Future Generation Computer Systems 93 (2019) 792–798

Contents lists available at ScienceDirect

Future Generation Computer Systems

journal homepage: www.elsevier.com/locate/fgcs

NDN-over-ZigBee: A ZigBee support for Named Data Networking

Amar Abane [a,b], Mehammed Daoui [a], Samia Bouzefrane [b,*], Paul Muhlethaler [c]

[a] LARI Lab, University Mouloud Mammeri of Tizi-Ouzou, Algeria
[b] CEDRIC Lab, Conservatoire National des Arts et Metiers, Paris, France
[c] Eva team, Inria, Paris, France

HIGHLIGHTS

- In the context of the IoT, an implementation for NDN over ZigBee is proposed.
- It targets typical IoT communications which require low-power consumption.
- The proposed implementation also adds NDN sensing abilities.
- The objective is to enhance NDN's suitability for IoT.

ARTICLE INFO

Article history:
Received 12 March 2017
Received in revised form 1 August 2017
Accepted 22 September 2017
Available online 25 October 2017

Keywords:
Named Data Networking
ZigBee
IoT

ABSTRACT

Named Data Networking (NDN) is a new architecture which allows communications using data's natural names rather than hosts' logical addresses. In recent years, several research projects have demonstrated the ability of NDN to support emerging IoT applications like home automation, smart cities and smart farming applications. This work aims to integrate NDN with ZigBee to give NDN a better support for IoT applications that are known to require wireless sensing/actuating abilities, mobility support and low power consumption. For this purpose, we present our NDN-over-ZigBee design and we show through experiments conducted with three different scenarios the suitability and the ease of use of NDN in IoT context. The choice of ZigBee is motivated by the fact that it is a network specification for low-power wireless personal area networks (WPANs) and supports a large number of nodes.

1. Introduction

The Internet of Things (IoT) is mainly built by smart embedded devices which bring new requirements in terms of connectivity, mobility support and efficient power-management. From a communication point of view, these devices are mostly connected via TCP/IP protocols. However, it is becoming clear that these (traditional) protocols are quite far from intrinsically addressing IoT platforms issues [1], such as handling complex and dynamic mobile networks, strongly securing communications and data exchanges, or linking the information with its location efficiently to easily discover it.

Unlike the Internet Protocol (IP), Named Data Networking (NDN) is a new architecture which proposes to access data using natural names rather than IP addresses of the hosts. Such a mechanism facilitates application development because the design does not consider host location. Also, this new architecture can support mobility of a large number of nodes which makes it very compatible with IoT requirements. Moreover, the security in NDN is based on the data, making it more flexible to support different security requirements.

To visualize NDN's advantages, suppose we need to design a communication solution for a large scale scenario: water balance management. The aim of the water balance is to estimate the intake and consumption of drinking water around one or several sources (e.g. dam). This balance is calculated on the basis of precipitation, consumption by the population, ground humidity, etc. An important issue in this case is the use of different wireless technologies in the same solution. Indeed, in a realistic design, a certain wireless standard (and its related equipments) is more suitable for one part of the architecture, when another standard is unavoidable for another part. For example, ZigBee is widely used in WSNs (Wireless Sensor Networks), thus it is a wise choice to use it for precipitation monitoring and ground humidity, since the coverage area is delimited and known; but it is more suitable to manage the population consumption by a solution like LoRaWAN, since each sensor (e.g. water consumption meter) will be independent and the communication range is very wide (Kilometers). In addition, each standard has its security mechanisms and obviously its flaws, as

[*] Corresponding author.
E-mail address: samia.bouzefrane@lecnam.net (S. Bouzefrane).

https://doi.org/10.1016/j.future.2017.09.053
0167-739X/© 2017 Elsevier B.V. All rights reserved.

well as the involved entities have different security requirements. All the technical challenge is to integrate these heterogeneous pieces to provide a uniform and reliable data acquisition. Security mechanisms must also be unified above the used protocols to ensure an end-to-end security from the sensors to the applications. In this situation, we can report important requirements for which the NDN's approach is more elegant and more efficient:

- *Allow applications to uniformly request data regardless whether it is provided by ZigBee, LoRaWAN or another technology.* NDN's approach naturally has this property; each data is accessed by its name without establishing and maintaining explicit connections between servers, applications and sensors. To provide an equivalent feature, a host-based (IP-based) architecture needs a middleware to bind each data with the corresponding sensor. Middlewares add protocol overhead, complexity and cause communication latency due to the additional processing.
- *Allow applications to check the origin of each sensed value even if the sensor is offline, replaced or has joined another local network.* A host-based approach cannot natively provide this kind of security, because the sensors must be available when the application retrieves the data in order to check its authenticity. Host-based architecture also needs middlewares and servers to implement this feature. In contrast, NDN allows to check authenticity of each Data packet at any time, even if the sensor is not available when the Data is consumed; the mandatory signature carried in each Data packet and the associated trust model make this verification possible.
- *Reuse the same data.* We can observe in a lot of IoT scenarii that applications ask for the same content at different times. In host-based architectures, each data is in a packet which involves two hosts (source and destination), thus reusing the same packet for different requests is not possible. A Data packet in NDN is independent from any source or destination address. This allows Data packets to be cached to satisfy further requests, and prefetch some popular packets to satisfy the requests faster.

