f, p_1, ... p_n)
3:     sendInterest(name)
4:     while not isDataArrived do:
5:         Wait()
6:     end while
7:     passDataToApplication()
8: end function
```

Algorithm 2 Object side

```
1: function EXECUTE LOCAL FUNCTION(I)   ▷ Where I is the received Interest
2:     o ← getObjectName(I)
3:     f ← getFunctionName(I)
4:     p_1 ... p_n ← getFunctionParameters(I)
5:     if isSynchronous(f) then
6:         res = f(p_1 ... p_n)              ▷ execute the function
7:         sendData(res)
8:     else
9:         sendAcknoledgeData()
10:        f(p_1 ... p_n)                    ▷ execute the function
11:        while functionIsRunning do:
12:            if AsyncronousNotification() then
13:                SendInterest(p_i)         ▷ p_i = callback name
14:                while not isDataArrived do:
15:                    Wait()
16:                end while
17:            end if
18:        end while
19:    end if
20: end function
```

Algorithms 1 and 2 describe the actions performed by the controller and by the object, respectively. The main task of the controller is to create an Interest name according to the specified naming schema and to send an Interest packet in the ICN network. On the object side, we distinguish if the function call is related to a synchronous or an asynchronous interaction. In case of synchronous interaction, after getting the object name, the function name and the (optional) parameters from the Interest name, the object executes the function and sends back to the controller the result. In case of asynchronous interaction, the object sends an acknowledgment data and starts executing the function. Then, when the function needs to asynchronously notify state changes through a callback (e.g., when a camera detects a target), it sends an Interest packet. We remark that objects can receive Interest packets from both controllers and other objects in case of function service chaining.

IV. SECURITY

In NDN, security is built-in, through a data-centric approach. Authorization and authentication properties and procedures are defined and applied for and to each named data; a named based trusted schema is also defined to implement them in a distributed form [5]. In what follows we limit ourselves to describe the security aspects related to the construction of the LNO, referring to the specific literature (such as [5], [18], [20] and the survey [21]) for the general issue of security in NDN.

A. Authorizing LNO interactions

In every IoT environment it is necessary to properly define the authorization procedures, allowing only selected objects to interact with other selected objects to create the smart environment. To this aim we need a trusted entity that, in our case, is the owner of the object. We remark that the owner could be a person, company or a public administration. The owner is responsible of the behavior of the objects and defines the security and authorization rules that regulate the objects interactions.

Let us consider for example, the security operations required to control the status of a television named `/tv1` on behalf of a remote controller named `/remote1`:

1 When the owner buy the objects she/he signs their public keys. The names of the resulting certificates will be used by both objects as their *key-locators* as described in [5]. This will demonstrate that the objects belong to a given owner and allows the construction of custom security validators.
2 Then the owner certificate is added to a list called *authorized-operators*, stored inside each object. This allows the owner to add further certificates to these objects in the future.
3 Finally, the remote controller certificate is added to a list called *allowed-entities* of the television and vice versa. This operation must be executed by an authorized operator.

The NDN/NLSR infrastructure allows to easily enforce such rules by defining a specific regular expression-based language. To enforce the rules of the described example, the permission check syntax is expressed by the following fragment of configuration script:

```
rule {
 id "LNO interest access control"
 for interest
 filter {
   type name
   regex <tv1><on>$
```

```
 }
 checker {
   type hierarchical
   sig-type rsa-sha256
 }
 trust-anchor {
   type dir
   file-name /authorized
   refresh 1h;
 }
}
```

B. Supporting LNO augmentation

Object augmentation allows network entities to provide methods for other objects, by publishing a name in the object namespace. This enables to enhance objects with new functions or override existing functions. The new "augmented" functions are announced on the routing plane, but a protection is needed against unauthorized object augmentation. To this aim, we devised an NDN/NLSR rule that enables network verification on nodes that announce names with a certain prefix. This rule is implemented as follows:

```
rule {
 id "LNO augmentation rule"
 for interest
 filter {
  type name
  regex ^<localhost><nlsr><prefix-update>
    [<advertise><withdraw>]<>*$
 }
 checker {
  type customized
  sig-type rsa-sha256
  key-locator {
   type name
   hyper-relation {
    k-regex ^(<>*)<KEY><>*<ksk-.*><>*$
    k-expand \\1
    h-relation is-prefix-of
    p-regex ^<localhost><nlsr><prefix-update>
            [<advertise><withdraw>](<>*)$
    p-expand \\1
   } } } }
```

This rule establishes a relationship between the announced/withdrawn name and the key name used for signing the command interest: the *filter* matches the NLSR Control Command Interest for advertisement/withdrawal and the *checker* checks if the left-hand part of the KeyLocator (before the "KEY" component) corresponds to the prefix of the announced/withdrawn name. The name of the public key required to enable signature verification is specified in the KeyLocator [2].

