Programming Support for a Delay-Tolerant Web of Things

Maël Auzias

THÈSE / UNIVERSITÉ DE BRETAGNE SUD

UFR Sciences et Sciences de l'Ingénieur

sous le sceau de l'Université Européenne de Bretagne

Pour obtenir le grade de :

DOCTEUR DE L'UNIVERSITÉ DE BRETAGNE SUD

Mention : Informatique

École Doctorale SICMA

présentée par

Maël Auzias

IRISA Institut de Recherche en Informatique et Systèmes Aléatoires

Programming Support for a Delay-Tolerant Web of Things

Thèse soutenue le 03-10-2017,
devant la commission d'examen composée de :

M. Frédéric GUINAND
Professeur des universités, Université Le Havre / Président

M. Nicolas MONTAVONT
Professeur des universités, IMT Atlantique / Rapporteur

M. Michaël MRISSA
Professeur des universités, Université de Pau et des Pays de l'Adour / Rapporteur

Mme. Pascale LAUNAY
Maître de Conférences, Université de Bretagne-Sud / Examinatrice

M. Yves MAHÉO
Maître de Conférences HDR, Université de Bretagne-Sud / Directeur de thèse

M. Frédéric RAIMBAULT
Maître de Conférences, Université de Bretagne-Sud / Encadrant de thèse

À Maryse

Acknowledgements

Acknowledgements below are for various persons that help me finishing this work, each by its own way. Since some are non-English speakers yet primarily concerned by these, there are written in appropriate language.

I thank a lot all my colleagues, especially Armel, Ali, Kader, Romain H., Pamela, Alan, Lucas B., Marc, Romain D., Mathieu, Lionel, Delphine, Raounak, Rymel and Nader, as well as my friends, Xavier, Matti, Aurélia, Audrey L., Victor, Margot, Lucas M., Milad, Lucas G., Carlos, Nano, Alba, Andreea M., Anna, AB, Luís, Elena, Daniele, and Andrea C., who help me by both pushing me to work over the clock but also by relaxing with agreeable moments shared altogether.

Merci aussi aux pouer-mouers qui ont aidé soit en bons moments via slack, soit en fournissant des pistes et conseils dans le domaine professionnel mais aussi pour Rust et OW.

Moc děkuju Míše a Peťe za společně strávené momenty, které mi pomohli se uvolnit, když jsem potřeboval. Taky moc děkuju Markétě a Magdaléně za jejich výbornou práci na Czech Vibes Sounds, která mi poskytla příjemné melodické prostředí, díky němuž jsem se mohl soustředit na vypracování mojí doktorantské práce.

Merci François, Marie-Claude, ke la don' a moin la chance woir zot ti péï, resours' à moin. Merci Mylène aussi, sem' moin, tou lé jours pou avancer.

Merci à ma famille pour le soutien, les encouragements et la motivation que vous avez sû me donner, notamment Maryse par le biais de ces mots rapportés.

Merci à Vincent Creuze et Jean-Paul Rigault pour leur lettre de recommandation ainsi que les discussions que nous avons eu et qui m'ont remotivé.

Merci enfin à Yves et Frédérique, mon directeur de thèse et mon co-encadrant, pour m'avoir guider et aider tout au long de ce travail, ainsi qu'à Mario, Sylviane et Anne sans qui le labo ne pourrait pas fonctionner.

Contents

List of Figures

List of Tables

1 Introduction

Contents

1.1 Context

The Internet of Things (IoT) is an emerging paradigm aiming to enhance our everyday life through automation and optimization tasks related to transport logistic, healthcare domain, smart-home and smart-enterprise, and so forth. According to Gartner, in 2020, there will be 26 billions of things for a market exceeding $300 billion. This new market has been invested by big companies, such as Samsung, Amazon or Google, that are developing *things* for the IoT and try to stay ahead of their concurrents by proposing emerging and innovative products. For instance, a message could be displayed on an exit door suggesting to take an umbrella since the weather station, located at the end of the street, sent an alert to it. This same umbrella could vibrate, ring or send a message to the owner's smartphone when he leaves without it.

The things are everyday objects equipped with a computer, usually an inexpensive single board, a small battery and a short-range wireless communication means. They are expected to be cheap and their power of action lies in their large number and high diversity to compensate their constrained battery and power-processing. The concept of *thing* can also be extended to more powerful and far more expensive objects such as smartphones, drones or cars.