On the other hand, ZigBee is a communication protocol designed to suit embedded device requirements. It provides low power consumption and supports a great number of devices over long distances with many different topologies. Nowadays, the success of many IoT solutions proves the efficiency of the ZigBee specification and its compatible hardware. Thus, if we provide NDN with ZigBee features, it will be much more attractive for IoT applications development.

In this paper, we propose to leverage the strengths of NDN and ZigBee, and combine them to make a new step towards IoT application development. In real terms, we propose an implementation that allows NDN communication over the ZigBee protocol acting as a layer 2 support for NDN. Since our implementation can run on any Linux distribution with an NDN module (NFD[1]), we believe that it will allow NDN to target a larger set of devices and gateways, and cover more IoT applications.

This paper is organized as follows. Section 2 presents a brief description of the NDN architecture, an overview of wireless supports and their integration with NDN. Section 3 details the design of the proposed NDN-over-ZigBee layer. Section 4 describes the performance tests undertaken to evaluate our implementation, and discusses the results obtained. Section 5 concludes our work with future perspectives.

2. Background

2.1. NDN overview

NDN [2] follows the Content-Centric Networking (CCN) architecture [3] which is based on the Information-Centric Networking (ICN) paradigm [4]. The basic principle of NDN is to identify the data rather than its holder. While the TCP/IP architecture is based on source and destination addresses to identify hosts and ensure communication, NDN uses data's natural names. The hourglass architecture of NDN contains in its thin waist all the relevant network operations which are traditionally provided by higher network layers (see Fig. 1). For instance, flow control and security mechanisms are supported in the network layer of the NDN architecture while they are implemented in transport and application layers in the IP architecture.

NDN defines two types of packets [5]: Interest and Data. Each NDN node maintains three data structures to ensure its operations: FIB (Forwarding Interest Base), PIT (Pending Interest Table) and CS (Content Store). The communication is consumer driven, which means that it is initiated by the consumer (i.e., the one who asks for data). As a consequence, an NDN application can play two roles: a consumer role if the application sends Interests to ask for Data packets, or a producer role if the application responds to incoming Interests by sending Data packets.

NDN packets do not carry any source or destination address. Interest forwarding is completely based on names. When a Data packet arrives in a node, the PIT is checked and the Data packet is forwarded to all the originating interfaces or applications (abstracted in NDN as "Faces") of the corresponding Interest. After that, the node discards the Interest from the PIT, and keeps for a while the recently forwarded Data packet in its CS. Thus, before forwarding an incoming Interest, a node firstly checks if the requested Data is available in its CS. If a corresponding Data is found, it is sent back as a response without forwarding the Interest any further. There is no limitation on NDN names, because the namespaces are hierarchical and unbounded, and the routers handle names as sequences of components bounded by "/" . The hosts' mobility does not affect a communication since the names of Interests and Data are kept coherent.

NDN was especially designed to support new communication patterns and emerging applications like those dedicated to IoT. Many research projects [6–9] demonstrate in different ways and areas that NDN is a promising architecture to build IoT applications as well as global IoT platforms.

2.2. NDN and wireless communication

The IEEE 802 Standard is a set of networking standards for both wired and wireless networks. The most known wireless specifications include 802.11 (WiFi), 802.15.4 (ZigBee) and 802.15.1 (Bluetooth). Due to its satisfactory bandwidth and admissible cost, WiFi nicely meets LAN requirements and is widely used in businesses and homes. However, there are network types that focus on other aspects like low power consumption, great numbers of nodes and long distances. Thankfully, many physical specifications were designed to support these aspects. For example, Bluetooth is designed for low-power wireless personal area networks (WPANs) and allows a 1–2 Mb/s data rate with an acceptable power consumption (BLE) and complexity. Other communication specifications were proposed for specialized networks such as DSRC for VANETs and 3G/4G for very long distances. Also, new wireless technologies explicitly designed for IoT have recently appeared such as Sigfox [10] and LoRaWAN [11].

Currently, the focus is on the Internet of Things and wireless sensors/actuators communication. Although this kind of communication does not generally require a large bandwidth, it needs an

[1] NFD stands for NDN Forwarding Daemon.

