A Realistic Deployment of Named Data Networking in the Internet of Things

Amar Abane, Mehammed Daoui, Samia Bouzefrane, Soumya Banerjee, Paul Mühlethaler

A Realistic Deployment of Named Data Networking in the Internet of Things

Amar Abane[1,2], Mehammed Daoui[1], Samia Bouzefrane[2], Soumya Banerjee[2], and Paul Muhlethaler[3]

[1]LARI Lab. University Mouloud Mammeri of Tizi-Ouzou, Algeria
[2]CEDRIC Lab. Conservatoire National des Arts et Métiers, Paris, France
[3]Inria EVA, Paris, France

Abstract

IP has been designed for Internet decades ago to connect computers and share expensive resources such as tape drives and printers. Nowadays, Internet of Things and other emerging applications use Internet to fetch and exchange content such as monitoring data and movies. This content-centric use of Internet highlights the limitations of the IP architecture. IETF Working Groups spend significant efforts to adapt the traditional IP stack to IoT systems, but the shortcomings of IP remain difficult to hide. In this context, the recently emerged Named Data Networking (NDN) architecture promises a better support of IoT systems and future Internet applications. This paper describes a realistic IoT architecture based on NDN. In practice, an integration of NDN in IoT devices over low-power wireless technologies is designed, deployed and evaluated considering a Smart Farming application scenario. This work aims to show that NDN is more suitable than IP for IoT systems, by giving another look at IP-based solutions for the IoT such as 6LoWPAN. For that, we design a simple packet compression scheme and a lightweight forwarding strategy that is compliant with the NDN vision while managing constrained devices. Evaluation result demonstrate the flexibility of NDN to support IoT environments.

Keywords: IoT, ICN, NDN, IEEE 802.15.4, Smart Agriculture, Wireless Mesh Networks.

1 Introduction

The Internet of Things (IoT) uses the interconnection of billions of small computing devices, called "Things", to provide access to services and information all over the world. This has been made possible by the democratization of smartphones, and more importantly by the affordability of resource-constrained data acquisition devices and corresponding wireless communication technologies.

In practice, IoT devices are powered by battery, have tens to thousands of *MHz* CPU and tens to thousands of *Kb* memory. These resource-limited devices communicate with user applications over the Internet through gateways. The global communication is achieved through the IP protocol suite and the device-gateway communication is based on low-power and lossy wireless technologies such as IEEE 802.15.4 [12] and IEEE 802.15.1 [11]. However, the IP protocol suite has been designed decades ago for a completely different purpose, and IoT features now highlight the limitations of IP [28]. For example, security is still focused on communication channels when the data itself need to be secured.

Moreover, IoT systems need efficient support for resource naming and discovery, which is not easy to deploy with IP in constrained infrastructures.

Since the IoT became a reality, IETF Working Groups have spent significant efforts to adapt the traditional TCP/IP stack to IoT systems (see Section 2). While IP adaptations for the IoT might be seen as cutting corners, the recently emerged Named Data Networking (NDN) architecture promises a better support of IoT systems and future Internet applications. NDN is an Information Centric Networking (ICN) protocol in which the first-class entity is the content. Networking operations are performed on content names, and hosts (without logical addresses) request named-content directly to the network. This principle brings native features such as a communication without establishing end-to-end connections nor name-to-address resolution. Moreover, no consumer-provider path or session needs to be maintained, providing a native support of multicast communication and connection disruption resulting from mobility.

However, it is quite challenging to take advantage of NDN features in current IoT solutions, without waiting for years the evolution of the global Internet architecture. The main reason is that fundamental modifications have to be brought to the current IP-based networking equipment and applications since the NDN paradigm operates on content names rather than host addresses. Furthermore, as long as IP solutions work for current applications, convince IP-enthusiasts and industrials on the benefits of NDN is still difficult. Fortunately, in recent years, many studies investigated the suitability of NDN for the IoT, making NDN more and more powerful [2, 5, 27]. With all this motivating work, real deployments of NDN can be envisioned. Although NDN is not ready for worldwide IoT deployments, we believe that real-world designs will enrich NDN experiments and help to figure out what is needed to make NDN a reality.

This paper describes a realistic deployment of an IoT architecture that enables NDN in constrained devices over low-power wireless technologies, commonly referred to as *low-end* IoT. Therefore, an NDN-IoT architecture is designed, deployed and evaluated. The motivation behind integrating NDN in low-end IoT is that low-end IoT is still in a development stage even with IP, which gives an opportunity to make NDN an important component of incoming IoT solutions.

We based our design on a realistic Smart Farming scenario. The concept of realistic in this context includes the following aspects. First, it aims to use NDN in the current Internet infrastructure. That is, scenarios that can not be deployed now are not considered. Second, realistic means using NDN to design solutions for current popular equipment such as prototyping boards. Third, the objective of a realistic solution is to provide a viable NDN-IoT solution that must be easy-to-use, low-cost, simple and lightweight.

The considered Smart Farming application is a cow monitoring system that uses sensors (e.g. movement, temperature, microphone.) to monitor health, fertility and activity of every cow individually. The devices are generally installed with a collar on cows. The data may be published from various places within the farm: inside, in the field, or in the milking parlor for example. The collected data can be visualized on the farmer's smartphone or stored/analyzed on its computer. The data may also be analyzed (e.g., with Machine Learning) to detect whether a cow is sick, or to forecast cows' activities such as heat periods to make accurate breeding decisions.

Our contributions can be summarized in two aspects. First, we show that NDN is more suitable than IP to support IoT applications. To achieve that, a detailed discussion on IP limitations and shortcomings is presented, followed by the design and deployment of a pragmatic NDN-IoT architecture that shows the simplicity of NDN compared to IP. The second contribution is a lightweight forwarding strategy that allows devices to retrieve content over the wireless network without a need for logical addresses nor route discovery.

The rest of this paper is organized as follows. Section 2 discusses the main IP efforts to support IoT challenges and their drawbacks. Section 3 gives an overview of NDN communication mechanisms in wired and wireless networks, and the main NDN entities: packets, data structures, etc. Section 4 discusses the possible deployment approaches of NDN with current networking equipment. Section 5 describes our architecture with its components and mechanisms. Section 6 provides details on our prototype design and implementation. Section 7 reports on multiple types of measurements and simulations conducted to evaluate our architecture, and Section 8 concludes this work with a discussion.

2 From IP to NDN

The emergence of the IoT puts IP networking to the test and highlights the mismatch between IP's host-centric paradigm and current application needs. Studying the IP-based solutions to support IoT applications tells much about the limitations of traditional networking. Although these solutions made IoT accessible to users, they resulted in extensions to TCP/IP protocols and the appearance of various other protocols acting like middleware between application, network and link layers.

For example, IPv6 uses a fixed length header of *40* bytes to improve packet processing speed, and assumes a minimum MTU of *1280* bytes to avoid fragmentation. This is reasonable for traditional networks, but the constraints of IoT are not considered at all. To cope up with this issue, 6LoWPAN [18] has been introduced as an adaptation layer between network and link layers to enable IPv6 networking over IEEE 802.15.4. This layer includes the mechanisms needed to support small MTUs such as header compression and packet fragmentation. Header compression is used for IPv6 and UDP headers to reduce their size in the majority of the transmitted packets, providing more space for application data. Packet fragmentation consists on splitting a standard IPv6 packet (i.e., 1280 bytes) into multiple link-layer frames (i.e., 127 bytes). Although these two operations enable IPv6 in low-power wireless networks such as IEEE 802.15.4, they bring additional overhead and processing, and consume more memory which is already rare in IoT devices.

For another example, most of IP solutions provide IoT features at the application layer using the REST (REpresentational State Transfer) architecture [19] similarly to Web services. To enable REST in IoT systems, the IETF CoRE WG defined the Constrained Application Protocol (CoAP) standard [29]. CoAP can be seen as a lighter version of the HTTP protocol. It is a data transfer protocol that provides a REST communication over UDP for constrained environments. Implementing such important features (i.e., CoAP) at application level indicates that the TCP/IP stack has reached its limit to support the new requirements.

Furthermore, requirements such as content discovery, secured caching, mobility and multicast communications make the task even more complicated for IP networking.

To sum up, IETF is doing considerable efforts to design protocols for constrained environments based on IP. The Constrained RESTful Environments (CoRE) group proposes CoAP to allow IoT devices exchange data as in the Web, and OSCORE for securing data objects at application level. The 6LoWPAN-WG is handling the adaptation of IPv6 over low-power low-rate networks such as IEEE 802.15.4. The ROLL WG is developing routing strategies and self-configurable mechanisms in low power networks and work closely with 6LoWPAN-WG. IETF has standardized RPL; a routing protocol for wireless constrained networks. The Light-Weight Implementation Guidance (LWIG) working group is helping to build minimal and interoperable IP-capable devices for constrained environments. The Thing-2-Thing Research Group (T2TRG) focuses on issues that may influence standardization processes in the IETF, to form the real IoT in which constrained devices can communicate with each other and with the global Internet.

Table 1: IP-based solutions for IoT vs. NDN features

IoT requirements	**IP-based efforts**	**NDN native features**
Resource naming	DNS, URI	Named content
Application data security	Object-based security	Self-secured packets
Request-response model	CoAP, REST	Consumer driven model
Small MTU	6LoWPAN	No fixed packet size
Caching	At application layer	In-network caching
Content dissemination/discovery	Multicast, CoRA-RD	Broadcast/multicast friendly

However, by looking at the solutions globally proposed to provide a viable IoT over TCP/IP, we observe that the most important functionalities are implemented in the application layer, using REST as a common architecture for communication. Therefore, the heart of the current IP-based IoT architecture is the application layer with REST instead of the network layer with IP as it is supposed to be. This happens because the host-based IP model can not support IoT requirements by itself; it needs a more rich and flexible architecture (e.g. REST) to provide caching, resource discovery and efficient security.

Even though current solutions (i.e., implemented in application-layer) may hide IP limitations to the final users of the IoT, the mismatch between the application-layer solutions and the actual TCP/IP architecture is a reality, in particular for the developers, and often causes performance degradation and interoperability issues.

One can easily imagine what will happen if we move the functionalities provided by the current IP-based IoT stack from the application layer to the network layer. In addition of being completely feasible, this can be more efficient, will reduce complexity in the application layer and greatly simplify application development. The obtained stack is then pretty close to the NDN architecture. To show that, Table 1 summarizes the main IP-based solutions for IoT compared to the main NDN native features that will be presented below. We observe that solutions which currently make IoT over IP feasible correspond exactly to the NDN features, except that they are provided by the core network in NDN.