The IoT becomes more and more ubiquitous in our environment yet giving a precise definition of IoT is a challenge since different visions coexist and research points of views vary widely. These visions categorize the IoT paradigm into different concepts at different levels. In [1], Atzori et al. illustrate the IoT as a convergence of three different visions focused: *1.* on the things themselves, *2.* on the Internet aspect, and *3.* on the semantic aspect. Another possible categorization is presented in [2], within which Gubbi et al. propose an application-scattered vision that goes from personal user to national utility involving other applications such as medical treatments, home automation, transport logistic, community service.

1.1.1 Intermittently Connected IoT

The things are often equipped with short-range radio interfaces, undergo energy-saving constraints that shutdown their radio interfaces, can be mobile (e.g., when embedded on robots or carried by humans) or can be deployed in area lacking of network infrastructure (e.g., in developing countries or in sparsely populated area as Laponian in Finland [3]). Under those challenging circumstances, it is clear that the things suffer from disconnections and difficulties to access a reliable network. Also, despite being aware of these networking challenges, current studies on IoT consider the IoT as an interconnected-extension of Internet. In the meantime, there is a growing support for the claim that things are not able to connect to Internet in many context. This thesis targets these contexts in which frequent disconnections are the norm rather than the exception. These targeted networks are known as Intermittently Connected Networks (ICN). As a postulate for this study, because of the lack of reliable communication-means, things communicate through hop-by-hop exchanges.

For example, such contexts can be illustrated by a town hosting sensors and actuators in and out the city center, possibly far away. These sensors would gather data on the environment (e.g., counting the number of cars passing through a tunnel or a bridge, measuring air-pollution) and actuators would cooperate to enhance environment quality (e.g., modify speed limit on road signs equipped with screens, close or open roads to move or lessen traffic jam). Because of the infrastructure costs (e.g., deployment, maintenance, individual things subscription-plans) and limitations (e.g., gray or white areas coverage is even overpriced when possible) this network is deployed without relying on any infrastructure.

1.1.2 Communication in Intermittently Connected IoT

For several years, great effort has been devoted to the study of communication challenges in ICN. A research group of the Internet Research Task Force (IRTF), the DT-NRG (Delay-Tolerant Networking Research group), proposed an architecture as well as a protocol to exchange messages between nodes where no infrastructure is present. This architecture, the Delay-Tolerant Networking (DTN) architecture [4], was historically needed for Inter-Planetary Networks (IPN). It defines an infrastructure-less communication system and is a substitute when Internet Protocols cannot be relied upon. It provides communication means without assuming a connected path between a message's source and its destination. It supports a wide naming-syntax to identify nodes without relying on IP addresses, and allows DTN nodes to exchange variable-length messages. To do so, these messages are transmitted, possibly over multiple paths, according to the store-carry-and-forward mechanism [5].

The store-carry-and-forward mechanism allows to exchange messages between two nodes that never meet and without any continuous end-to-end path by exploiting node storage and through hop-by-hop forwarding between intermediary nodes. A message is stored temporarily on a node while the node is moving. This message is thus carried by its mobile host that will take advantage of contact opportunities by forwarding it to other devices when possible so the message eventually reaches its destination. In traditional networks, all along a packet route, from its source to its destination, there is only one single copy of this packet at all time. Each time the packet is forwarded, it is deleted from the previous node. On the contrary, for efficiency and reliability reasons, DTN nodes usually forward several copies of each message they store. As a result, it can be said that a message is *disseminated* in a delay-tolerant network.

Such system could be deployed in the previous example illustrating the city equipped with sensors and actuators. Messages between the things would be exchanged thanks to mobile sensor and actuators. DTN is designed to work well when, in this case, things are mobile. It can also be useful when things are actually fixed but rely upon data-mules (buses, tramways or even metros equipped with specific devices) that literally move messages from devices to devices. These data-mules can also be considered as things.

One specific protocol defines how to process such messages: Bundle Protocol (BP) [6]. BP is the *de facto* standard protocol over the DTN architecture to exchange messages. These messages, called bundles, are composed of one primary block enclosing bundle metadata, zero or more Meta-Extension Block (MEB [7]) that can trigger specific processing on the carrier, and one, or more, payload blocks. Bundles copies are disseminated in the network according to the store-carry-and-forward mechanism. The decision of a BP node to forward a copy of a bundle and to which intermediary node, results from a routing algorithm. BP RFC does not specify any routing algorithm. However, such algorithm or protocol is required to determine best intermediary nodes to which forward bundles. Many routing algorithms, proposed by the research community, exist nonetheless: epidemic, statistically oriented, location focused, socially based and so forth. BP implementations need to be completed with such algorithm and they usually embed several algorithms by default.