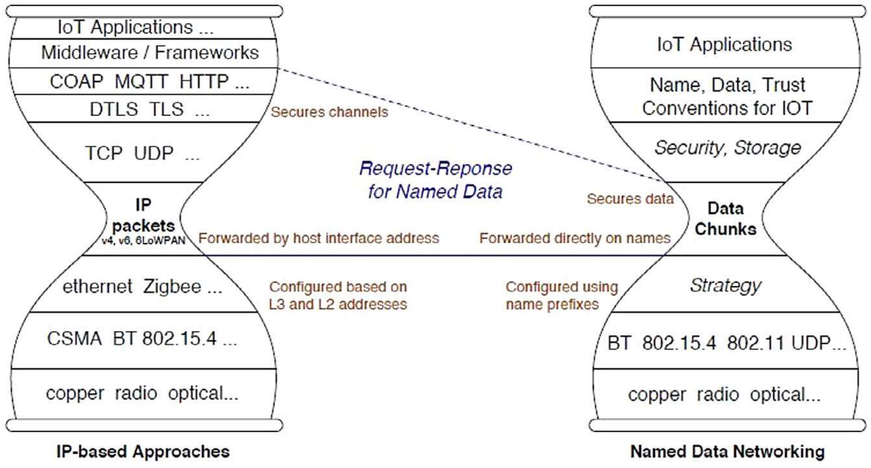

Fig. 1. Architecture comparison of NDN and IP [9].

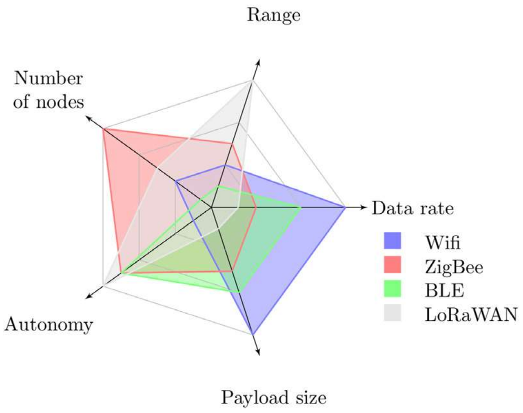

Fig. 2. Wireless technologies comparison.

efficient power management plan, a low cost of production and needs to support a great number of –mobile-nodes in a simple way. To meet these requirements, ZigBee was designed as a mesh network specification for WPANs. It was mainly designed to support a large number of nodes (sensors) over long distances and provide low power consumption since most ZigBee devices are powered by battery. This has led ZigBee to be closely associated with WSN communication and the Internet of Things.

To summarize, Fig. 2 gives a comparison of some relevant wireless technologies involved in the IoT according to some important evaluation criteria.

As said above, NDN is a promising network architecture for IoT and mobile communications. Many NDN-based IoT architectures have to use proprietary protocols to manage sensors/actuators rather than using NDN in their whole solution. Hence, the integration of NDN with ZigBee may be very interesting and will allow NDN to cover more emerging applications such as home automation, smart cities, smart farming, etc. In addition, ZigBee is designed to simply allow sensing and actuating (through XBee modules for example [12]). This particular feature can be integrated with NDN to provide an almost native sensing/actuating ability.

An implementation of the NDN protocol stack is now available for RIOT-OS [13] (NDN-RIOT [14]) which basically supports the IEEE 802.15.4 specification; but RIOT-OS is a specialized platform which targets a limited set of devices, and we believe that a more general support will be beneficial and useful. For example, one may need to use ZigBee as a wireless support in an existing NDN application that runs on a general purpose platform such as Linux.

As regards other wireless specifications, WiFi is currently supported by NDN implementation, and an implementation of NDN over Bluetooth (called NDNBlue) was proposed in [15].

Generally, the main task in a L3 protocol integration with low rate wireless networks is the header compression (e.g., 6LoWPAN [16]). Fortunately, since NDN packets are not based on predefined field size (unlike IP), the NDN integration can be naturally designed: it is sufficient to ban some unrelevant fields and limit the size and number of components in NDN names. However, an NDN communication is consumer driven; meaning that only the consumer can initiate the communication. Knowing that low rate wireless protocols also have communication constraints (like sleep mode for sensors), the integration must find a way to adapt the pull-based nature of NDN to the communication pattern of the considered wireless protocol. For example, Class A devices in LoRaWAN cannot be reached instantly by the gateway; this characteristic needs to carefully choose how to combine NDN with LoRaWAN. In the case of ZigBee, we have made some design choices which will be described in details in the next section.

In all cases, NDN integration with low rate wireless protocols can play an important role in building a global IoT platform. The device address will be used only for local communications, while global communication and security will be ensured based on the content names. This will allow heterogeneous wireless technologies to be used to produce the same Data packet format which is bound to its producer (sensor). In addition, by combining packet reuse with the small headers of low rate wireless protocols, the network will transmit much less overhead than IP-based solutions. Since the NDN security is built on the content, we believe that shifting the security paradigm from channels to contents can remedy the security flaws of the underlying low rate wireless protocols.