3 Overview of Named Data Networking

Unlike the host-centric IP networking, NDN operates with URI-like names. To give a simple idea of NDN, we can imagine it as the HTTP's request-response model running at network layer. With the difference that NDN supports this communication model through packets carrying names as the main information, and all the networking operations (i.e., routing, forwarding, etc.) operate on names, not on binary network addresses.

Each NDN request is a packet called Interest, and can fetch *one* packet called Data. In the remaining if this document, "Interest" and "Data" starting with capital letter refer to the NDN Interest and Data packets respectively, a "consumer" is an application that issues Interests to request content while a "producer" is the application that creates and sends Data packets.

In NDN, every piece of content is identified by a unique name which applications use to request and retrieve data. Content names are independent from host location, which means that a content keeps the same name everywhere; at the content producer, caches and consumers. This feature is combined with self-secured contents to provide reusable packets and enable in-network caching, since a packet is independent from its source and destination hosts. By considering data names instead of host addresses, NDN retrieves content regardless of where it is located or how it is transported.

3.1 Naming and Packets

Another dissimilarity between NDN and IP is that NDN packets are encoded in the TLV (Type-Length-Value) format [31]. TLV encoding represents an NDN packet as a collection of sub-TLVs, without an explicit packet header. A TLV block consists on a sequence of bytes starting with a predefined number (*Type*), followed by its *Length* and its *Value*. Both Interest and Data contain a Name and may carry additional information according to the defined fields. Although Interest and Data packets have default and optional fields respectively (see Figure 1) , they do not have predefined packet size or field sizes.

A content is identified using a hierarchical name that contains a sequence of name components [24]. Each packet must contain a *Name* element. Name is represented by a 2-level nested TLV. The outer TLV indicates the complete Name element through the TLV-type (7). For example, the name *"/cowHealth/farm/cow/21/temp"* may identify the temperature value related to the cow with Id. 21 within the farm. With hierarchically structured names, the same data type related to another cow can be named *"/cowHealth/farm/cow/25/temp"*.

Figure 2 illustrates a TLV representation of an Interest. The Interest is identified by the type value: 0x05, the Name by 0x07, a Name-component by 0x08 and the Nonce by 0x0a.

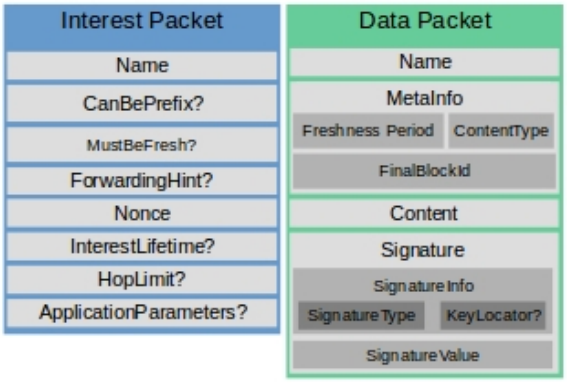

Figure 1: Interest and Data fields

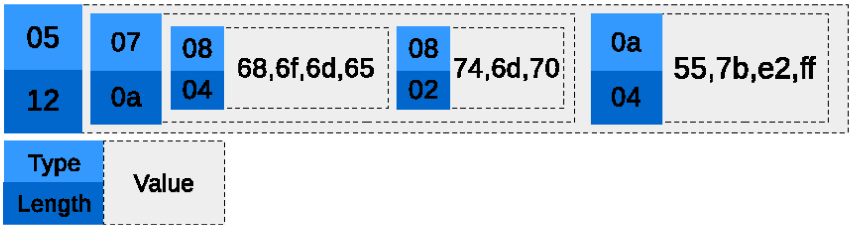

Figure 2: Interest TLV encoding example

3.2 Communication process

Each NDN node requires two data structures to process packets: Forwarding Information Base (FIB) and Pending Interest Table (PIT). Optionally, a Content Store (CS) can be used to store Data packets and provide in-network caching. The roles of these data structures are the following:

- The PIT maintains an entry for every forwarded Interest until its corresponding Data is received or until the entry lifetime is expired. A typical PIT entry contains the Interest, its incoming interface(s), the interface(s) to which it has been forwarded and a timer for Interest timeout. PIT entries are used to keep trace of Interests in order to forward the Data packet to the corresponding consumer(s). The PIT is also used to aggregate Interests requesting the same content to avoid redundancy.
- The native in-network caching is managed using the CS. After retrieving a Data packet, an NDN forwarder may store a copy of that packet in the CS before forwarding it to the next hop. Like any other caching data structure, CS is managed with caching placement and replacement policies such as LRU.
- The FIB contains information about the reachability of the content. A FIB entry associates a name-prefix to the interface(s) from which the content can be retrieved. The FIB is populated by routing protocols and is checked every time a node needs to forward an Interest.

The processing steps of Interest and Data packets at a node are depicted in Figure 3 and a typical NDN communication operates as follows:

1. The consumer application (e.g., on farmer's laptop) requests data by sending an Interest carrying the name of the data (e.g. */cowHealth/farm/area/1/cow/21/temp*).

2. Upon receiving an Interest, a forwarder first checks if matching Data already exists in its CS. If the corresponding Data is found, it is sent back as a response without forwarding the Interest any further. When no matching data is found in the CS, the forwarder checks the PIT if an Interest for the same content is already waiting; if so, the new Interest is not forwarded and only the originating interface is added to the existing PIT entry. The Interest is forwarded only if no corresponding Data is found in the CS and no similar Interest is already in the PIT. In this case, the Interest is forwarded according to the longest prefix match (LPM) against the FIB entries. For example, if the possible matching in the FIB are */cowHealth*, */cowHealth/farm* and */cowHealth/farm/area/1*, the longest one is chosen. After that, the forwarder creates a PIT entry for the Interest and forwards it through the corresponding interface. If no matching is found, either the Interest is flooded to all outgoing interfaces or deleted, according to the forwarding strategy.

3. When the Interest reaches the content producer (e.g., the sensor) or an intermediate cache node, the Data packet containing the content is sent back. The Data packet follows the reverse path of the Interest; following the matching PIT entry on every forwarder. That is, all interested consumers (i.e., which issued an Interest) will receive a copy of that Data. After forwarding a Data, a forwarder discards the PIT entry and stores the recent Data packet in its CS. If a node receives a Data packet without a matching entry in the PIT, the Data is considered unsolicited and is dropped.

In wireless networks the packet processing is the same, but the forwarding decision may be slightly different than in wired networks. The main reason is that a wireless radio corresponds to only one network interface; thus, a forwarder cannot distinguish between different next-hops using network interfaces. Using addresses (e.g., MAC) to forward packets reduces the data dissemination potential of NDN and limits its benefits in mobile wireless networks. Moreover, mapping names to addresses requires transmission overhead to discover and maintain routes, and consumes more memory to maintain the FIB. That is, using broadcast communications is the most simple and efficient forwarding mechanism in NDN wireless networks.

In the following, we introduce Controlled Flooding (CF) , a simple broadcast-based forwarding strategy for NDN over wireless networks. To keep the benefits of flood-

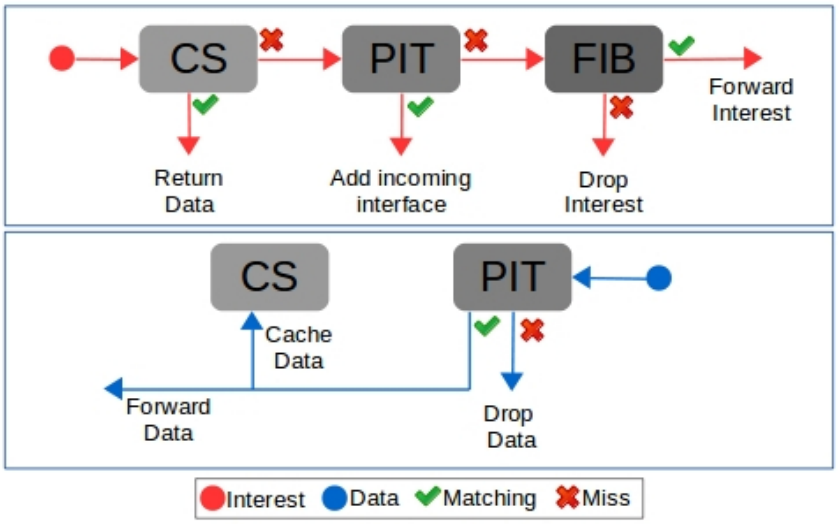

Figure 3: Interest and Data processing inside a node

ing/broadcast while reducing overhead and redundancy, CF nodes exploit broadcast to overhear communications and possibly avoid forwarding some packets. To do so, every (potential) forwarder defers its packet transmission with a random delay during which it keeps listening on the shared wireless medium. While waiting, if the forwarder overhears a packet (i.e., Interest or Data) with the same name, it cancels its transmission. In practice, Interest and Data transmissions are deferred for Δ_I and Δ_D periods of time respectively. Both Δ_I and Δ_D are computed based on an interval, defer window (dw), from which an integer value is randomly chosen to generate the waiting delays as follows [3]:

$$\Delta_D = rand[0, dw] \times DeferSlotTime \tag{1}$$

$$\Delta_I = (dw + rand[0, dw]) \times DeferSlotTime \tag{2}$$

where $DeferSlotTime$ is a short period of time.

Here, Δ_I and Δ_D are selected in disjoint intervals with $\Delta_I > \Delta_D$ to give higher priority to Data packet transmissions and avoid useless Interest broadcasts. During the Δ_I waiting time, a potential forwarder listens to the channel: if it overhears the same Interest or the requested Data, it cancels its own transmission.

4 NDN Deployment: Possible Approaches

Conceptually, a clean-slate NDN deployment requires network entities to support routing and packet processing based on names, and implement some forwarding strategies and security procedures. In addition, more storage is needed for caching and stateful forwarding [15].

More generally, NDN can be deployed as an overlay on top of IP, can replace IP as a native network protocol over the link layer (e.g., NDN over Ethernet), or IP and NDN can coexist in the same network. The first approach, the overlay, is easy to deploy and creates a uniform content-centric layer. It is realistic and the NDN testbed [23] is an example of such approach. However, this solution creates complexity and overhead for the underlying network protocol, and IP-based applications must switch to NDN in order to use the network. Moreover, the overlay approach considers NDN as a transport/application protocol for IP, and thus does not provide a coexistence between the two network protocols (i.e.,

IP and NDN). More importantly, implementing both NDN and IP stacks is not feasible with IoT constrained devices that can barely support the current IP stack. The second approach, deploying NDN as a native network protocol, works only for environments that do not need to communicate with global IP networks, such as isolated vehicular networks or local networks. Hence, we consider it as unrealistic. The third and last approach is to make IP and NDN coexist within the global network. This approach may either use NDN at the core and keep IP at the edge of the network (NDN-core), or deploy NDN at the edge and keep IP networking at the core (NDN-edge).