In the remaining, "DTN" stands for the architecture itself, whereas "DTNs" represents networks following the DTN architecture, and DT-IoT means Delay-Tolerant Internet of Things. The DT-IoT represents IoT paradigm with a DTN approach at the networking level.

1.1.3 Programming Support in Intermittently Connected IoT

BP-based platforms can provide a solution for the connectivity challenge identified for the DT-IoT. However, DTN and BP only solve the connectivity challenge and do not provide any support to easily program things cooperation. BP can be likened to a unreliable transport protocol for DTNs. A middleware is needed on top of a BP-based platform.

Middleware systems for the IoT are more and more inspired by the Web [8]. Indeed, things are usually considered as client or server. Clients request servers through regular HTTP methods or through communication means relying on protocols similar to HTTP. Then, servers answer with responses embedding regular HTTP code response. This architecture and programming style are directly inherited from the traditional Web. Traditional Web is to the Internet what the Web of Things (WoT) is to the IoT. In this thesis, the term WoT is then used to qualify middleware systems that follow traditional Web principles. In many contexts and applications the traditional Web is lightweight in terms of network usage, RAM and CPU, scalable and easy to apprehend and comprehend. For these reasons, it makes sense to reuse Web technologies for the IoT.

The REST architecture [9] allows to go further than just Web principles. The REST architecture is a set of several constraints: client/server architecture, stateless server, cacheable responses, layered design, uniform interface and, optionally, code-on-demand. It enforces scalability, interoperability, portability, performance and simplicity of interactions between RESTful nodes. All these properties greatly benefit to a WoT context, thus this thesis' contributions are explicitly investigated to be RESTful.

Resource-Oriented Approach

RESTful WoT can be categorized in two different approaches. The first approach is resource-oriented. In this approach the things host resources that are, according to W3C [10], considered as an "item of interest, in an information space, identified by a URI".

Several protocols and implementations encompass this approach. One stands out as an emerging, Web-oriented and RESTful protocol: Constrained Application Protocol (CoAP [11]). CoAP is a request/response-based protocol that allows resource-constrained devices, such as things, to interact together asynchronously. Relying on UDP, a retransmission mechanism is defined in order to provide reliable exchange. Requests and responses respectively contain HTTP methods and HTTP statuses, with some slight semantic differences on HTTP statuses.

The context of this thesis is focused on the DT-IoT. Also, since Web-oriented systems are investigated, the context can be narrowed down to the Delay-Tolerant Web of Things (DT-WoT). Besides, if CoAP seems to be a great candidate for a DT-WoT programming support, it cannot be used as is. Indeed, since CoAP has been designed with traditional network in mind, some of its features are unfit for DTNs. For instance, CoAP, relying on UDP, uses IP addresses and port as communication-entrypoints whereas DTNs may not rely on IP at all. Moreover, its congestion control and retransmission mechanisms are not suitable as is for DTNs. A work on these unfitting features to transpose CoAP into a DTN context is required.

Service-Oriented Approach

Resource-oriented systems provide little means to describe, identify and discover the resources, especially in DTN context were no node can act as a centralized registry. The lack of such mechanisms is inherently fulfilled by the Service-Oriented Approach. Indeed, this approach is composed of a set of rules defining the description of services, their advertisement/discovery and their invocation.

In addition to the resource-oriented approach, the service-oriented approach is also considered in this thesis, both being compatible with REST. This approach relies on and encompasses what is usually called service-oriented architecture, service-oriented networking and service-oriented computing. Generally speaking, "SOA" stands for Service-Oriented Architecture, but in the remaining of this thesis SOA stands for Service-Oriented Approach.

Going from resource-oriented solutions to SOA solutions offers more means and possibilities. Service description, service advertisement and service discovery are ones of the main concerns in SOA platforms. SOA solutions are constituted of three different entities: registry, clients and service providers. Service providers describe their own service functionalities and capacities in a document called a service descriptor. This descriptor is published into the registry. Clients find service providers thanks to registry lookups and can then invoke service providers directly.