However, due to the small payload of low rate wireless protocols; the first rule for NDN integration is to limit names length. This will give less expressive names, but encoding mechanisms can be used to carry more information in small names. Forwarding strategies choice is also limited: although NDN provides plenty of possibilities to select interfaces for Interests forwarding, wireless nodes can only rebroadcast the received Interests, increasing the probabilities of collisions. However, the case of one hop communications (devices-gateway) is not much affected. Low rate wireless protocols can also limit the integration possibilities. For example, the limited packet number and the restricted payload in LoRaWAN, and the small number of nodes in BLE creates a lot of limitations. Fortunately, as Fig. 2 shows, ZigBee technology provides satisfying and balanced characteristics for NDN integration, which is described in the next section.

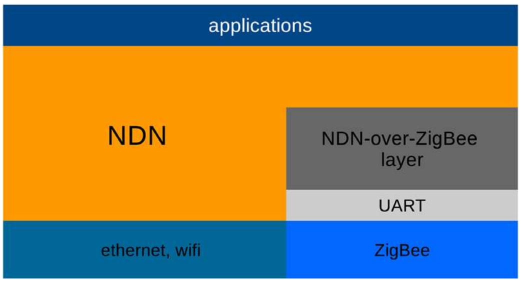

Fig. 3. Software and hardware layers of an NDN-over-ZigBee node.

3. NDN-over-ZigBee design

For the reasons given in the last section, we have chosen to use ZigBee protocol as a wireless communication support for NDN. To achieve this, we have developed a software layer added to the NDN module. The role of this layer is to intercept Interests with a certain prefix name and to send them through ZigBee. We have decided not to integrate this layer into the NDN software module for two main reasons: Firstly, the implementation of NDN module (called NFD [17]) is under development and may substantially change and cause stability issues. Secondly, the hardware modules used for ZigBee communications (called XBee) offer I/O pins that are used to handle sensors and actuators for many interesting IoT applications; hence integrating I/O pins management in the NDN module is deprecated since the NDN module has become a complete network protocol. Fig. 3 depicts the software and hardware layers stack after integrating the ZigBee layer.

3.1. Software structure

A simple NDN application can act as a producer or a consumer. Thus, an NDN node that runs an application is generally either sending Interests to get Data (consumer) or waiting for Interests to respond with Data (producer). This behavior leads us to divide the ZigBee layer into two independent processes: the first manages sending Interests and getting Data (emission process); the second receives incoming Interests and sends Data back (reception process). In this way, each node can use only one of the two processes according to its role: emission process for a consumer, reception process for a producer. This design choice provides a lightweight and less complex NDN-over-ZigBee layer running in each node, and avoids hardware access conflict. However, a router or a gateway node can be considered as a producer; thus it uses the reception process.

To handle communication errors, the emission process can detect when the remote ZigBee hardware is not responding. In the other side, the reception process can detect and inform the emission process when the remote application is not responding. In this way, a consumer application that uses our NDN-over-ZigBee layer can identify whether the remote ZigBee hardware or the remote application is not responding.

3.2. Sensors management

In addition to the management of ZigBee communications, the NDN-over-ZigBee layer is able to manage sensors that are connected to ZigBee hardware modules. In fact, in addition to receiving Interests, the reception process can also read the state of its hardware module pins, perform a personalized process on each value, and send personalized Interests that carry the result in their names (notification Interest [18]) through another network interface (Ethernet, Wifi, etc.).

By sending local sensor data using Interests rather than the basic NDN communication pattern, a node can send sensor information in real time and the whole network can benefit from it. This feature is cheap and mainly designed for gateway nodes to provide them with real-time sensing capability in addition to their main role (gateway). For example, gateway nodes can collect data related to the environment and send them to servers and/or routers for information purposes.

In addition to the personalized Interest name, pins' values may need to be converted to meaningful information. To couple each input pin with its corresponding treatment and Interest name, we use a configuration file. This file makes it possible to define what function is associated to each pin, and which Interest name must be sent after reading each pin. Reading input pins is a task performed periodically; the period is also defined in the configuration file.

In both processes, control bytes and timeouts are used to drive the communication. The following algorithms describe the main actions of each process[2] . Algorithm 1 recaps the emission process. Algorithm 2 recaps the reception process. Algorithms 3 and 4 are the callback functions used by the Algorithm 2 respectively when Interest timeout occurs and when the Data packet arrives:

```
Function SendInterest
Data: Interest from NDN daemon
begin
    Send Interest through ZigBee frames
    Get ZigBee frames from remote XBee module
    Create the Data packet from buffered frames
    Send Data to NDN daemon
end
```

Algorithm 1: Emission process.

```
Function ReceiveFrame
Data: frame from ZigBee HW module
begin
    if the frame type is a local sensing frame then
        Send notification Interest carrying the corresponding value
    else
        if the frame type is a remote data frame then
            Add the frame and its source address to the buffer
        end
        if the frame is the last frame of the Interest then
            Create Interest from buffered frames
            Associate the Interest to 'onTimeout' and 'onData'
            callbacks
            Send Interest to NDN Daemon
        end
    end
end
```

Algorithm 2: Reception process.

```
Function onTimeout
Data: ZigBee source address
begin
    Send ZigBee frame('application not responding' , ZigBee
    source address)
end
```

Algorithm 3: On Timeout callback.

```
Function onData
Data: NDN Data packet , ZigBee source address
begin
    Send ZigBee frames(NDN Data packet , ZigBee source address)
end
```

Algorithm 4: On Data callback.