With NDN-core, IP applications do not need to be changed at all, but a global deployment of NDN as a native network protocol is currently not feasible as mentioned before. As an exceptional example, the POINT project [33] had to work with ISPs to deploy a real-world prototype in which an ICN architecture is used at the core of the network. The prototype then introduces ICN in the core network without changing the rest (i.e., the edge) of the Internet.

With NDN-edge, the core network keeps running IP, while applications and devices run native NDN. This solution is easy to deploy and does not required deep changes in the infrastructure. Moreover, it provides a progressive integration of NDN.

In both NDN-core and NDN-edge, the coexistence of IP and NDN can be achieved by using peripheral nodes such as gateways to translate between NDN names and IP protocol stack information. For example, Cisco's hICN [20] encodes names as IPv6 addresses to allow hICN packets to be processed by both ICN-based and IP-based routers, and Zhang et. al. [32] proposed a dual-stack scheme for NDN switches and IP switches to coexist in local area networks.

At application level, when NDN and IP stacks have to coexist together, NDN and IP applications require completely different mechanisms. As the purpose is to take the most of NDN while providing feasible solutions, we report on two possible translation approaches between NDN and IP as described in [15]:

- The first solution is to provide a translation between TCP/IP or UDP/IP and NDN. The advantage of this approach is the support of various application protocols with the same transport-level translation. However, as network and transport layers in the IP stack have limited expressiveness, some NDN features will not be exploited. For example, translating a TCP packet into an NDN packet may use information from TCP/IP headers to generate NDN names. That is, the name will be associated to the specific TCP connection. This provides benefits such as caching within the same TCP connection (e.g., efficient re-transmission of lost packets), but cannot support caching across different TCP connections (e.g., multicast to different consumers). Moreover, data-centric security of NDN will be limited as names are still related to hosts and connections.
- The other approach consists on translating between application-level protocols such as HTTP to NDN. Application-level information is much more expressive and data-oriented than network and transport information. Thus, NDN names generated from HTTP headers will be more meaningful regardless of hosts and connections. This allows this approach to take much more NDN benefits than the previous one.

Furthermore, to avoid translation, a hybrid deployment can be adopted combining NDN-edge with NDN-overlay approaches to achieve the maximum possible integration of NDN. This combination is realistic since devices at network edge implement only NDN and equipment at core network have enough resources to support NDN over IP with an additional overhead. The NDN-IoT architecture we propose is based on such hybrid solution and is discussed below.

Integration approaches discussed above are summarized in Figure 4.

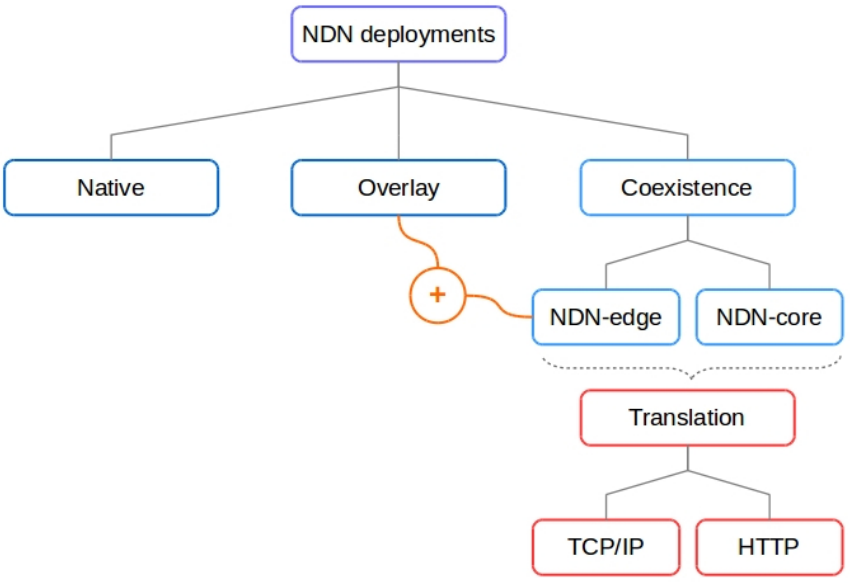

Figure 4: NDN integration approaches

5 Proposed NDN-IoT Deployment

This section describes the realistic NDN architecture we envision for the IoT Smart Farming application. After a study of the integration possibilities, we chose the NDN-edge approach combined with an NDN over UDP/IP in the core network. Our motivation is explained below, followed by an overview of the wireless technologies applicable in the IoT and our choice. The architecture, its components and mechanisms are described afterwards.

5.1 Deployment Approach

When applied to the IoT, the NDN-edge integration corresponds to the deployment of NDN in low-end IoT. In other words, NDN is used where the content is produced and/or consumed. On the one side, IoT devices run native NDN applications over a wireless link-layer technology. On the other side, using NDN over IP-based transport protocols such as UDP allows applications on computers and smartphones to communicate with IoT devices via NDN.

In addition to be completely feasible in the current Internet infrastructure, this approach takes advantage of all NDN features such as naming content, data-centric security, and caching. Moreover, as IoT design is at its early stage, particularly at low-end IoT, this approach is a reasonable starting point to create NDN-capable devices together with NDN native applications without waste of time. In addition, integrating NDN from the edge of the network supports a progressive and incremental integration. Experience learned from local deployments will lead to a stronger NDN architecture and various possibilities can be envisioned for the long term.

Furthermore, most IoT applications rely on Internet to reach cloud servers. However, there are cases when Internet connectivity is not available but local network connectivity exists, such as in some smart agriculture scenarios. Typically, IP applications stop working without global connectivity, but applications in NDN deployments would be able to work as long as connectivity exists. They can discover names and exchange data between two locally connected devices without going through the Cloud.

5.2 Wireless Technology

The IEEE 802 Standard is a set of networking standards for both wired and wireless networks. The most known wireless specifications include 802.11 [13] (e.g., WiFi), 802.15.4 [12] (e.g., ZigBee) and 802.15.1 [11] (e.g., Bluetooth).

Although communication solutions for IoT do not generally require a large bandwidth, they need an efficient power management plan, a low cost of production and must support a large number of nodes in a simple way. That is, IoT wireless technologies are typically focused on low power consumption, large number of nodes and (relatively) long range communication. Many physical and link-layer specifications provide such a compromise. For example, Bluetooth Low Energy (BLE) and IEEE 802.15.4 are designed for wireless personal area networks (WPANs) and allow satisfactory data rate with low-power consumption and reasonable complexity. Other communication specifications are available for specialized networks such as WAVE [14] for VANETs and 3G/4G for very long distances. More recently, new wireless technologies explicitly designed for IoT have appeared such as Sigfox [30] and LoRa [16]. However, these recent technologies still expensive for customers in comparison to IEEE 802.15.4 and BLE. Among these wireless technologies, Bluetooth Low Energy (BLE) and IEEE 802.15.4 appear to offer a satisfactory compromise between range, power consumption and cost. Both are considered as low-power low-rate technologies and are currently dominating in IoT systems. BLE and IEEE 802.15.4 operate in the 2.4 GHz ISM spectrum, but have their own modulation scheme, bit rate, channel map and channel spacing, and upper layers [21]. We chose to consider the IEEE 802.15.4 for its stack and configuration simplicity, and for the flexibility it provides in terms of network topology.

5.3 Architecture

The first benefit of the NDN integration in low-end IoT is to make end-devices an integral part of the NDN network, whether they are producers or consumers. In addition to send Interest and Data packets over IEEE 802.15.4 frames, an efficient integration should consider packet transformation at the intersection between the wireless local network and the backbone, to accommodate constrained devices.

As we target a realistic deployment, our architecture follows common IoT solutions as despicted in Figure 5. The gateway and the end-devices communicate with NDN over IEEE 802.15.4, while the gateway and applications communicate with NDN over UDP/IP. We assume that each end-device is a source of data and thus a content producer. The gateway forwards Interests issued from user applications, whereupon end-devices can reply with Data packets that they produced or stored in their caches. Each end-device generates data under a specific prefix-name obtained altogether with required security materials (e.g., signature keys, certificates) through a pairing process. Each end-device uses a different content name, but a common prefix is shared between end-devices and the gateway within the local network. A Wireless Local Area Network (WLAN) is formed by a gateway and a set of end-devices. Each WLAN is accessible via the gateway under a Common Prefix (CP). In summary, architecture components are the following:

1. End-devices (EDs): Wireless nodes running NDN protocol and communicating through IEEE 802.15.4 transceivers.

2. Gateway: Physical node that allows NDN applications to reach EDs from the Interent.

3. Local Manager (LM): Its role is to manage device identities, access control, etc. Thus, it handles a pairing process for EDs, an authentication server, etc. LM is a software component typically included in the gateway.

4. WLAN: A gateway, including the LM and a set of EDs form a wireless local network accessible via a common prefix (CP).

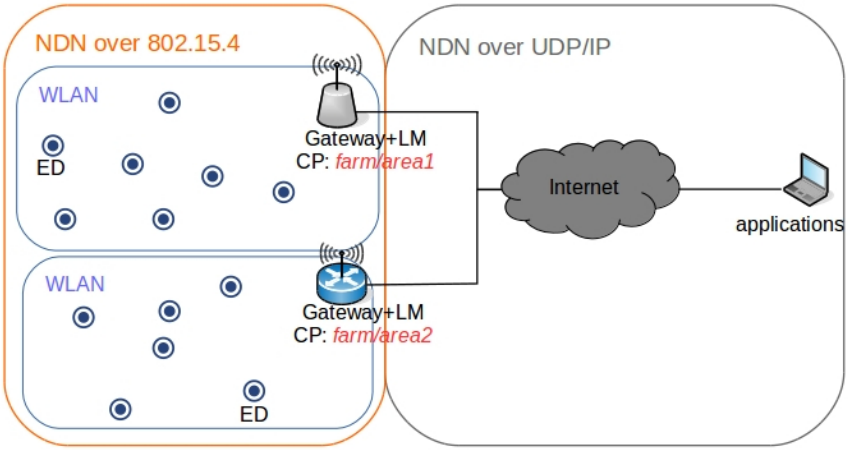

Figure 5: NDN-802.15.4 architecture

Figure 6 depicts the NDN protocol stack with IEEE 802.15.4 integration, a simplified OSI model and the 6LoWPAN stack.