Thanks to their loosely coupled and late binding properties, SOA systems are adapted for DTNs. Indeed, in such environment no node can be relied upon as a registry, so the discovery challenge needs to be overcome. Furthermore, DTN nodes undergo specific networking constraints that must be taken into consideration in the service description. For instance, it would be very useful that a service informs when it is available (i.e., when is its host awake), until when it is expected to be available and in which location it can be requested. These details are not often meaningful in traditional networks and are thus left out. CoAP exploits a discovery and description mechanism [12], but it is too limited to suit DTNs: it relies on a direct exchange between nodes and resource descriptions do not provide all needed details in DTN environment. Very few works in the DTN literature address SOA even though it seems really pertinent to enable service-oriented systems in these networks.

1.2 Proposal

The objective of this thesis can be summed up as: *how to provide a programming support for the* DT-W*o*T*?* First a resource-oriented solution is sought. Indeed, since it is a common way to solve the programming challenge within the IoT, it makes sense to follow this practice that was already proven as lightweight and scalable in traditional networks. Second, a service-oriented solution is sought. It aims to provide means for discovery and resource/service description that lack in the resource-oriented approach.

Contributions of this thesis are explicitly investigated to be as close as possible to existing standards (RFC and architectures). In fact, standards are well known, reliable, tested and approved. They also ease programmers implementation and developing tasks as they already understand the standards.

To address the two problems stated above, the following contributions are proposed and presented in this thesis:

- The first contribution is the proposition of a resource-oriented programming support to enable a DT-WoT. It consists in a transposition of CoAP through fundamental adjustments and enhancements to use BP as binding. These modifications and adaptations are implemented into a prototype named BoaP. Most CoAP features are respectfully transposed into BoaP and are compliant with RFC 7252 (e.g., serialization, CON/NON messages, deduplication, and so forth).

- An original method to evaluate middleware systems in DTNs is proposed. This method relies on an emulation platform, provided by Swarm/Docker, exploiting two different kinds of node mobility: an artificial one and a reality-based one. Accurately evaluating protocols and implementations is a challenge in DTNs. Only coarse-grained simulations are usually done since node mobility, scenarios, radio-range, radio-interferences, node-networking behaviors are approximated. The current literature shows no consensus on experimenting methods and approximations. Only real world experiments could provide precise data but very few are done due to their cost. In this thesis, implementations running on nodes were not simulated. Indeed, to assess BoaP performance it is first deployed in a small physical network. Afterwards, an emulation system is used, relying on Swarm/Docker. This architecture is adapted to DTN context: it emulates network nodes while simulating network links. Experiment results are closer to real-world deployment than simulation-based results. These experiments ascertained that BoaP is reliable enough and could even be used to support a RESTful and service-oriented middleware for DT-WoT.

- An SOA and RESTful programming support built with IoT in mind and adapted to DTN environments comes as the last contribution. This service-oriented middleware lies on top of BoaP and is named LILWENE[1]. The specifications of this middleware define a service descriptor as short as possible yet fully enclosing service details and providing fields for future usage. These descriptors are advertised through a publish/subscribe interface into topics supported by BP endpoint identifications. The registry architecture is directory-less and each node participates in the advertisement and discovery. Each node hosts a local registry that gathers service descriptors. These clients can then discover service providers and invoke them. A service invocation, supported by BoaP, introduces a couple of new options compared to BoaP. An API and an implementation elements are also described.

1.3 Thesis Outline

The remainder of the thesis is organized as follows.

Chapter 2 reviews DTN architecture, BP protocol and introduces opportunistic networking. Then both resource and service oriented approaches and protocols for both IoT and DT-IoT are presented.

Chapter 3 presents the BP and CoAP protocols and implementations at a technical level. Then, CoAP modifications and adaptations, composed of fundamental adjustments and enhancements, are proposed to provide a Delay-Tolerant CoAP. Next, a prototype, stemming from these adaptations, named BoaP is described. A comparison between CoAP and BoaP features, that are compliant for most of them, concludes this chapter. In addition, this chapter also presents an interface with a BP implementation (IBR-DTN).

Chapter 4 introduces BoaP evaluations. First, BoaP is deployed in a small physical network and then in a simulated network with emulated nodes in four different experiments: three using an artificial model for the nodes mobility and the last one using reality-based traces. To assess BoaP performances a new emulator system relying on Swarm/Docker is proposed to provide an emulation environment.

Chapter 5 contains the proposition of the RESTful and service oriented middleware support named LILWENE[1]. It details service discovery and advertisement, and service invocation means coupled with an API and some implementation elements.

Chapter 6 concludes the document and broadens next challenges of this work.