[2] In the actual code, additional verifications are included.

Table 1
Typical FIB of an NDN-over-ZigBee node.

Consumer			Producer, gateway or router		
Prefix	Face	Cost	Prefix	Face	Cost
/zigbee	Local emission process	0	/zigbee	Local application or next hop	0 (or +)

3.3. ZigBee layer-2 addressing

In our implementation, the ZigBee protocol acts as a layer-2 support for NDN. It uses the 16-bit addressing defined in the ZigBee specification: the emission process needs a 16-bit destination address to send its Interests. The default destination address for the emission process is the broadcast address 0xFF, but we can specify any other destination address when starting the emission process. For example, if a node needs to communicate with a fixed gateway, we can use the address 0x00 (which is the default address of the coordinator node in the ZigBee specification) as a destination address.

When responding, the reception process uses the source address of the incoming Interest as the destination address. This avoids communication conflicts due to broadcasts, and a node running a reception process can manage multiple communications without worrying about address conflicts.

3.4. NDN routing and ZigBee layer

In a consumer node, the emission process needs to intercept Interests with a certain prefix to send them through ZigBee. This prefix is specified during the start of the process and the default value is *'/ZigBee'*. In this way, all the Interests that have names starting with *'/ZigBee'* prefix (or other specified prefix) will be sent through ZigBee. The NDN daemon acts in a normal way; it forwards these Interests to the emission process and when the corresponding Data comes back, it sends them to the appropriate application.

In a producer or a router node, the NDN daemon needs to know the Face corresponding to the ZigBee Interests in order to forward them. This Face can be either a local application (if the node runs a producer application) or a next hop through Ethernet or WiFi (if the node is a router/gateway). Table 1 gives the typical FIBs of a consumer node and a producer/router node.

4. Implementation and tests

The proposed NDN-over-ZigBee layer was implemented in Python using the PyNDN2 library [19]. Thanks to which, our implementation can run on any Linux-based distribution with NFD installed such as Ubuntu, Debian and Raspbian.

In order to test the implementation and check if it is interesting for IoT applications, our experimentations deal with three scenarios: the first one is a point-to-point communication between a consumer node and a producer node. The second one is a devices-gateway communication (i.e. many to one). It consists of two consumer nodes (devices) that interact with one gateway on which a producer application runs. The last scenario is a qualitative evaluation of the sensing features and error detection.

To be as close as possible to usual IoT hardware platforms which are embedded and resource-constrained, we carried out all the experiments with Raspberry Pis 1 model B+[3] and XbeePro S1 modules. For the sensing test in scenario 3, we used a CO_2 gas sensor connected to the producer's XbeePro module.

In scenarios 1 and 2, we performed different sets of Interest-Data exchanges by sending 5, 10, 25, 50 Interests, and restarted again with different Data packet sizes, ranging from 50 bytes to 293 bytes. During these tests, we measured the total time taken by each exchange (from sending the Interest to getting the Data) and also the percentage of satisfied Interests on each set. The results are shown and discussed below.

Scenario 3 is a point-to-point communication in which we voluntarily induced different errors in the producer's side (breaking its application, unplugging its Xbee module, etc.). We also checked whether the reception process performs correctly with respect to the sensing features as well as the appropriate treatments described in the last section.

Figs. 4 and 5 represent the average time taken to retrieve a Data packet, respectively for the point-to-point scenario and the devices-gateway scenario, according to the Data packet size. In both scenarios, this average time does not increase significantly when the Data packet size increases. Considering the tight bandwidth of Xbee modules and by assuming that most of the IoT traffic consists in sending commands/notifications and getting acknowledgments or small information, the results show that our implementation can support this kind of communication.

During the experimentations, we also measured the percentage of Interest satisfaction[4] for the point-to-point scenario and the gateway scenario. In all the tests of Scenario 1, Interest satisfaction was 100%. For Scenario 2, it remains higher than 98% for the biggest Data packet sizes ($>$ 150 bytes) and the small Data packet sizes went out with 100% Interest satisfaction.

Concerning scenario 3, our implementation nicely detected the expected errors such as: a remote Xbee module is not responding, a remote application is not responding, and avoided infinite waiting time for a response. The sensing management was correctly performed according to the specifications of the configuration file.