An entity that has to add a functionality (e.g. `/camera1/detect`) to an LNO needs the authorization from the owner of the parent object (e.g. `/camera1`), before being allowed to publish its method on the global routing plane. The owner of the parent object name prefix provides the other entity with a signed certificate that is used to publish that specific name under the same object prefix.

V. Proof of Concept

In this section we describe some IoT use cases that we implemented using the Lightweight Named Object concept, to show how it can simplify the way to interact with "things" and among "things". The proposed uses cases have been implemented by using NDN and in particular using NFD v0.5.1. [3].

A. Synchronous interaction: controlling a smart lamp

We started by implementing a synchronous interaction in the case of a smart lamp implementing the network architecture depicted in figure 5. In particular, we developed an NDN node that publishes the LNO interface of a Philips HUE smart lamp advertising the names corresponding to the lamp APIs and listening for action requests coming in the form of NDN Interest packets. Basically, the node translates the HUE proprietary protocol requests and responses into NDN Interests and Data packets.

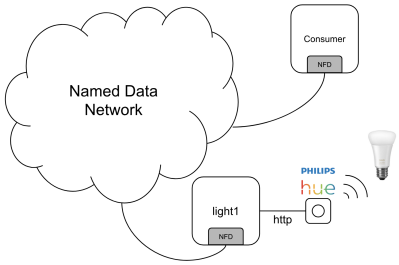

Fig. 5. Architecture of the smart lamp control use case

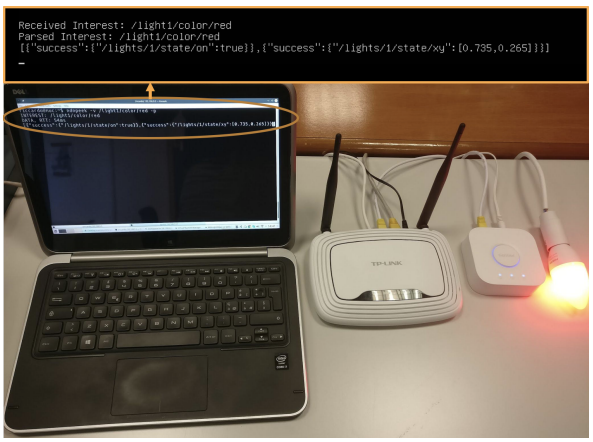

Fig. 6. Testbed setup, composed by two NDN nodes installed on a laptop and a smart lamp

We implemented the LNO `/light1` by using an LNO abstraction that offers two easy synchronous methods to switch on the lamp and to set the color of the smart bulb, as described in the flow diagram of figure 7. We also implemented a test client using a standard NDN consumer that can interact with

[2]Currently, the NDN validation system does not allow to create security rules with a relationship among the name announced/withdrawn and the name of the key used to sign the Command Interest. The name to be announced is encoded in the name of the Command Interest and the actual NDN validator engine does not consider encoded name components. To extract announced/withdrawn name from the command interest name, we had to implement a patch in the ndn-cxx library, specifically in the checkRelation method of the KeyLocatorChecker class, which extracts the name to announce/withdraw from the TLV encoded component of the command interest

[3]https://named-data.net/doc/NFD/0.5.1/

the lamp by invoking named methods such as `/light1/on` or `/light1/color/#FE00EA`. The testbed implementation is shown in figure 6.

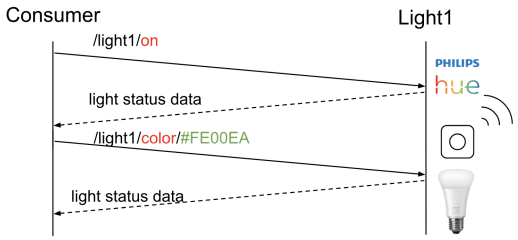

Fig. 7. Smart lamp NDN sequence diagram: switch on and change color operations

B. Camera detection and asynchronous calls

To assess the call chaining features we devised a more complex use case: switching on a light once a given person is recognized by a camera. To this aim we setup the network architecture depicted in figure 8.

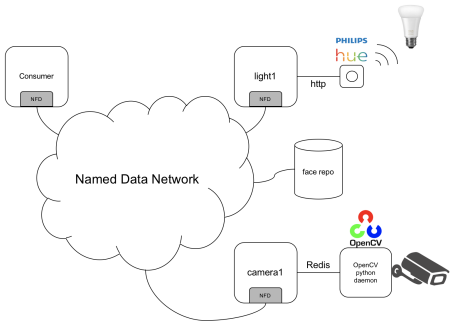

Fig. 8. Architecture of the camera detection use case

We connected two other NDN nodes to the previous network: a custom NDN node publishing the LNO interface of the video camera (named `/camera1`), and an NDN standard repository publishing the faces (named `/faces`). Clearly, this scenario requires an asynchronous interaction among two LNOs, the light and the camera, showing how this interaction can be easily achieved using the LNO abstraction. Moreover the `camera1` is augmented with the `detect` function. This method accepts as parameters the face to detect and the action to do after detection, both of them in the form of names.