[1]LILWENE: Light services middleware for delay tolerant Web-of-things Networks

2
State of the Art

Contents

Studies in relation with the contributions of this thesis are reviewed below. First, challenged networks are defined. The challenged networks are categorized in two visions: Delay-Tolerant Networking (DTN) and Opportunistic Networking. In this section, the DTN architecture as well as the bundle protocol (BP) are explained in details since the contributions of this thesis rely on those. The opportunistic networking paradigm is presented in a broader manner with a focus on routing algorithms since most of the research works concern such algorithms.

Afterwards, the different approaches of programming supports for the Internet of Things (IoT) or the Web of Things (WoT) are presented and are categorized according to their approach that is either resource-oriented or service-oriented. Since this categorization is coarse-grained, additional category is also considered and named 'other'. In this section, CoAP is detailed in depth for the same reasons DTN and BP are.

Finally, in like manners that IoT/WoT programming supports are reviewed, works on programming supports in DTN context are presented. They are categorized as pragmatic programming and programming paradigms.

2.1 Challenged Networks

Traditional Internet protocols rely on implicit assumptions such as continuous bi-directional end-to-end path between any pair of nodes, low error rates and low message loss, a single route for network packets is enough to reach acceptable communication performance, and short and roughly symmetric round-trip delays. In 2003 [5], Fall identified as *challenged networks* any network in which one of these assumptions cannot be met. To work around these challenges networks nodes communicate through hop-by-hop exchanges according to the store-carry-and-forward mechanism. When a node emit a message, it temporarily stores the message and carries it while moving. When it encounters another node it forwards a copy of this message. If the encountered node was the message's destination then the message is considered as delivered, otherwise the node is just an intermediary node and will forward copies of the message too.

Challenged networks has been targeted by many studies and projects. These works usually identified themselves as being part of either the disruption/delay-tolerant networks (DTNs) research field or the Opportunistic Networks (OppNets) research field. Historically, DTN approaches were the first to target challenged networks and more specifically Interplanetary Networks (IPN). However, the DTN architecture and the Bundle Protocol were not a good fit for all the terrestrial scenarios. In order to support communication in these scenarios another approach emerged that is identified as Opportunistic Networking. As presented by Pelusi et al. in [13], the exact difference between opportunistic and DTNs approaches is not clear and is generally just a vocabulary choice. In this document, DTN approaches are networks and applications relaying on DTN (RFC 4838 [4]) and BP (RFC 5050 [6]) opposed to OppNet approaches that frequently present the studied networks as Disconnected Mobile Ad Hoc Networks.

2.1.1 Delay-Tolerant Networking

Earliest studies on challenged networks are related to the Delay-Tolerant Networking architecture (DTN) stemming from the historic need to provide communication for challenged networks that were IPN. Due to very long distances, even at the speed of light, the latency is more than 1 second for Earth-to-Moon distance and varies between 4 and 24 minutes for Earth-to-Mars distance. IPN suffer from long and variable delays and disconnections because of their continuous movement and all celestial objects disrupting their network links that require a line-of-sight.

In May 2001, in order to support deep space exploration communication between Earth and the different satellites and space probes, an architecture for the IPN is defined by Vint Cerf et al. in [14]. The architecture relies on a store-and-forward overlay network on top of a transport layer. In 2003, Fall proposed the term Delay-Tolerant Networking (DTN) to describe and encompass all types of long-delay, disconnected, disrupted or intermittently connected networks, where unreliable connectivity may be experienced [5]. Finally, in 2007, two important RFC are written: one, written by Vint Cerf et al., describes the DTN architecture itself [6] and the other one, written by Keith Scott et al., defines a protocol to exchange messages in a DTN overlay [4].

The DTN architecture (RFC 4838, [4]) aims to provide architectural guidelines for protocols in challenging networks according to the following design principles:

- Messages vary in length and can be long. Stream and small packets shall be avoided.
- Node identification syntax must enable a wide range of addressing conventions to better support interoperability.
- Node storage are at disposition to provide needed resources for the store-carry-and-forward mechanism.
- Security means shall protect the network from undesirable traffic.
- The means to help the network to better deliver data according to the applications' needs shall be available (such as classes of service, lifetime of messages and delivery options).

The Bundle Protocol (BP, RFC 5050) [6] is the *de facto* standard for the bundle-layer of the DTN architecture [4]. The BP forms a message-based overlay that follows the store-carry-and-forward principle. The BP defines the format of the messages, called bundles, and the logic layout to process them.