In addition to these three scenarios, we have made a comparison between our NDN-over-ZigBee implementation and the basic NDN wireless communication over WiFi. For this purpose, we have estimated[5] the additional data (bytes) needed to perform the NDN communication over WiFi as provided by the NDN module (NFD). By default, NFD uses the UDP/IP protocol stack to send NDN packets. Thus, we calculated the overhead needed for our first scenario described above as if it had been made with NDN over UDP/IPv4 with 802.11 (WiFi) as a communication support.

Fig. 6 shows the comparison of overheads between our NDN-over-ZigBee implementation and the basic NDN-over-UDP/IPv4/WiFi with different Data packet sizes. In the NDN-over-UDP/IPv4/WiFi case, the total overhead is the sum of overheads added successively by UDP, IPv4 and 802.11 protocols in order to send one Interest and get one Data. In the NDN-over-ZigBee case, the total overhead is the sum of overheads added successively by ZigBee and our implemented layer in order to send one Interest and get one Data.

Because current network equipment and OSs cannot directly support NDN communications over layer 2 protocols [20], NDN runs on top of UDP/IP or TCP/IP stack. Therefore, the comparison shown in Fig. 6 makes sense; it gives an idea about the complexity and the waste that occurs when running NDN above traditional

[3] 700 MHz CPU and 521 Mo RAM.

[4] The Interest lifetime was fixed to the default value (4 s).

[5] This estimation was made by assuming that each IPv4 packet size is 256 bytes without adding option bytes. Also, security (encryption, etc.) have not been taken into account.

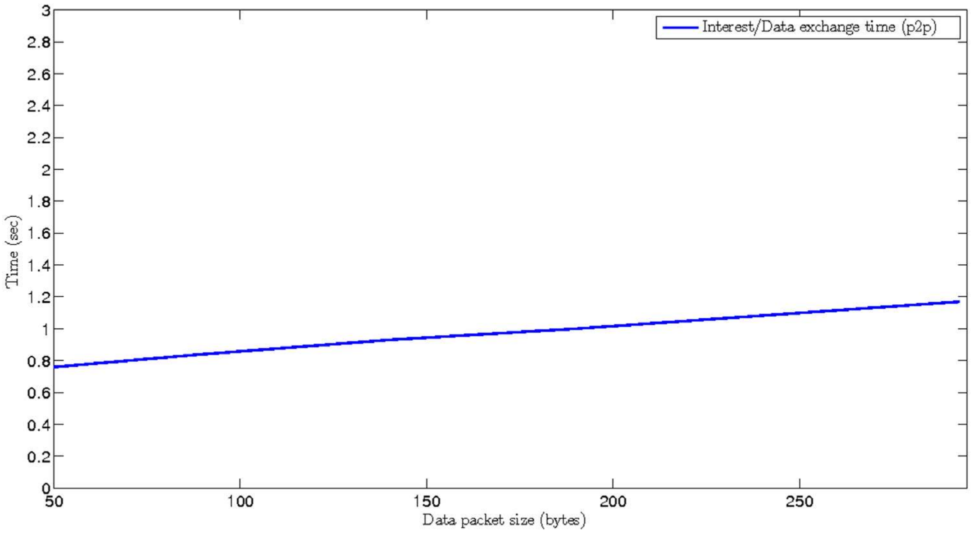

Fig. 4. Data retrieval time (p2p scenario).

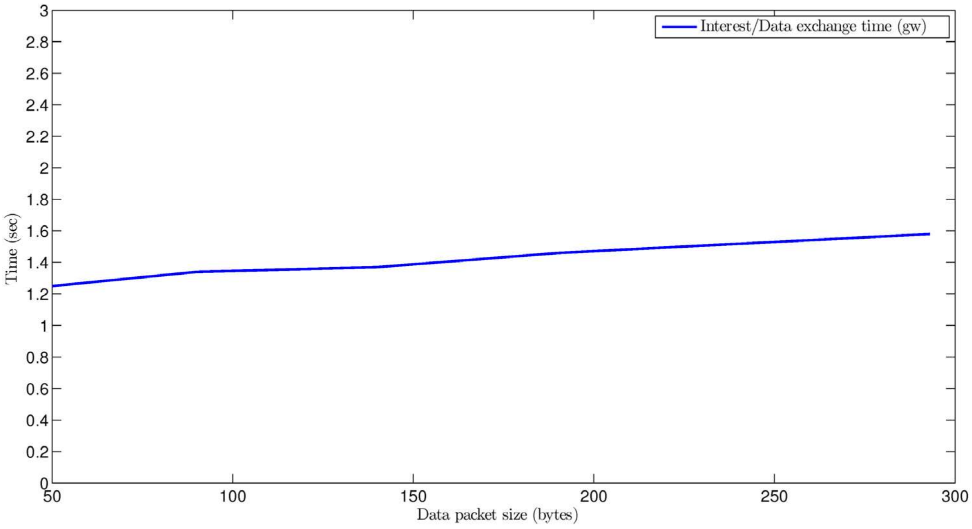

Fig. 5. Data retrieval time (gateway scenario).