The detection is performed by using recent machine learning algorithms that can be possibly substituted/updated in the future. In the implemented scenario a deep learning algorithm is trained by using the image of the face specified by the first parameter and fetched through the NDN network by some node caches or by the face repo. The detection activity performed by the object (or near the object, for instance following the approach of Krol et al. in [14]) offers the benefit of implementing the principle of *data locality* avoiding that the whole video stream flows from the camera device to a centralized cloud to be processed.

Then, when the camera detects a person, with a certain level of probability, it sends an Interest towards the name specified by the second parameter: in our case this action will switch on the light with the color specified by the parameter. Finally, the third parameter specifies another recipient of the notification; in our case we use this name to notify the consumer that a detection took place. The producer publishes a new name using a "nonce" and the consumer can retrieve the details of the detection using that unique name (e.g. the name of the detected persons together with the details of the camera that detected that person). An example of interaction among the NDN nodes is reported in figure 9, showing all the names in the Interest and Data packets.

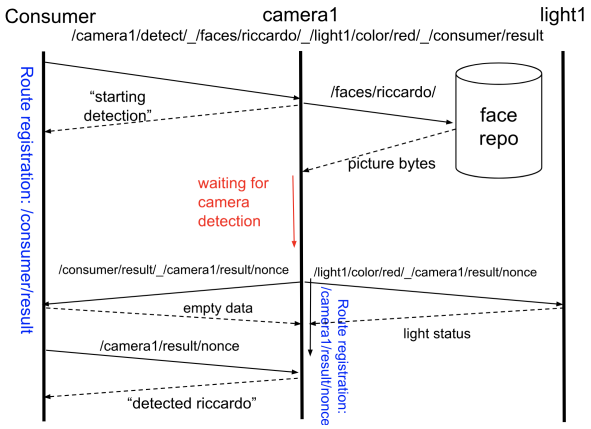

Fig. 9. Flow diagram of the camera detection use case

The proposed architecture allows to natively support multiple cameras and multiple consumers and detect multiple persons. A potential extension of this use case is thus showed in figure 10 where we depict four LNO objects corresponding to four different security cameras named `/camera1` ... `/camera4` and two security operators `/operator1` and `/operator2`. Operator1 is interested in all video streaming from the cameras detecting Alice while Operator2 is interested in cameras detecting Alice or Bob. Clearly the NDN built-in caching and multicast features allow simplifying and optimizing all the nodes interactions. Moreover, using the same architecture of figure 8, operators can be notified about the detection and they can start fetching the video stream using a distributed strategy orchestrated by NDN. The NDN network can provide the video stream detecting Alice to all the security operators at the same time, leveraging on the NDN multicast functionality. Moreover packet filtering can be performed as close as possible to the video, avoiding bandwidth waste and allowing easy programmability.

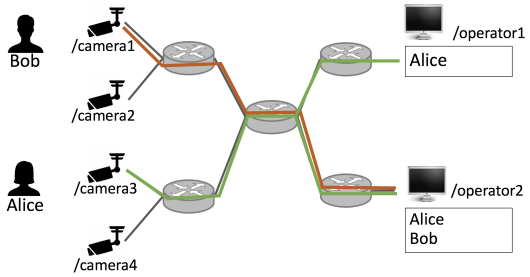

Fig. 10. Example of a many-to-many use case, where LNO is used for surveillance

C. Interactive shell for objects configuration and maintaining

The LNO architecture has been designed to support existing IoT frameworks. However, it also allows a direct programming of objects. Thus, we designed a custom IoT shell running on a web page. The shell communicates with NDN nodes using a web-socket and it is able to "translate" simple scripts in a set of NDN interests. Each Interest packet represents a function and it is forwarded by the routing plane to the NDN object able to execute it. Responses are echoed back on the web page.

The keyword "NOTIFY" is used to implement the asynchronous function invocations and thus to create parallel requests: if this keyword is present, the application starts the execution of the IF/ENDIF block in a separate thread, in order to let the user to execute other commands at the same time. Otherwise, commands are executed one by one.