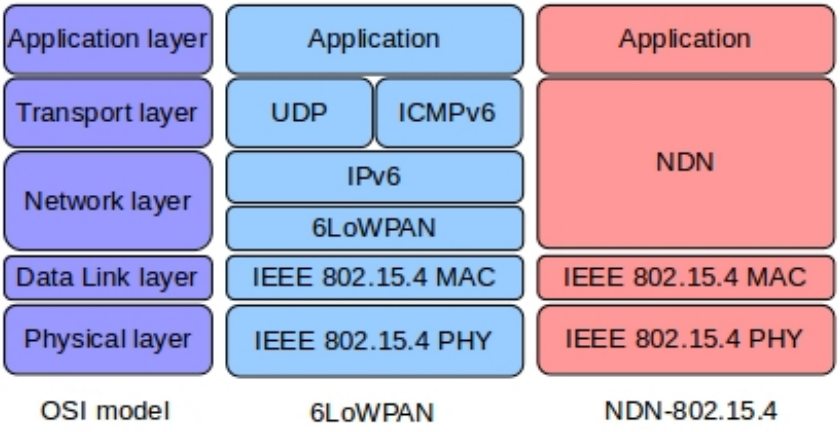

Figure 6: NDN-802.15.4, OSI model and 6LoWPAN stack

To provide a better support of NDN over IEEE 802.15.4, we exploit the flexibility of NDN to fit better with IoT devices constraints such as small MTU, limited memory and CPU. The proposed operations are implemented in the EDs and the gateway and described below.

5.4 Stateless Packet Compression

Interest packets issued by applications can rapidly become large, due to long content names and additional fields. The corresponding Data will be even larger due to content and signatures. However, we observe that requested content can be retrieved from EDs without sending all fields of an Interest. Missing fields can be computed based on predefined configurations, packets specification and previously shared information. Similarly, Data packets produced by EDs can be sent with missing fields that will be completed by the

Table 2: Packet fields classification

Class	Fields	Treatment
STATIC	(part of) *Name*, *NameComponent*	Not transmitted
INFERRED	*SignatureType*, *KeyLocator*	Not transmitted
DEFAULT_VALUE	*ContentType*, *FreshnessPeriod*, *HopLimit* *InterestLifetime*, *MustBeFresh*, *CanBePrefix*	Not transmitted when default value
VARIABLE	*Nonce*, *Content*, *SignatureValue*, *Parameters*	Always transmitted
UNSUPPORTED	Rest of the fields (e.g. *ForwardingHint*).	Not transmitted

gateway. To exploit this feature, we designed a stateless packet compression scheme to reduce overhead in the WLAN and avoid packet fragmentation.

The reasons of choosing a stateless compression are multiple. First, as the information compressed in NDN packets mainly consists on names and characters, even a simple compression scheme will significantly reduce packet sizes. That is, a context is not required to achieve an efficient compression ratio. Second, stateless compression does not require additional memory and overhead to operate and is simple to implement in constrained devices. Third, from an implementation point of view, a stateless compression process consists on omitting/updating certain bytes when sending the packet. Thus, the sender does not need to maintain the two states of the packet (i.e. compressed and decompressed) like in IP. Similarly, the receiver decompresses the packet by adding/updating certain bytes.

Based on the same principle as for IP header compression, we adopt a packet fields classification according to which the packets are compressed. The field classification is summarized in Table 2 and the description of the classes is the following:

- Static: fields shared by EDs and the gateway/LM in the local wireless network. For example, CP is a part of content Name shared by all WLAN nodes. Such fields are never sent between the gateway and EDs, either in Interest or in Data packets.
- Inferred: fields that are not exactly the same for the gateway/LM and all the EDs, but they can be calculated using WLAN shared configuration and conventions (e.g. trust conventions). For instance, we assume that common information has been shared between the gateway and EDs through a pairing process. When such information exists, related fields are not transmitted.
- Default value: fields with a default value defined in the NDN specification. That is, these fields are not transmitted when they have the default value, but are transmitted otherwise. For example, *ContentType* in a Data packet is not transmitted when the packet contains application data.
- Variable: fields that can not be inferred and are not common to WLAN entities. Thus, they must always be transmitted.
- Unsupported: fields that EDs and/or the WLAN do not support because their processing is too complex for constrained devices. They are never transmitted. This class is intended to support packet fields restrictions for the wireless forwarding mechanisms. When no explicit restriction is defined, these fields are transmitted.

5.5 Lightweight Wireless Forwarding Strategy

5.5.1 Design Guidelines

Given the limited resources and capacity offered by EDs, we consider the following guidelines and requirements for a lightweight forwarding strategy:

1. Rely on a minimal state to process packets without maintaining explicit routes.

2. Avoid reverting to a random Interest flooding phase in order to provide accurate forwarding decisions while reducing collision risks, network congestion and overhead.
3. Avoid node identification or addressing and use only broadcast communications to correctly fit the NDN vision.
4. Avoid additional data structures to preserve the lightweight aspect of the NDN stack and allow more space to caching.
5. Distribute decisions and computation tasks over the network, and minimize complexity.

Based on these guidelines designe, we designed a forwarding strategy, called Reinforcement-based Lightweight Forwarding (R-LF) strategy and detailed below.

5.5.2 Approach and Assumptions

Routing and forwarding operations are significantly different in NDN and IP. In IP, only the routing operation is smart in the sense that different routing protocols can be envisioned. The forwarding operation always consists in finding the longest match available in the routing table and sending the packet to the corresponding next hop. In NDN however, in addition to the routing operation that can be smart as in IP, multiple approaches are possible to handle packet forwarding with more or less additional information and with or without caching.

Our proposed strategy does not use an explicit routing phase to gather or update forwarding information. R-LF operates according to the following steps: (i) the nodes overhear Data packets and learn a cost value by reinforcement, (ii) the nodes decide to forward an Interest with a delay according to their cost-based eligibility, (iii) the nodes update their cost from the result, which can be an Interest timeout or a received Data packet.

The following describes the R-LF approach followed with the mathematical formalization.

5.5.3 General Description

To forward packets, a node traditionally decides in terms of what the next hop is, which can be considered as a spatial forwarding decision. However, as NDN forwarding is based on content names and R-LF uses only broadcast directly on top on the MAC layer, a node decides in terms of when it should forward the Interest; i.e., how long it should wait before forwarding. Such a process can be seen as a temporal forwarding decision. This approach has been explored before and consists in ensuring that the more eligible node will forward the Interest first [17].

To describe the forwarding approach, the following assumptions are made:

- Interest and Data packets carry the cost value of the sending node, denoted as *C-field*.
- Interest flooding with random delays similar to CF are used when the first Interest is issued for an unknown content.
- Nodes are able to overhear Interest and Data packets related to other communications.

R-LF operates according to two phases: (i) a reinforcement learning that consists in maintaining a cost value for each available content prefix, (ii) an adjustment of the waiting delay based on the neighborhood activity.

The first learning phase starts after a source node receives a randomly flooded Interest, and acts as follows (see Figure 7 steps 1 and 2): (i) the source node responds with a Data packet carrying the initial cost, (ii) the first forwarder on the source-consumer path computes its cost with a reinforcement technique, replaces the C-field with the value

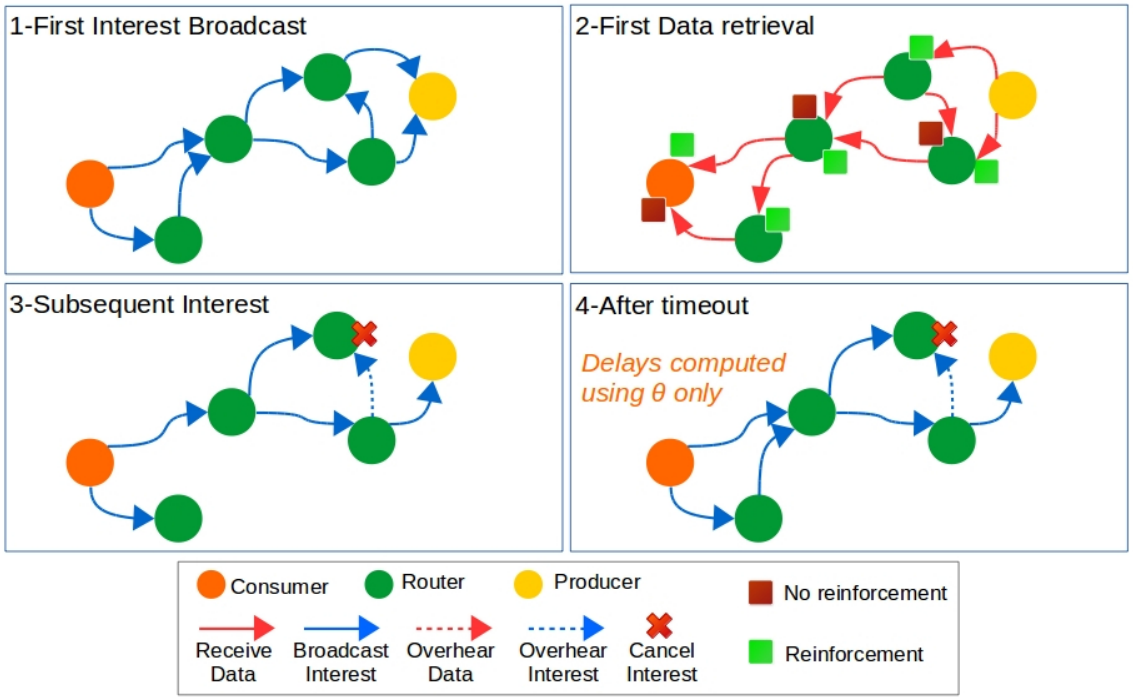

Figure 7: Common forwarding situations with R-LF

obtained, and forwards back the Data packet, (iii) each node on the path follows the same procedure until the Data packet reaches the consumer, (iv) in the vicinity of the path, the nodes that overhear the Data packet can perform a passive cost update to learn their eligibility relative to the data source.

The random flooding phase is then over, and the first learning phase is set up. Each node updates the cost related to the corresponding prefix-name after retrieving (or overhearing) a Data packet with a smaller cost. Let us refer to this phase as the *reinforcement* phase.