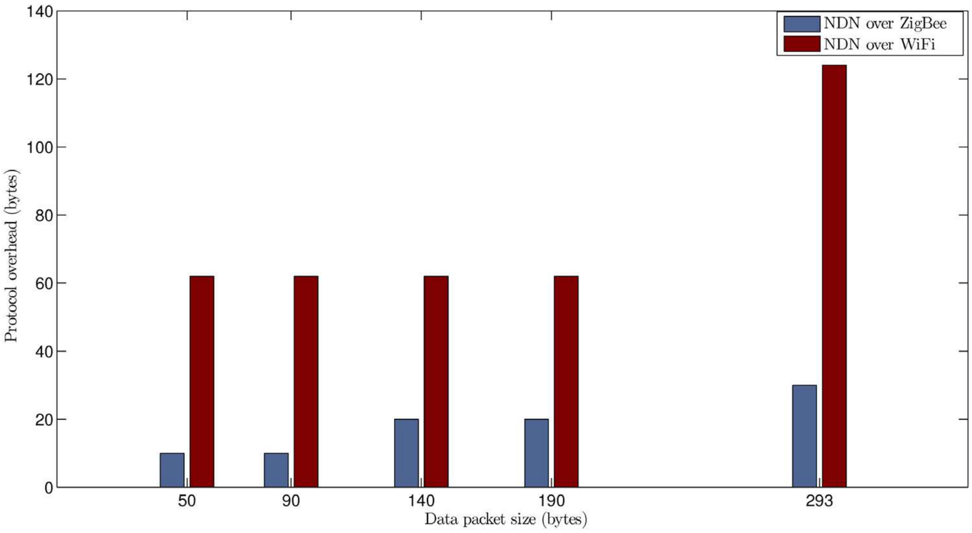

Fig. 6. Total overhead comparison.

protocol stacks comparing to using it with IoT wireless protocols. We can see that the additional data sent when using WiFi is much greater than using ZigBee; and additional data means additional processing and more power consumption. This difference is highlighted when sending a great number of individual commands/notifications or retrieving small data packets, because the large bandwidth offered by WiFi is no longer interesting in these cases. Note that in the case of small information, even if we consider NDN communication directly over WiFi the overhead will be greater than NDN-over-ZigBee overhead. The waste of data presented here has a significant impact on energy consumption, thus the known advantage of ZigBee comparing to WiFi in terms of energy consumption is enhanced by our implementation.

Since the purpose of our implementation is to target NDN/IoT applications, we can say that the results are satisfactory for sending commands/notifications, sensing and pulling small volumes of data. Also, error detection allows applications to manage errors in the network more efficiently. However, the implementation is a prototype that can be improved to reduce Interest-Data exchange delay.

5. Conclusion

In this paper, we proposed and implemented a ZigBee support for NDN communications by extending the NDN stack with an adaptation layer. Since ZigBee is widely used, we aim to target more IoT applications using the proposed NDN implementation. In addition to providing a cheap wireless support, our implementation integrates to NDN nodes sensing capabilities that we hope can open new perspectives in the NDN/IoT development area. We have developed this layer as an application in order to provide users with a simple way to finely customize it to fit their needs.

Many new wireless protocols for IoT rely on using packets with small payloads in order to provide communications with low energy consumption. Even if these IoT wireless protocols provide a tight bandwidth; they offer a very efficient power management, they are quite cheap and they provide nodes native sensing/actuating abilities. Thus, more effort must be made in order to integrate NDN with them.

As future work, we aim to improve the efficiency of the NDN-over-ZigBee layer by reducing time exchanges and by making it faster and more reliable. Besides the sensing management, we are currently developing an actuating support. Thus, symmetrically to the sensing feature, the emission process can receive specialized Interests that express commands towards actuators connected to its ZigBee hardware module; and rather than sending these Interests, the emission process will locally send ZigBee commands to modify the corresponding output pins.

Acknowledgments

This work is supported by 17MDU988 project which is a PHC (Partenariat Hubert Curien) Tassili, a French–Algerian cooperation program. This Tassili project is funded by the French ministries (MAEDI and MENESR) and the Algerian ministry MESRS.

References

[1] Y. Zhang, D. Raychadhuri, L. Grieco, E. Baccelli, J. Burke, R. Ravindran, G. Wang, ICN Based Architecture for Iot - Requirements and Challenges, Tech. Rep., ICN Research Group, 2014. http://tools.ietf.org/html/draft-zhang-iot-icn-challenges-01.

[2] L. Zhang, D. Estrin, J. Burke, NameD Data Networking NDN Project, Tech. Rep. NDN-0001, NDN, 2010. http://named-data.net/techreports.html.