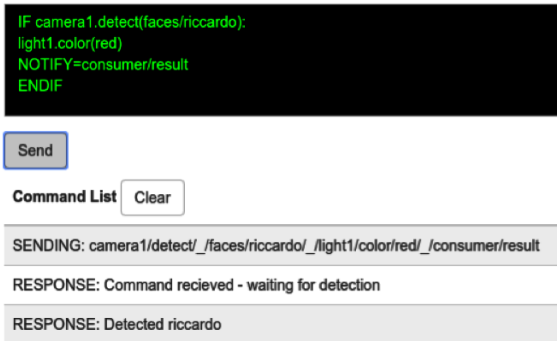

Fig. 11. Example of the LNO interactive shell used to program the behavior described in section V-B

In the example of figure 11 we show the commands needed to implement the use case presented in the previous subsection. The semantic associated to the NDN LNOs provides an easy way to program, debug and maintain the above configuration.

VI. Results and Discussion

In this section, we compare LNO and functional approaches with the aim of assessing their impact in an ICN network, in terms of forwarding load. A pivotal characteristic of LNO is that augmented functions are exposed as object methods rather than as global functions (with the object data as parameter), as in case of NFN [15]. At the ICN layer this characteristics implies the use of hierarchical names versus flat names, to access augmented functions. This, in turn, influences the number of routing entries in the ICN Forward Information Base (FIB) needed to support the system, which is an important performance indicator determining scalability and even feasibility of ICN applications [22]. Thus, we compare the two different choices, LNO and global augmented functions (GAF), in terms of average number of required FIB entries per node.

To this aim, we simulate a system of objects (sensor/actuators) interconnected by an ICN network. We generate a graph composed by $N = 500$ nodes connected by a scale-free network topology with parameters $\alpha = 0.41$, $\beta = 0.54$ and $\gamma = 0.05$ [23]. We assume that leaf nodes represent objects and that internal nodes have enough computational capacity to execute both ICN processing and augmented functions. The resulting network has $L = 284$ leaf nodes (i.e. objects) and 216 internal nodes.

We divided the objects in G groups: the objects of the same group expose the same set of internal functions and M augmented functions. To simulate the fact that some objects may be more popular than others, the groups cardinality follows a Zipf distribution (power law) with parameter $\alpha = 1.1$.

For instance a group of 5 temperature sensors with only `getCurrentTemperature` internal functions may be deployed on five distinct leaf nodes and may request to the network internal nodes to execute $M = 3$ augmented functions, namely: `getAverageTemperature` `getMinTemperature` `getMaxTemperature`.

We considered two different deployment strategies for the augmented functions:

- **Edge**: the functions are deployed on the leaf *gateway*, i.e., the inner node that is directly connected to the leaf.
- **Server**: all the functions are deployed on a single server node (e.g. hosted in a central cloud) that is the one with the greatest "betweenness centrality".

To build the FIBs, we used the following approach. Each leaf node has a pre-configured default FIB entry ("/") towards its gateway and an entry for the name of the object it hosts (e.g. `/temp-sensor1`), which points to the local application entity handling all the object internal functions. Such an object entry is also announced to the ICN routing plane.

Internal nodes hosting augmented functions have an entry per function in their FIB, whose naming scheme is the following: the augmented function name (e.g. `/getMinTemperature`) in case of GAF; the augmented function name under the naming subtree identified by the object name (e.g. `/temp-sensor1/getMinTemperature`) in case of LNO. Such function entries are also advertised on the ICN routing plane.

A named-based version of the Dijkstra routing algorithm computes the routes toward all published names for each network node. We remind that ICN forwarding adopts a longest prefix matching (LPM) forwarding strategy, but applied to names, rather than to bits of IP addresses. As a consequence, many routing entries are redundant and we removed them for building the final FIB. We call this procedure *route-aggregation* (similar to IP supernetting). For instance, if a node has an entry for `/temp-sensor1` and an entry for `/temp-sensor1/getMinTemperature`, and both entries have the same next-hop (aka output face) then the second entry can be removed since Interest messages for `/temp-sensor1/getMinTemperature` are properly forwarded by the `/temp-sensor1` entry too. If the next-hops were different, the entry removal would have not been possible.

In figure 12 we plot the average number of entries in the FIB of a generic network node. We varied the number of groups G in case of $L = 284$ objects and $M = 10$ augmented functions per group. For both kinds of LNO deployments the number of FIB entries does not increase with the number of groups, indeed, for LNO, what actually impacts the FIB size is the total number $L \times M$ of per-object augmented function names

(e.g. `/temp-sensor1/getMinTemperature`) and the number L of object names (e.g. `/temp-sensor1`), whose values are constant in this simulation. The big difference in FIB size between the Edge and Server cases is due to the fact that in the Edge deployment the route-aggregation process practically cancels the effect of $L \times M$ contribution, by removing the per-object augmented function names from the FIBs in a large fraction of network nodes. Actually, a per-object function name only remains in the FIB of the leaf node hosting the object and on its gateway.