To describe the forwarding process, we define the delay to wait before forwarding an Interest as $\Phi(a)$. The formal definition of $\Phi(a)$ will be detailed later. The forwarding decision in a relay node consists in finding the appropriate value of a that gives a correct delay to wait. Since a is calculated in two steps, let $a = \Delta + \theta$, with Δ and θ as explained in the following.

Let $C_x(p)$ and $C_y(p)$ be the current cost for prefix p at nodes x and y respectively. Whenever node x receives an Interest issued (or forwarded) by node y, it computes the value $\Delta = C_y(p) - C_x(p)$. Here, Δ quantifies the global eligibility of node x to forward the Interest. If $\Delta \geq 0$ then node x can potentially forward the Interest.

The value of θ is locally computed by the forwarder based on its neighborhood activity to refine Δ before calculating the delay time. Let us refer to this phase as the *Delta adjustment* phase.

After computing a, the Interest forwarding is delayed for $\Phi(a)$ units of time. During the delay-listening time, if node x detects that a forwarder z is transmitting a packet with the same name, it deduces that z is more eligible to handle the Interest and cancels its pending transmission (see Figure 7 step 3).

With the delta adjustment, the random-delayed forwarding is used only when the prefix is unknown by the forwarder (x) and is reset by the sender (y). Thus, even after an Interest timeout in x and y, the value of θ can be used in most cases to distinguish nodes eligibility (see Figure 7 step 4).

The next subsection provides the mathematical details.

5.5.4 Details and Mathematical Formalism

Reinforcement phase: The cost value at node x is updated according to Equation 3:

$$C_x(p) = (1-\alpha)C_x(p) + \alpha\,(r + \min C_y(p)) \tag{3}$$

In this equation used in Q-learning and Q-routing [6], α is the learning rate, r is the reward, $C_x(p)$ is the cost at node x for the prefix p, and $\min C_y(p)$ is the smallest cost heard by node x from node y.

Assuming the hop-count as a metric, the reward is always equal to *1*, and the cost of each node increases as the distance to the content source increases. The cost at the content source is *0*.

The cost values reflect the distance to the content source and are used to decide on a forwarder's eligibility. Moreover, the approximate nature of the update formula produces a large number of possible cost values over the network, which helps to avoid obtaining the same waiting delay for different nodes.

Since the nodes remember only the smallest heard cost, a node may have an obsolete estimation of its cost value. To avoid that, after an Interest timeout, a node resets its cost value (i.e. $C_x(p)$) to *0* and the smallest heard cost (i.e. $\min C_y(p)$) to the maximum cost $\hat{\Delta}$, in order to accept cost updates. Note that in Interest packets, a cost value of *0* indicates that the sender has reset its cost or it has no information about the content prefix. Thus, it does not interfere with the *0* cost value of a Data packet which actually means that the packet has been sent by the producer. The estimation of $\hat{\Delta}$ is presented further.

When caching is enabled, the cost carried in a cached Data packet may introduce uncertainty in the reinforcement calculus, especially when a relay node has cached only few chunks of the requested content. To overcome this, when a cached Data is returned by a relay node, the cost carried in that Data is the highest expected value (i.e., $\hat{\Delta}$). This way, cached Data packets do not lead to a reinforcement update at other nodes, since cached Data packets may not carry accurate cost information.

Delta adjustment: The adjustment serves two purposes: it refines Δ to deal with local uncertainty in real time and allows each node to handle multiple content prefixes simultaneously. In fact, using only Δ to compute delays, even if it is accurate, does not allow different content names cohabitation to be supported.

To compute θ, let N_a be the neighborhood activity rate for all data names. From the perspective of a node, N_a can be computed by $N_a = D_u/I_d$, where D_u is the number of unsolicited received Data and I_d is the number of non-forwarded Interests (dropped Interests).

Then, θ may be simply defined as

$$\theta = Th - N_a \tag{4}$$

where $Th \leq 1$ is the activity threshold above which the waiting time should be increased.

For simplicity, but without losing accuracy, N_a is kept between *0* and *1*. Thus, if Th is lower than *1* (e.g. *75%*), θ can be negative. In this case, the value of Δ is reduced, which will increase the waiting time. When no statistic is available, $N_a = Th$.

Delay function: After defining the appropriate value of a, the delay time is computed with a function that is inversely proportional to the value of a. Such a function can be intuitively defined by:

$$\Phi(a) = \frac{M}{e^{a/2}} + m \tag{5}$$

This function ensures that when two nodes can both forward an Interest, the node with the highest value of a will delay its transmission for a shorter time than the node with

the lowest value, as depicted in Figure 8. In addition, m forces the forwarder to wait for a minimum time to let the transmission of the corresponding Data packet if any, while M controls the upper-bound of the calculated delays.

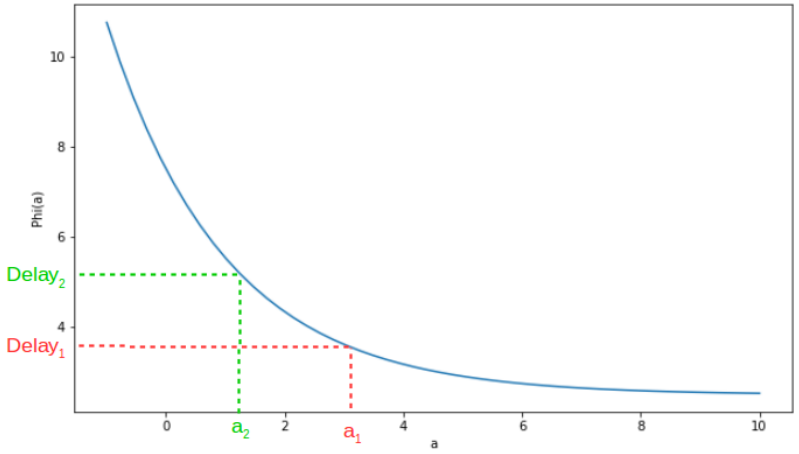

Figure 8: Delay function example

The importance of Φ is capital and a parameter calibration is needed to have an efficient distribution of waiting times.

We can observe that $\lim_{a\to\infty} \Phi(a) = m$. Therefore, we need to ensure that $\Phi(\hat{a}) > m$, where $\hat{a}$ is an estimation of the highest value of a.

According to Equation 4, $\theta \leq Th$. Given that the lowest cost value that can be computed by a node is close to *0*, we can deduce the highest gap between two cost values as being the highest cost value in the network.

To estimate the highest reinforcement value expected in a network, we use the following Q-learning update property:

$$C_x(p) \leq \epsilon + C_y(p) \Rightarrow C'_x(p) \leq \epsilon + C_y(p) \tag{6}$$

where $C'_x(p)$ is the updated value of $C_x(p)$ and $C_y(p)$ is the min overheard cost.

This property is proven in [26] using the Q-learning update properties and initial conditions.

Given that NDN packets do not loop by design and considering a grid topology of n nodes, we assume that the average distance between two nodes should not exceed $\sqrt{n}$ hops. Then, we use Equation 6 to recursively estimate the maximum expected value of Δ as $\hat{\Delta} = (\sqrt{n} + 1)$. Then, we deduce an estimation of $\hat{a}$ to set the parameters of $\Phi(a)$.

The forwarding decision process for a node x with a cost of $C_x(p)$, receiving an Interest from node y with a cost of $C_y(p)$ for a prefix p is summarized in Algorithm 1.

6 Prototype Design and Implementation

The prototype design considers current IoT equipment available for common users. First, Arduino single-board microcontrollers are a typical example of constrained devices to create EDs; with a low-power, slow-speed CPU and a few kilobytes of RAM and Flash. When deploying IoT applications in environments such as agricultural fields, it is common to use sensors and actuators running on such constrained equipment [4]. These devices are intended to support the NDN stack and run a producer application that basically creates, names and signs Data packets, and processes Interests received from the gateway. Second,

Function ProcessInterest
Data: Interest packet
begin
  if *p is unknown* **then**
    if $C_y(p) == 0$ **then**
      Broadcast with random delay
    else
      Drop Interest (node not eligible)
    end
  else
    if $C_y(p) == 0$ **then**
      $\Delta = \hat{\Delta} - C_x$
    else
      $\Delta = C_y - C_x$
    end
    if $\Delta \geq 0$ **then**
      $\theta = Th - N_a$
      $a = \Delta + \theta$
      Broadcast with $\Phi(a)$ delay
    else
      Drop Interest (node not eligible)
    end
  end
end

Algorithm 1: Interest forwarding process

the gateway has typically more CPU and memory resources than EDs. It also has more power as it is commonly plugged to a constant source of energy. The class of equipment we use as gateways is represented by Raspberry-Pi single-board computers. Raspberry-Pi hardware is widely used for prototyping and making IoT low-cost applications for testing and developing Proofs-of-Concepts and embedded systems. The wireless communications are achieved using XBee radio modules [25].

6.1 Gateway

The gateway includes the LM and supports NDN communications over IEEE 802.15.4 with a process (called NDN-15.4) that we design to work next to the NDN forwarding module (i.e., NFD [1]). Figure 9 depicts the software and hardware components involved in the gateway after integrating NDN over IEEE 802.15.4.

The NDN-15.4 process manages sending Interests and getting Data, and receiving Interests and sending Data. That is, the gateway is able either to receive Interests from EDs, forward them to the backbone and get the Data back, or to forward Interests from the backbone to EDs and get Data packets back. As the forwarding strategy is based on broadcast only, the NDN-15.4 process uses the broadcast address to send packets. In practice, the NDN-15.4 process intercepts Interests with a certain prefix p (e.g., */cowHealth/farm*) and sends them over the wireless link. This prefix has to be set in order to forward all Interests with names starting with p to the IEEE 802.15.4 WLAN. To achieve that, the NDN-15.4 process registers the prefix p to NFD, which creates a FIB entry to bind p to the NDN-15.4 process. Then, NFD acts in a normal way; upon receiving an Interest with a name-prefix p, it forwards it to the NDN-15.4 process. When the corresponding Data comes back to the gateway, it is forwarded to the appropriate application or next

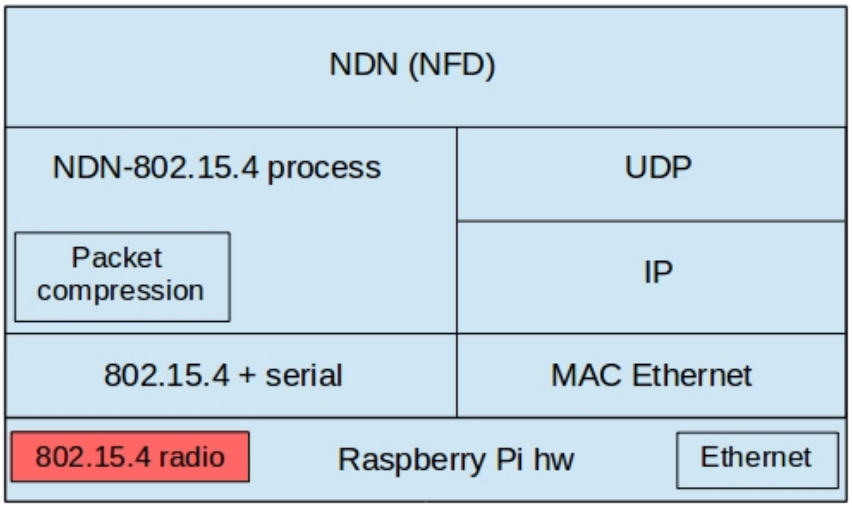

Figure 9: Architecture of the gateway

Table 3: Typical FIB at the gateway

Prefix	Face	Cost
/cowHealth/farm	NDN-15.4 process	0 (local)

hop according to the PIT. Table 3 gives the typical FIB at the gateway, and Figure 10 represents the NDN-15.4 process in pseudo-code.