[3] M. Mosko, I. Solis, E. Uzun, C. Wood, CCNx 1.0 protocol architecture, Tech. Rep., PARC, 2015. http://www.ccnx.org/pubs/CCNxProtocolArchitecture.pdf.

[4] B. Ahlgren, C. Dannewitz, C. Imbrenda, D. Kutscher, B. Ohlman, A survey of information-centric networking, IEEE Commun. Mag. 50 (7) (2012) 26–36.

[5] NDN Project Team, NDN Specification Documentation, Tech. Rep. 0.1a2, NDN, 2014. http://named-data.net/wp-content/uploads/2013/11/packetformat.pdf.

[6] W. Shang, Q. Ding, A. Marianantoni, J. Burke, L. Zhang, Securing building management systems using named data networking, IEEE Netw. J. 28 (3) (2014) 50–56.

[7] D. Saxena, V. Raychoudhury, N. SriMahathi, SmartHealth-NDNoT: Named data network of things for healthcare services, in: ACM Workshop on Pervasive Wireless Healthcare, 2015, pp. 45–50.

[8] M.A. Hail, S. Fischer, Flexible API for IoT services with named data networking, in: IEEE International Conference on Emerging Technologies and Innovative Business Practices for the Transformation of Societies (EmergiTech),2016, pp.1–6.

[9] W. Shang, A. Bannisy, T. Liangz, Z. Wangx, Y. Yu, A. Afanasyev, J. Thompsonx, J. Burkex, B. Zhangz, L. Zhang, Named data networking of things (Invited paper), in: The 1st IEEE Intl. Conf. on Internet-of-Things Design and Implementation, Berlin, Germany, 2016. https://named-data.net/publications/ndn-iotdi-2016/.

[10] Sigfox, Website Sigfox, [Online], http://www.sigfox.com/.

[11] LoRa Alliance, Website LoRa Alliance, [Online], https://www.lora-alliance.org/.

[12] Digi International Inc., XBee/XBee-PRO DigiMesh 2.4 OEM RF Modules, Tech. Rep., Digi International, 2008. http://www.digi.com/products/xbee-rf-solutions/modules/xbee-802-15-4.

[13] E. Baccelli, O. Hahm, M. Gunes, M. Wahlisch, T. Schmidt, RIOT OS: Towards an OS for the Internet of Things, in: The 32nd IEEE International Conference on Computer Communications, INFOCOM 2013, 2013.

[14] W. Shang, A. Afanasyev, L. Zhang, The design and implementation of the NDN protocol stack for RIOT-OS, Tech. Rep. NDN-0043, NDN, 2016. http://named-data.net/techreports.html.

[15] A. Attam, I. Moiseenkoy, NDNBlue: NDN over bluetooth, NDN-0015, 2013. http://named-data.net/techreports.html.

[16] IETF, Transmission of IPv6 Packets over IEEE 802.15.4 Networks [Online], https://tools.ietf.org/html/rfc4944.

[17] NDN Project Team, NFD Overview [Online], http://named-data.net/doc/NFD/current/overview.html.

[18] M. Amadeo, C. Campolo, A. Molinaro, Internet of Things via named data networking: The support of push traffic, in: International Conference and Workshop on the Network of the Future, NOF, 2014, pp. 1–5.

[19] Github, NDN client library with TLV wire format support in native Python, [Online], https://github.com/named-data/PyNDN2.

[20] NDN Project Team, get NFD connected, [Online], http://named-data.net/2015/01/06/get-nfd-connected/#more-2221.

Amar Abane is a 2nd year Ph.D. student in LARI Lab (University Mouloud MAMMERI of Tizi-Ouzou, Algeria) and CEDRIC Lab (Conservatoire National des Arts et Métiers, Paris). After an MS degree internship dealing with Named Data Networking for IoT, he was involved in the PHC Tassili project (17MDU988) around the topic: “A secured NDN architecture for IoT: example of Smart Farming". Currently, he focuses his activities on NDN and low rate wireless technologies for IoT. In parallel to its Ph.D., he is also a web and mobile developer.

Mehammed Daoui is Research Director at the LARI Laboratory, University Mouloud Mammeri of Tizi-Ouzou, Algeria. He is involved in several research areas related to networks, embedded systems and IoT.

Samia Bouzefrane is Associate Professor and has the accreditation to conduct research (HSR) at the Conservatoir National des Arts et metiers (CNAM) of Paris. She received her PhD in Computer Science from the University of Poitiers (France) in 1998. After four years at the University of Le Havre (France), she joined in 2002 the CEDRIC Lab of CNAM. She is the co-author of many books (Operating Systems, Smart Cards, and Identify Management Systems). Her current research areas cover security in cloud computing and internet of things, resource allocation in Cloud computing, and new paradigm for networking such as NFV and NDN.