In case of GAF what impacts the FIB size is the total number of augmented functions names (e.g. `/getMinTemperature`), i.e. $G \times M$, and the number L of objects names. We observe that Edge and Server deployments lead to the same results and this will be the case also for other performance measures addressed below. This is due to the fact that any FIB must have the routes towards any augmented function and object name (even though pointing to different next-hops in the different deployments) and route-aggregation is not possible since the related names are flat.

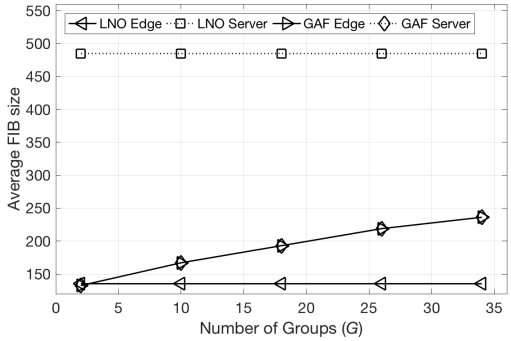

Fig. 12. Average number of FIB entries for different deployment strategies by varying the number of groups

In figure 13 we repeat the same analysis by increasing the network size from 300 to 1000 nodes, and thus varying the number L of objects (leaf nodes) too, as reported in the x-axis of the figure. Number of groups and of augmented functions per group are $G = 20$ and $M = 10$, respectively. In any case, we see an almost linear increase of the average FIB size. Indeed, as discussed before, the FIB size of any solution increases with L and with other possible contributions too. More precisely: $L \times M$ for LNO-Server, justifying the slope increasing with respect to LNO-Edge; and $G \times M$ in case of GAF, justifying the almost constant difference with respect to LNO-Edge.

Figure 14 reports the average FIB size versus the number of augmented functions M, in case of $G = 20$ and $L = 284$. The plot confirms the LNO and GAF dependencies from the parameters discussed above.

To sum up, our analysis shows that LNO scales better with respect to GAF from an ICN point of view, when we have the need (e.g., for low latency requirements) or the opportunity of deploying augmented functions on the edge of the network; this is because it is possible to exploit LNO hierarchical naming and ICN forwarding LPM to aggregate FIB tables. Conversely, when augmented functions have to be deployed

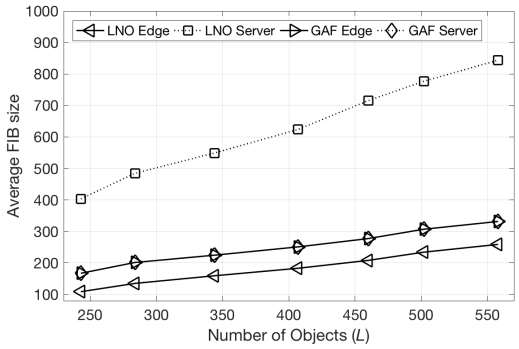

Fig. 13. Average number of FIB entries for different deployment strategies by varying the number of objects (and network size)

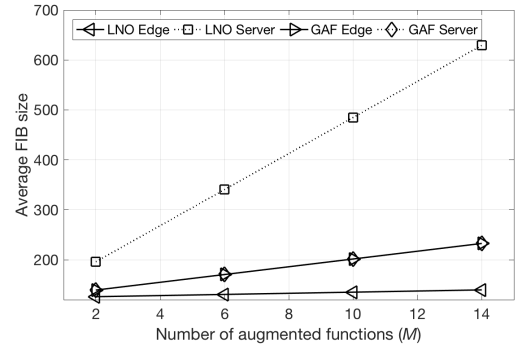

Fig. 14. Average number of FIB entries for different deployment strategies by varying the number of augmented functions per group

in a central node (e.g., a central cloud) the GAF approach provides a better ICN scalability.

VII. Related work

Named Data, Named Function, Named Object

Information Centric Networking (ICN) has been proposed by Van Jacobson in 2006 [24] and then widely investigated in the literature. One of the most remarkable result in the field is the design and implementation of the Named Data Networking (NDN) architecture [25]. The basic brick of the architecture is the concept of named data, basically any kind of information chunk identified by a unique name. Clients (consumers) express their willingness to retrieve a given data by emitting an Interest packet containing the name of the data. The network is in charge of delivering back to the requesting client the Data packet corresponding to that name.

Later on, other researchers proposed Named Function Networking (NFN) [15], as an ICN style where "*the request must carry at least two names in order to be satisfied*" [26]. Proposed in 2012 and with the first prototype delivered in 2014, NFN, instead of using names to organize the access to data, it uses them to access and invoke functions, which may incidentally produce passive content (data). NFN starts from the assumption that there exists many raw data but clients usually desire customized elaborations on such data. The NFN applications request such data elaboration by Interest packets. The network consequently invokes the execution of the function (called lambda function) that produces the output eventually delivered to the requisting client [15]. This

approach allows clients to name the desired result, server-agnostically, while the network is in charge of finding execution places and caches the results.