The stateless packet compression scheme described before is implemented in the NDN-15.4 process. The NDN-15.4 process is implemented in Python using the PyNDN2 library [10]. The library that handles IEEE 802.15.4 frames is also implemented in Python.

6.2 End-devices

A lightweight version of the NDN protocol stack is available on Arduino thanks to the ndn-cpp Lite library [22]. This library supports encoding and decoding TLV packets, and includes cryptographic algorithms such as HMAC and ECSDA. We extended ndn-cpp Lite with a simple IEEE 802.15.4 communication library that handles XBee modules to send and receive NDN packets over IEEE 802.15.4 frames, in the same way as the gateway does. Figure 11 depicts the architecture of an ED including software modules and libraries. Figure 12 shows an actual picture of an ED prototype.

6.3 Consumer Application

To collect data, an NDN consumer application runs on a computer or laptop and periodically issues Interests which are forwarded by the gateway to the WLAN. Depending on the purpose of the monitoring, the application can display data on a map, on a dashboard, or simply store it. Applications are developed in Python using the NDN library PyNDN2.

7 Deployment and Evaluation

The cow health monitoring system deployment consists on one WLAN with one gateway and four EDs. A consumer application runs on a laptop connected to the gateway on the local network (LAN) and periodically sends Interests to collect data.

```
/* callback for incoming Interest from the backbone */
function onInterest(interest) :
    frameBuffer = encodeAndCompress(interest)
    ieee802154.broadcast(frameBuffer)

/* callback for incoming Data from the backbone */
function onData(data) :
    frameBuffer = encodeAndCompress(data)
    ieee802154.broadcast(frameBuffer)

function main() :
    /* to receive Interests from the backbone */
    backboneFace = Face()
    prefix = "/farm"
    backboneFace.registerPrefix(prefix, onInterest)

    /* connect to the 802.15.4 radio */
    ieee802154 = Ieee802154()

    while True:
        /* process incoming packets from the backbone */
        backboneFace.processEvents()

        /* process incoming packets from the WLAN */
            frame = ieee802154.wait_read_frame(0.01)
            if frame :
                    if frame.isData() :
                        data = decodeAndDecompress(frame)
                        backboneFace.put(data)
                    else if frame.isInterest() :
                        interest = decodeAndDecompress(frame)
                        backboneFace.issue(interest)
```

Figure 10: NDN-15.4 process

The WLAN is identified by the common prefix *CP = /cowHelath/farm/area/1*. Each ED serves content under a name */cowHelath/farm/area/1/cow/<cowID>/temp*, formed by the CP, a cow ID, and data type which is temperature in our case. Here, <cowID> is a 1-byte number that identifies each cow.

A 32-byte HMAC signature is used to secure Data packets. The secret keys are directly hard-coded in EDs and the monitoring application since key management is beyond the scope of this work. However, the security support descibed in [34] can be deployed as well.

An Interest packet issued by the consumer (i.e., uncompressed) carries the Name, a Nonce, a MustBeFresh indicator to accept only fresh data, and has a lifetime with the default (i.e., 4 seconds).

In the uncompressed Data packet created by the ED, the Name contains an additional component that represents the timestamp of the data collection. The Data name obtained is then */cowHelath/farm/area/1/cow/<cowID>/temp/<timestamp>*. Each Data packet has a 4-byte content that contains the measured temperature. In MetaInfo field, a 1-byte ContentType indicates that the packet carries raw content, followed by a 2-byte Fresh-

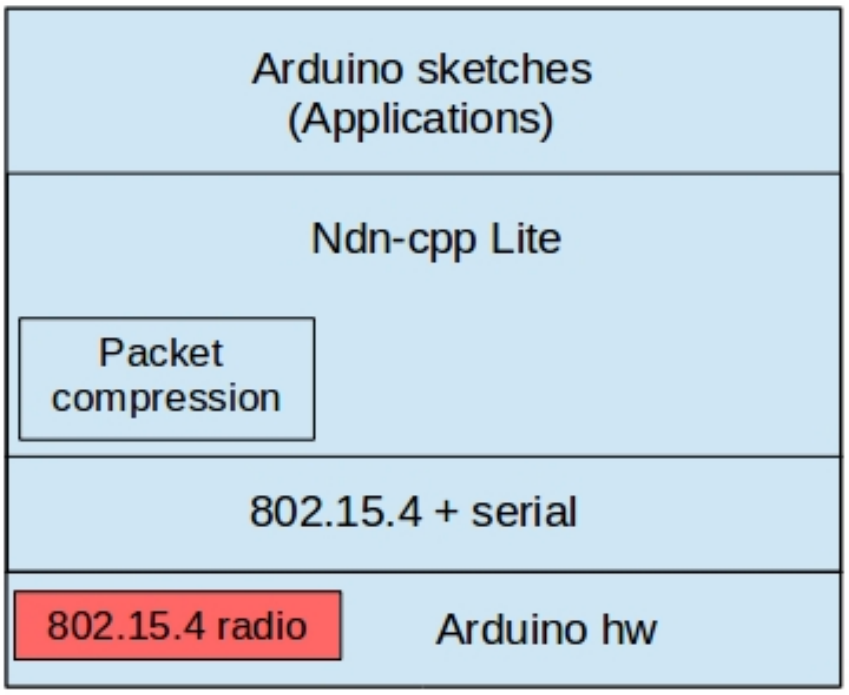

Figure 11: Architecture of the ED

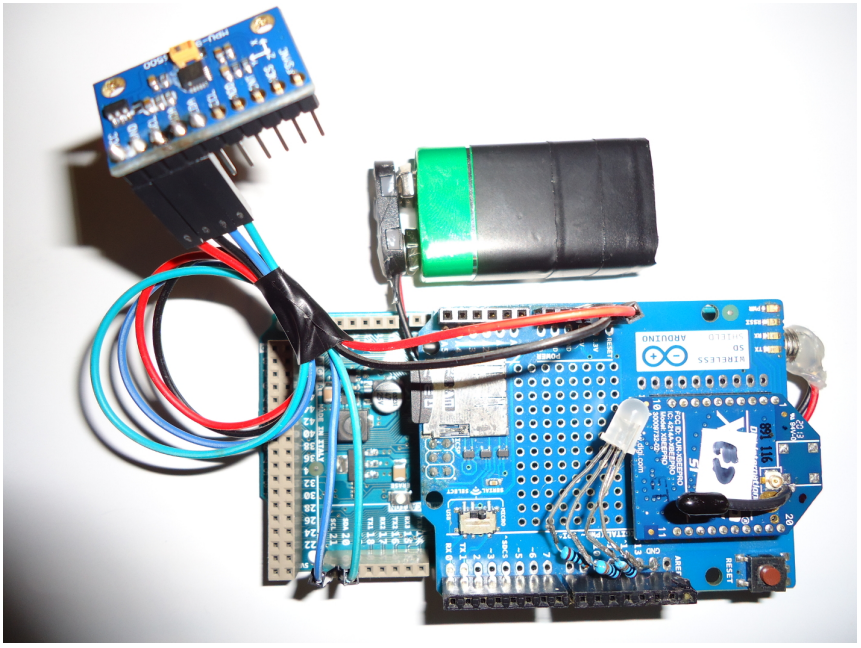

Figure 12: Picture of an ED

nessPeriod field. Finally, the Signature TLV component contains a 1-byte SignatureType and a 16-byte KeyLocator.

A typical Interest-Data exchange operates as follows. The Interest issued by the application is forwarded to the gateway over UDP/IP. Then, the gateway compresses it and sends it over the IEEE 802.15.4 link. After the wireless forwarding process, the corresponding ED responds to the Interest by a signed Data with the corresponding content. Finally, the gateway decompresses the Data packet and forwards it toward the consumer application.

Table 4: NDN-802.15.4 and 6LoWPAN features comparison

Feature	**NDN-802.15.4**	**6LoWPAN**
Fragmentation	Yes	Yes
Packet structure	Flexible packet format	Fixed packet format
Compression	Stateless packet compression	Stateful header compression
Mobility	Simple adaptations	Additional protocols NEMO, AdapterMIPv6, etc.
Security	Native data-centric security	MAC and TLS security

Four types of evaluation are reported in the following: (i) a features comparison between our NDN-IoT architecture and 6LoWPAN, with a discussion on the feasibility of NDN-IoT in terms of implementation and security, (ii) a theoretical evaluation of the gain achieved with the packet compression scheme, (iii) results on code size and communication delays measured in our deployment, (iv) a simulation evaluation of the R-LF forwarding strategy compared to CF in a local-scale deployment.

7.1 Features and Feasibility

The proposed packet compression does not always require decompression. Indeed, the gateway and EDs perform different (de)compression operations. When an ED receives a compressed Interest, it does not need to calculate all the missing fields to generate the Data or to forward the Interest. Furthermore, if the decompression is required, each field can be extracted separately with TLV encoding. When a Data packet reaches the gateway, it is decompressed by adding/updating the needed bytes, and then forwarded to the backbone.

Using the data-centric security of NDN, the data related to each cow is signed directly when it is collected by the corresponding ED. Hence, every cow has a unique identity (not an address) in the network system. This identity is securely bound to its data at network level.

Moreover, data authenticity is not compromised by packet compression. When each ED signs its Data, the original (i.e., uncompressed) Data packet is signed before the compression. At the gateway, the decompression process adds the exact bytes needed to make the signature verification correct. This way, if the gateway is partially compromised but the security information is safe, the decompressed packet will contain errors, and this will be detected by the consumer. When the Data signature is delegated to the gateway, the Data packet is signed by the gateway after decompression.

As an empirical evaluation, we report in Table 4 an overall comparison between NDN-IoT and 6LoWPAN features.

7.2 Theoretical Performance

Figure 13 depicts a comparison between initial (i.e., uncompressed) Interest and Data packets and their compressed versions. The Data packet represented here does not include the signature, which is about *32* bytes in both initial and compressed Data. Considering the packet structures in the deployment described above, the size of the Interest is reduced from *57* bytes to *24* bytes. Similarly, the Data packet size is reduced from *93* bytes to *34* bytes.

Note that the most part of the compression gain is achieved by reducing name size in the packets. That is, when CP becomes longer such as in large scale deployments, the compression gain will increase. Moreover, this compression gain comes at the cost of only few microseconds of delay, as no complex processing or context storage is required.

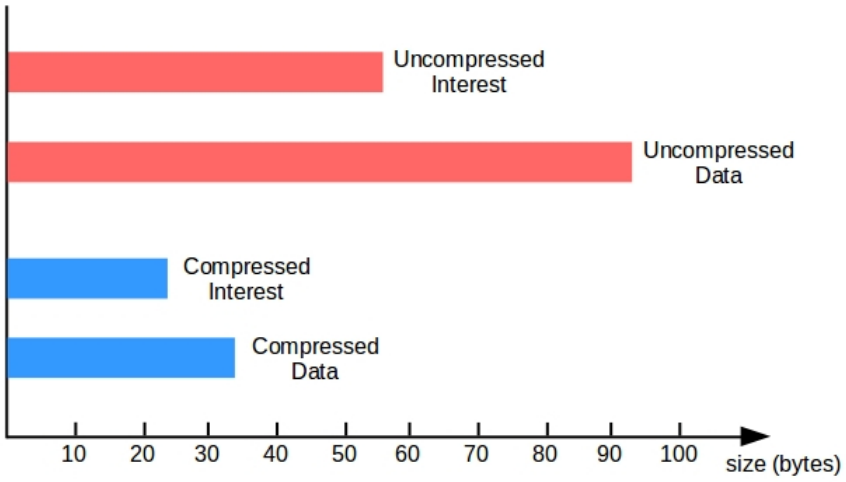

Figure 13: Compression improvement

7.3 Prototype Measurements

Table 5 reports on memory and processing time required by the NDN stack implementation in EDs considering both Arduino UNO (*16 Mhz* MCU) and DUE (*84 Mhz* MCU).

The forwarding process delay with R-LF for the first Interest (i.e., unknown prefix name) is approximately $145\mu s$ on Arduino UNO and $55\mu s$ on Arduino DUE. Subsequent Interests forwarding (i.e., known prefix name) takes about $50\mu s$ on Arduino UNO with *5* entries in the FIB, and $10\mu s$ on Arduino DUE considering *10* FIB entries. This delay difference between first and subsequent forwarding is mainly due to the random number generation which is used in R-LF for only the first Interest forwarding. With R-LF, a FIB entry update after receiving a Data packet consists on a reinforcement learning computation and takes $70\mu s$ on Arduino UNO and $18\mu s$ on Arduino DUE. The measured values show the simplicity of forwarding decisions in the R-LF strategy.

Concerning memory space required by the implementation, only 28% of flash memory and 50% of RAM are needed in the Arduino UNO. The implementation on the Arduino DUE occupies 6% of the total memory. The evaluated implementation includes the three NDN data structures, the communication over IEEE 802.15.4 including packet compression scheme and the R-LF strategy. Although these values increase when adding NDN packet definition and security algorithms, the leeway is still large for such components.

As an empirical comparison, some open source implementations of the IPv6 stack over IEEE 802.15.4 on Arduino (Mega) take about 12% storage and 45% RAM, while our NDN stack occupies about 3% storage and 12% RAM on an Arduino Mega board.

Table 6 reports on the one-hop Round Trip Time (RTT) measured at the gateway to send an Interest and get the corresponding Data using Arduino DUE devices, and the Compression Delay (CD) added to the communication due to packet compression.

In comparison, the measured RTT is below 6LoWPAN performance usually reported (e.g *9* to *25 ms*) [8, 9]. However, 6LoWPAN packets are not signed and additional layers are required to support data naming, while the measured RTT includes Data creation and signature, and Interest-Data (de)compression. Moreover, we recall that our implementation is a prototype that can be improved.

The compression delay (CD) does not exceed $6\mu s$ as it only consists on adding/skipping bytes while transmitting a packet.

Table 5: Memory and processing measurements

Operation	Arduino UNO	Arduino DUE
First Interest forwarding	$145\mu s$	$55\mu s$
Subsequent Interest forwarding	$50\mu s$	$10\mu s$
Data forwarding	$50\mu s$	$10\mu s$
Reinforcement	$70\mu s$	$18\mu s$
Required memory	28% (Flash) 50% (RAM)	6% (Total)

Table 6: Communication measurements at the gateway

Operation	Measure
Round-Trip Time	$72ms$
Compression Delay	$6\mu s$

7.4 Wireless Forwarding Performance

To evaluate the R-LF strategy from a networking perspective, we consider a scenario that reflects our smart agriculture deployment in a larger scale. The network topology consists of *16* nodes (EDs) organized in a grid of *180m* x *180m*. Each of the *16* nodes produces *100* different contents under a specific name prefix: */cowHealth/farm/area/1/cow/<cowID>*. A randomly placed consumer represents the gateway and continuously requests content by issuing Interests. The content requested in our scenario can be considered as Web content which usually follows Zipf distribution [7]. The requested content item is chosen as follows. One of the *16* possible contents (i.e., EDs) is chosen according to the Zipf distribution with $\alpha = 1.3$, then one content item is uniformly chosen among the *100* available items.

Figure 14 gives an example of the simulated topology, and Table 7 reports the relevant simulation parameters.

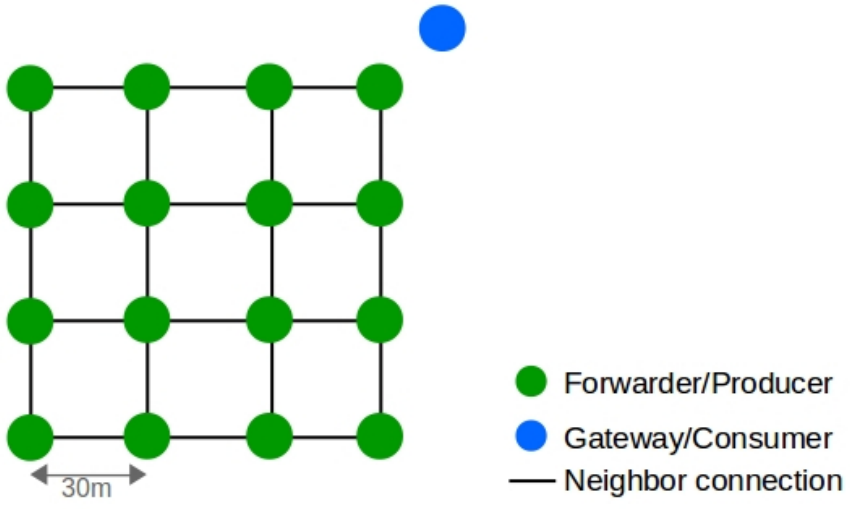

Figure 14: Simulated topology

Table 7: Simulation parameters

Parameter	Value
Data packet size	*34 B*
Interest packet size	*24 B*
Interest send interval	*1 s*
Max Interest re-transmissions (at the gateway)	*1*
Cache size	*20* packets
Cache replacement	LRU
Data freshness	*60 s*
Wireless bit-rate	*250 Kbps*
Wireless MAC protocol	*802.15.4 CSMA*
Communication range	*35 m*

We evaluated the R-LF strategy compared to CF-127 and CF-255, which correspond to the CF strategy with $dw = 127$ and 255 respectively (see Section 3). Preliminary simulations are used to set the best R-LF parameters as $\alpha = 0.85$, $M = 5.0$, $N = 3.5$, $\hat{\Delta} = 9$, and $th = 0.75$.

The following metrics are measured:

- The total number of successfully transmitted frames to measure the generated overhead.
- The mean round-trip time (RTT) for an Interest-Data exchange measured at the gateway.
- The Interest satisfaction rate at the consumer (i.e., gateway) to check the accuracy of the forwarding decisions.
- The mean delay time at link-layer (i.e. back-off) spent by a node before getting access to the medium to transmit a frame.

These metrics have been chosen for the following reasons. First, they are related to the objective of our forwarding strategy to take advantage of broadcast without suffering from its inconvenient. That is, the number of transmitted frames over the network indicates if the broadcast effect is attenuated, the RTT ensures that waiting delays are not too high, and the Interest satisfaction rate measures the data delivery efficiency of each approach. Moreover, the back-off time indicate how much the strategy reduces medium access contention. Second, the four metrics mutually impact one another and it is difficult to optimize the four at the same time. Third, evaluation of forwarding strategies in related work usually measure these metrics or equivalent ones.

In all the simulations, the reported results correspond to the average values obtained with 7 random executions. Results are reported in Figure 15.

We observe that the three strategies (i.e., R-LF, CF-127 and CD-255) achieve the same Interest satisfaction rate which is slightly close to 100%. However, this Interest satisfaction rate does not come at the same cost for all the approaches as discussed below.

The results show that the total number of frames transmitted by R-LF is significantly lower than CF-127 and CF-255. Given that both CF and R-LF use only broadcast, this indicates that forwarding decisions with R-LF are gratly improved. Moreover, as only request and response packets exist in R-LF and CF, the low number of transmitted frames indicates that R-LF is able to choose the best paths to retrieve content, which is not guaranteed when the forwarding relies on Interest flooding with random delays for example. Notice that reducing the overhead is one of the design goals of R-LF.

R-LF outperforms CF in terms of RTT even though both strategies use waiting delays to forward packets. That is, the superiority of R-LF mainly results from the accuracy of

the waiting delays used to forward Interests. Furthermore, as Interest forwarding decisions are more accurate, forwarders do not need to defer Data packet transmissions.

Moreover, the accuracy of forwarding decisions with R-LF greatly impacts the contention access on the wireless medium. This is clearly shown by the reduced back-off time measured with R-LF compared to CF.