In this work, we propose to extend the concept of NFN not only to operate on the data produced by IoT objects, but exposing an easy interface to programmers, allowing programmers to move from functional programming to object oriented programming. This requires methods to manage objects and "enrich" objects with functionality by using the namespace provided by ICN to organize objects and services. For instance, in our proposed evolution from the NFN name `/mapReduce(/sensors(/myhouse,temp,2014),/avg)` to the LNO name `myhouse/sensors/temp/avg(2014)`, the `avg` function is seen as a "method" of the temperature sensors, and the temperatures sensors are unaware of which function is used for calculating the average or for retrieving historical data.

Object oriented distributed computing

There are many frameworks for distributing objects on different hosts and for coordinating them. These frameworks have a long history and some of them dates back to the late 80s. One of the oldest is CORBA (Common Object Request Broker Architecture) an OMG standard designed to facilitate computer communications through a middleware for distributed computing. Differently from CORBA objects, LNOs do not require any connection among the parts composing the objects and rely on a different networking technology (ICN) that natively offers the namespace where the object live, rather than requiring a dedicated middleware. Also, remote procedure calls have been investigated for a long time; probably the most successful implementations are Java RMI (Remote Method Invocation) and Apache River (former Jini [27]). Apache River is a network architecture for the construction of distributed systems in the form of modular co-operating services; differently from Java RMI, in Apache River services are connected to clients through a lookup service (dynamic multicast discovery) to which they have to connect to. In this way clients do not need to know where the services are located. For the security aspects, the Kerberos protocol provides network authentication services to clients. Its behavior requires clients' time synchronization and the presence a central server.

ICN for IoT

Information Centric Networking offers a clean slate alternative that natively supports multicast, mobility, and content oriented security, as well as an organizing namespace. For these reasons it has been proposed, also in standardization bodies [13], to address part of the IoT challenges [4]. ICN usage for IoT has been investigated e.g. to provide solutions to device organization [6], [7], [8], [28] and in-network data processing [6] (e.g. to compute the average among different values coming from various sensors in order to limit the data flowing, programming a kind of stream analytic). ICN has also been proposed together with novel light authentication schemes [11], [12] and routing [11], [9], exploiting also its native in-network caching functionality [29].

Named data solutions for in-network computation

In [14] authors present Named Function as a Service (NFaaS), a framework that extends the Named Data Networking architecture to support in-network function execution. NFaaS builds on very lightweight VMs and allows for dynamic execution of custom code. This work is in part inspired by the previous Service Centric Networking (SCN) [30] which focus on how to create processing workflows inside an ICN network through a manipulated concatenation of names that identify network services in the network. Even though we share the same vision of objects with SCN, we are more similar to NFN [15] as regards the role of the network of orchestrating the function execution server in an agnostic way. Indeed, in our work we aim to show potential benefits of our solution in IoT environments, where real objects constitute a name basis "for humans" even if the processing of an object data can be done inside different network entities that could be situated in different locations in the network. Finally, our objects are in general state-full as, in our case, the function call may change the internal state of the object. Finally, recently, ICN has been also proposed for implementing micro-service architectures [31].

VIII. Conclusions

In this work, we presented an abstraction named Lightweight Named Object (LNO), based on the Information Centric Networking architecture. LNO provides a convenient abstraction of real objects supporting easy discovery, programming and management procedures for real or virtual Internet of Things devices. The proposed abstraction fully exploits most of the ICN benefits and can be used to support objects interaction in IoT environments. To this aim, we devised a hierarchical naming schema, which is specifically designed for the NDN framework, to support all the LNO capabilities. The proposed naming schema allows the diffusion of names by means of the NLSR routing protocol. Thanks to NDN and NLSR, we can enforce all the security procedures required in IoT scenarios, to avoid objects takeover and preventing unauthorized access. Specific rules have been defined for the NDN forwarding agents and for the NLSR routing agents. A Proof of Concept has been implemented using commercial IoT products and showing objects augmentation and function chaining, as well as an interactive web socket console able to interact with LNO-enabled things. Performance evaluation in a simulated network scenario shows how LNO is a viable solution, especially in edge computing scenarios, where it improves the simplicity of network service programmability and exhibits good scaling capabilities with respect to the number of objects and of augmented functions.

References

[1] A. Zanella, N. Bui, A. Castellani, L. Vangelista, and M. Zorzi, "Internet of things for smart cities," *IEEE Internet of Things journal*, vol. 1, no. 1, pp. 22–32, 2014.

[2] A. Botta, W. De Donato, V. Persico, and A. Pescapé, "Integration of cloud computing and internet of things: a survey," *Future Generation Computer Systems*, vol. 56, pp. 684–700, 2016.

[3] I. Ishaq, D. Carels, G. K. Teklemariam, J. Hoebeke, F. V. d. Abeele, E. D. Poorter, I. Moerman, and P. Demeester, "Ietf standardization in the field of the internet of things (iot): a survey," *Journal of Sensor and Actuator Networks*, vol. 2, no. 2, pp. 235–287, 2013.

[4] W. Shang, Y. Yu, R. Droms, and L. Zhang, "Challenges in iot networking via tcp/ip architecture," *Tech. Report NDN-0038. NDN Project*, 2016.

[5] Y. Yu, A. Afanasyev, D. Clark, V. Jacobson, L. Zhang *et al.*, "Schematizing trust in named data networking," in *Proceedings of the 2nd International Conference on Information-Centric Networking*. ACM, 2015, pp. 177–186.

[6] M. Ascigil, S. Reñé, G. Xylomenos, I. Psaras, and G. Pavlou, "A keyword-based icn-iot platform," in *Proceedings of the 4th ACM Conference on Information-Centric Networking*. ACM, 2017, pp. 22–28.

[7] S. S. Adhatarao, M. Arumaithurai, D. Kutscher, and X. Fu, "Isi: Integrate sensor networks to internet with icn," *IEEE Internet of Things Journal*, 2017.

[8] S. Arshad, B. Shahzaad, M. A. Azam, J. Loo, S. H. Ahmed, and S. Aslam, "Hierarchical and flat based hybrid naming scheme in content-centric networks of things," *IEEE Internet of Things Journal*, 2018.

[9] Z. Yan, S. Zeadally, and Y.-J. Park, "A novel vehicular information network architecture based on named data networking (ndn)," *IEEE Internet of Things Journal*, vol. 1, no. 6, pp. 525–532, 2014.

[10] A. M. Alberti, G. D. Scarpioni, V. J. Magalhaes, S. A. Cerqueira, J. J. Rodrigues, and R. d. R. Righi, "Advancing novagenesis architecture towards future internet of things," *IEEE Internet of Things Journal*, 2017.

[11] T. Mick, R. Tourani, and S. Misra, "Laser: Lightweight authentication and secured routing for ndn iot in smart cities," *IEEE Internet of Things Journal*, 2017.

[12] W. Shang, A. Bannis, T. Liang, Z. Wang, Y. Yu, A. Afanasyev, J. Thompson, J. Burke, B. Zhang, and L. Zhang, "Named data networking of things," in *Internet-of-Things Design and Implementation (IoTDI), 2016 IEEE First International Conference on*. IEEE, 2016, pp. 117–128.

[13] Y. Zhang, D. Raychadhuri, R. Ravindran, and G. Wang, "Icn based architecture for iot," *IRTF contribution, October*, 2013.

[14] M. Król and I. Psaras, "Nfaas: named function as a service," in *Proceedings of the 4th ACM Conference on Information-Centric Networking*. ACM, 2017, pp. 134–144.

[15] M. Sifalakis, B. Kohler, C. Scherb, and C. Tschudin, "An information centric network for computing the distribution of computations," in *Proceedings of the 1st ACM Conference on Information-Centric Networking*, ser. ACM-ICN '14. New York, NY, USA: ACM, 2014, pp. 137–146. [Online]. Available: http://doi.acm.org/10.1145/2660129.2660150

[16] C. Simonyi, "Intentional programming: Innovation in the legacy age," in *IFIP Working group*, vol. 2, 1996, pp. 1024–1043.

[17] E. Baccelli, C. Mehlis, O. Hahm, T. C. Schmidt, and M. Wählisch, "Information centric networking in the iot: experiments with ndn in the wild," in *Proceedings of the 1st ACM Conference on Information-Centric Networking*. ACM, 2014, pp. 77–86.

[18] L. Zhang, D. Estrin, J. Burke, V. Jacobson, J. D. Thornton, D. K. Smetters, B. Zhang, G. Tsudik, D. Massey, C. Papadopoulos *et al.*, "Named data networking (ndn) project," *Relatório Técnico NDN-0001, Xerox Palo Alto Research Center-PARC*, 2010.

[19] A. Hoque, S. O. Amin, A. Alyyan, B. Zhang, L. Zhang, and L. Wang, "Nlsr: named-data link state routing protocol," in *Proceedings of the 3rd ACM SIGCOMM workshop on Information-centric networking*. ACM, 2013, pp. 15–20.

[20] B. Hamdane, A. Serhrouchni, A. Fadlallah, and S. G. El Fatmi, "Named-data security scheme for named data networking," in *Network of the Future (NOF), 2012 Third International Conference on the*. IEEE, 2012, pp. 1–6.

[21] S. Chen and F. Mizero, "A survey on security in named data networking," 12 2015.