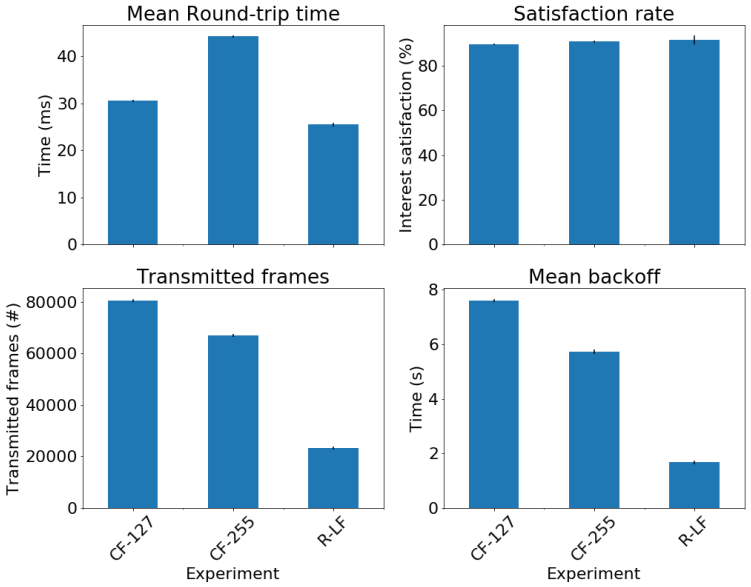

Figure 15: R-LF evaluation results

8 Discussion and Conclusion

We investigated in this paper how to take advantage of NDN for the IoT in a simple and feasible solution. To that purpose, a realistic NDN-IoT architecture has been designed and realized considering the IEEE 802.15.4 wireless technology. After identifying the integration of NDN in the low-end IoT as the most realistic approach, the main integration issues have been discussed, and some mechanisms have been proposed. The proposed mechanisms show the flexibility of NDN to support low rate lossy technologies such as the IEEE 802.15.4. The NDN-IoT architecture proposed aims to shape a novel and strong NDN-IoT duo.

Moreover, lightweight NDN forwarding in wireless networks with broadcast has been investigated. The objective through the forwarding approach proposed is to show that broadcast can be used successfully in constrained networks, while ensuring reduced overhead and accurate forwarding decisions. For that, we designed R-LF; a forwarding strategy based only on content names and broadcast without any host identification. R-LF is a reactive forwarding strategy that does not require additional communication to maintain forwarding information. Results obtained show that R-LF is able to provide efficient data retrieval using exclusively the content-centric paradigm of NDN, without any host identification such as logical or link-layer addresses.

The deployment evaluation provides some preliminary measurements and empirical comparison with IP-based solutions. For example, the deployment may show the lightweight and simplicity of NDN implementations for IoT.

Overall, the main limitation we may identify in this work is the lack of direct performance comparisons between NDN and IP. Although it could be useful, this can be explained by several reasons. First, we greatly rely on the discussions about IP limitations and NDN native features to show the superiority of NDN, which is indisputable in many aspects such as security, native caching and simplicity. Second, when comparing IP and NDN in a given scenario, the fairness of the configurations is frequently questioning, for example when enabling caching.

References

[1] Alexander Afanasyev, Junxiao Shi, Beichuan Zhang, and NFD Team. "NFD developer's guide". Technical Report NDN-0021, NDN, February 2015.

[2] Marica Amadeo, Claudia Campolo, Antonio Iera, and Antonella Molinaro. "named data networking for iot: an architectural perspective". *Conference on European Networks and Communications (EuCNC)*, pages 1–5, June 2014.

[3] Marica Amadeo, Claudia Campolo, and Antonella Molinaro. Forwarding strategies in named data wireless ad hoc networks: Design and evaluation. *Journal of Network and Computer Applications*, 50(Supplement C):148 – 158, 2015.

[4] Andrei Klubnikin. IoT Agriculture: How to Build Smart Greenhouse?

[5] Emmanuel Baccelli, Christian Mehlis, Oliver Hahm, Thomas C. Schmidt, and Matthias 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-ICN '14, pages 77–86, New York, NY, USA, 2014. ACM.

[6] Justin A. Boyan and Michael L. Littman. Packet routing in dynamically changing networks: A reinforcement learning approach. In *Proceedings of the 6th International Conference on Neural Information Processing Systems*, NIPS'93, pages 671–678, San Francisco, CA, USA, 1993. Morgan Kaufmann Publishers Inc.

[7] L. Breslau, Pei Cao, Li Fan, G. Phillips, and S. Shenker. Web caching and zipf-like distributions: evidence and implications. In *IEEE INFOCOM '99. Conference on Computer Communications. Proceedings. Eighteenth Annual Joint Conference of the IEEE Computer and Communications Societies. The Future is Now (Cat. No.99CH36320)*, volume 1, pages 126–134 vol.1, March 1999.

[8] Brendan Cody-Kenny, David Guerin, Desmond Ennis, Ricardo Simon Carbajo, Meriel Huggard, and Ciaran Mc Goldrick. Performance evaluation of the 6lowpan protocol on micaz and telosb motes. In *Proceedings of the 4th ACM Workshop on Performance Monitoring and Measurement of Heterogeneous Wireless and Wired Networks*, PM2HW2N '09, pages 25–30, New York, NY, USA, 2009. ACM.

[9] G. Gardasevic, S. Mijovic, A. Stajkic, and C. Buratti. On the performance of 6lowpan through experimentation. In *2015 International Wireless Communications and Mobile Computing Conference (IWCMC)*, pages 696–701, Aug 2015.

[10] Github. "NDN client library with TLV wire format support in native Python".

[11] IEEE. Ieee standard for information technology– local and metropolitan area networks– specific requirements– part 15.1a: Wireless medium access control (mac) and physical layer (phy) specifications for wireless personal area networks (wpan). *IEEE Std 802.15.1-2005 (Revision of IEEE Std 802.15.1-2002)*, pages 1–700, June 2005.

[12] IEEE. Ieee standard for local and metropolitan area networks–part 15.4: Low-rate wireless personal area networks (lr-wpans). *IEEE Std 802.15.4-2011 (Revision of IEEE Std 802.15.4-2006)*, pages 1–314, Sep. 2011.

[13] IEEE. Ieee standard for information technology—telecommunications and information exchange between systems local and metropolitan area networks—specific requirements - part 11: Wireless lan medium access control (mac) and physical layer (phy) specifications. *IEEE Std 802.11-2016 (Revision of IEEE Std 802.11-2012)*, pages 1–3534, Dec 2016.

[14] IEEE. Ieee draft trial-use standard for wireless access in vehicular environments (wave) - resource manager. *IEEE Std P1609.1/D17, Jul 2006*, pages 1–66, April 2019.

[15] Teng Liang, Ju Pan, and Beichuan Zhang. Ndnizing existing applications: Research issues and experiences. In *5th ACM Conference onInformation-Centric Networking (ICN '18)*, September 2018.

[16] LoRa Alliance. "website LoRa Alliance".

[17] Meisel Michael, Pappas Vasileios, and Zhang Lixia. Listen first, broadcast later: Topology-agnostic forwarding under high dynamics. In *Annual conference of international technology alliance in network and information science*, 2010.

[18] Gabriel Montenegro, Jonathan Hui, David Culler, and Nandakishore Kushalnagar. Transmission of IPv6 Packets over IEEE 802.15.4 Networks. RFC 4944, September 2007.
[19] John Mueller. "Understanding SOAP and REST Basics And Differences", 2013.
[20] Luca Muscariello, Giovanna Carofiglio, Jordan Auge, and Michele Papalini. Hybrid Information-Centric Networking. Internet-Draft draft-muscariello-intarea-hicn-01, Internet Engineering Task Force, December 2018. Work in Progress.
[21] PrithviRaj Narendra, Simon Duquennoy, and Thiemo Voigt. Ble and ieee 802.15.4 in the iot: Evaluation and interoperability considerations. In Benny Mandler, Johann Marquez-Barja, Miguel Elias Mitre Campista, Dagmar Cagáňová, Hakima Chaouchi, Sherali Zeadally, Mohamad Badra, Stefano Giordano, Maria Fazio, Andrey Somov, and Radu-Laurentiu Vieriu, editors, *Internet of Things. IoT Infrastructures*, pages 427–438, Cham, 2016. Springer International Publishing.
[22] NDN. NDN Common Client Libraries (NDN-CCL) Documentation.
[23] NDN. NDN Testbed status page.
[24] NDN Project Team. "NDN technical memo: Naming conventions". Technical Report NDN-0022, NDN, July 2014.
[25] Python. "pyserial 2.7".
[26] Shailesh Kumar. Confidence-based dual reinforcement q-routing: an on-line adaptive network routing algorithm. Master's thesis, The University of Texas at Austin, 1998.
[27] Wentao Shang, Adeola Bannisy, Teng Liangz, Zhehao Wangx, Yingdi Yu, Alexander Afanasyev, Jeff Thompsonx, Jeff Burkex, Beichuan Zhangz, and Lixia Zhang. Named Data Networking of Things (Invited paper). In *The 1st IEEE Intl. Conf. on Internet-of-Things Design and Implementation*, Berlin, Germany, April 2016.
[28] Wentao Shang, Yingdi Yu, Ralph Droms, and Lixia Zhang. Challenges in IoT networking via TCP/IP architecture. Technical Report NDN-0038, NDN, February 2016.
[29] Zach Shelby, Klaus Hartke, and Carsten Bormann. The Constrained Application Protocol (CoAP). RFC 7252, June 2014.
[30] Sigfox. "website Sigfox".
[31] NDN Team. "NDN Packet Format Specification".
[32] H. Wu, J. Shi, Y. Wang, Y. Wang, G. Zhang, Y. Wang, B. Liu, and B. Zhang. On incremental deployment of named data networking in local area networks. In *2017 ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS)*, pages 82–94, May 2017.
[33] George Xylomenos, Yannis Thomas, Xenofon Vasilakos, Michael Georgiades, Alexander Phinikarides, Ioannis Doumanis, Stuart Porter, Dirk Trossen, Sebastian Robitzsch, Martin J. Reed, Mays F. Al-Naday, George Petropoulos, Konstantinos V. Katsaros, Maria-Evgenia Xezonaki, and Janne Riihijärvi. IP over ICN goes live. *CoRR*, abs/1804.07511, 2018.
[34] Z. Zhang, Y. Yu, H. Zhang, E. Newberry, S. Mastorakis, Y. Li, A. Afanasyev, and L. Zhang. An overview of security support in named data networking. *IEEE Communications Magazine*, 56(11):62–68, November 2018.