[22] A. Detti, M. Pomposini, N. Blefari-Melazzi, and S. Salsano, "Supporting the web with an information centric network that routes by name," *Computer Networks*, vol. 56, no. 17, pp. 3705–3722, 2012.

[23] B. Bollobás, C. Borgs, J. Chayes, and O. Riordan, "Directed scale-free graphs," in *Proceedings of the fourteenth annual ACM-SIAM symposium on Discrete algorithms*. Society for Industrial and Applied Mathematics, 2003, pp. 132–139.

[24] V. Jacobson, D. K. Smetters, J. D. Thornton, M. F. Plass, N. H. Briggs, and R. L. Braynard, "Networking named content," in *Proceedings of the 5th international conference on Emerging networking experiments and technologies*. ACM, 2009, pp. 1–12.

[25] L. Zhang, A. Afanasyev, J. Burke, V. Jacobson, P. Crowley, C. Papadopoulos, L. Wang, B. Zhang *et al.*, "Named data networking," *ACM SIGCOMM Computer Communication Review*, vol. 44, no. 3, pp. 66–73, 2014.

[26] C. Tschudin, "Named function networking: Service chaining, big picture, division of labor boundaries," year, iRTF ICNRG interim meeting, Boston, Jan 2015.

[27] K. Arnold, R. Scheifler, J. Waldo, B. O'Sullivan, and A. Wollrath, *Jini specification*. Addison-Wesley Longman Publishing Co., Inc., 1999.

[28] S. Arshad, M. A. Azam, S. H. Ahmed, and J. Loo, "Towards information-centric networking (icn) naming for internet of things (iot): The case of smart campus," ser. ICFNDS '17. New York, NY, USA: ACM, 2017.

[29] C. Gündogan, P. Kietzmann, T. C. Schmidt, M. Lenders, H. Petersen, M. Wählisch, M. Frey, and F. Shzu-Juraschek, "Information-centric networking for the industrial iot," in *Proc. of 4th ACM Conference on Information-Centric Networking (ICN), Demo Session. New York, NY, USA: ACM*, 2017, pp. 214–215.

[30] T. Braun, V. Hilt, M. Hofmann, I. Rimac, M. Steiner, and M. Varvello, "Service-centric networking," in *2011 IEEE International Conference on Communications Workshops (ICC)*, June 2011, pp. 1–6.

[31] K. B. Long, H. Yang, and Y. Kim, "Icn-based service discovery mechanism for microservice architecture," in *Ubiquitous and Future Networks (ICUFN), 2017 Ninth International Conference on*. IEEE, 2017, pp. 773–775.

Lorenzo Bracciale has been an Assistant Professor in the University of Rome Tor Vergata since June 2013. He obtained his Ph.D. with a thesis on peer-to-peer multimedia communications. He collaborated with several companies and other researchers worldwide for projects regarding delay tolerant networking and mobile applications. His current research interests include autonomous and self-organizing systems, either in Wireless Sensor Networks or in Mobile Networks.

Pierpaolo Loreti received the Laurea Degree in Electronic Engineering (cum laude) in July 1998 and the PHD degree in telecommunications and microelectronics in June 2002, from the University of Rome Tor Vergata. From 2002 to 2005 we was a Researcher of "Consorzio Nazionale Interuniversitario per le Telecomunicazioni" (CNIT). From 2006 he is a Researcher of the Dep. of Electronic Engineering and an Aggregate Professor at the Internet Engineering Laurea at the University of Roma Tor Vergata.

Andrea Detti is a professor of the University of Rome Tor Vergata. His research activity spans on different topics in the area of computer networks and copes with framework design, analytical modeling, performance evaluation through simulation and testbed. He is co-author of more than 60 papers on journals and conference proceedings, mainly regarding information- centric, software-defined, overlay, wireless and optical networks. Currently is the research area is focused on Information-Centric-Network (ICN), Software Defined Networks (SDN).

Riccardo Paolillo graduated in Computer Engineering, he is CNIT researcher and assistant at the University of Rome Tor Vergata. He has developed professional software since 2010. He has created native iOS apps, backend in Java EE with Spring, video analysis systems in C# with OpenCV, backend and machine learning in Python and he is a C programmer in the Linux kernel. Now sole administrator of ABLE srls software development company.

Nicola Blefari Melazzi is a Full Professor of Telecommunications at the University of Rome Tor Vergata, Italy. He is the Director of CNIT a consortium of 37 Italian Universities. Dr. Blefari-Melazzi has participated in over 30 international projects; he has been the principal investigator of five cooperative EU funded projects. He has been an evaluator for many research proposals and a reviewer for numerous EU projects. He is author/coauthor of about 200 papers, in international journals and conference proceedings. His research interests include the performance evaluation, design and control of broadband integrated networks, wireless LANs, satellite networks and of the Internet.