As a network overlay, the BP relies on convergence layers that abstract subnet-specific protocols to transport bundles (e.g., TCP, UDP, LTP). Bundles have a lifetime and will be deleted if it expires. In order to overcome network disruptions and high delays, the BP uses a cache, on nodes, to store bundles. These bundles are either processed by an application (if the destination is on the node), or forwarded to other nodes towards the bundle destination. A BP endpoint is identified by an Endpoint Identification (EID) that takes the form of a URI. A BP endpoint can either be a singleton or a set of BP nodes that register themselves with a common EID, thus allowing multicast-like operations to be performed.

The BP bundles have to be routed from node to node. Despite being required to determine to which node forward a bundle, the BP specification does not fix a routing method. Nonetheless, many routing algorithms exist, each of them intended to be adapted to a networking context (e.g., the mobility of the nodes) or to a type of application. A key characteristic of a routing algorithm is its choice to allow multiple copies of a bundle in the network (e.g., as in the epidemic approach). Since the beginning of the research for these networks routing algorithms has drawn much attention: as early as 2006 a survey was done on routing protocols in DTNs [15].

Bundles are constituted of one primary block (header), then zero or more extension blocks called called Metadata Extension Blocks (MEB) [7], and one or more payload blocks. The primary block carries options that influence the treatment performed by the nodes that forward and receive the bundle. For example, a Report-When-Bundle-Delivered option will make the destination node emit an administrative bundle when receiving the bundle. Extension blocks can be used to make specific processing decisions regarding bundles, e.g., routing decisions.

The bundles have a TTL and will be deleted from node's storage when it expires. The optimized TTL value is, at my best knowledge, not defined. In fact, the perfect value differs according to the different kind networks. Some networks may work with lower delivery ratio, while other would require a higher, or even highest, delivery ratio. These delivery ratio are related to the TTL value: with a very short TTL the bundles are likely to die before reaching their destination, on the contrary with an infinite TTL the bundle will reach its destination before dying. These previous extreme values have another important side effect. Indeed, the network storage usage is directly impacted by the TTL value. With a very short value fewer nodes will store the bundles (and not for long), so their storage shall not be saturated. In contrast, with a very long TTL, a lot of nodes will have to store a lot of bundles, hence saturating their storage.

The BP does not offer a reliable means of communication since it relies on intercontact to forward messages and nodes' mobility. Nevertheless, a built-in mechanism, named "custody transfer", aims to enhance reliability. The custody transfer requests that a BP node takes the responsibility for delivering a bundle to its destination. The responsibility is released when the node forwards the bundle to another node accepting this responsibility.

Several BP implementations exist. Interplanetary Overlay Network (ION [16]) or Postellation [17], for instance, are designed for spacecraft flight software whereas IBR-DTN [18] is more suitable for constrained nodes with a small RAM usage in comparison with DTN2 [19] that was one of the first BP implementation. IBR-DTN is one of the most up-to-date implementation with an active community. Also, because it is friendly with resource constrained nodes, thanks to its RAM usage, it is the implementation chosen for this thesis since the IoT, composed of resource-constrained nodes, is at the core of this thesis's context.

The IBR-DTN implementation runs a daemon that stores, carries and forwards bundles to other DTN nodes in the network. Added to this daemon a couple of command line tools are provided by the Linux package that offer to send and receive messages, to synchronize folders between different hosts, and so forth. The daemon and the tools communicate via an API. The entrypoint of this API is a TCP socket.

This way of implementing the BP in a separate daemon process (accessible via a TCP socket or another inter-process communication facility) is a versatile system: the BP daemon can be detached from a sensor node and placed on a more powerful node if the sensor is too constrained. On the down side, it induces an additional delay. It should also be noted that IBR-DTN is not the only implementation that follows this system, suggested in [6], DTN2 does too.

In this thesis, DTN is considered as mature and complete. Indeed, despite relying on *unfinished business*, such as IPND [20][1], the neighbor discovery of DTN, and the very limited of real-world deployments, the DTNRG (Delay-Tolerant Networking Research group), which proposed the architecture, has been wound down in April 2016[2]. Also, for this reason, no contribution are sought on the DTN, per say, that is at the base of all contributions.

2.1.2 Opportunistic Networking

The opportunistic networking approach stems from both DTN approach and Mobile *ad hoc* Networking (MANET) approach. Mobile *ad hoc* Networks (MANETs) do not cover all challenges and generally consider node density as high enough to mask away the node mobility by constructing "stable", connected end-to-end paths, as in the wired Internet. The approach of opportunistic networking does not assume that an end-to-end connected path exists between a pair of nodes and does not consider the node mobility as a problem but rather as an exploitable opportunity to link disrupted paths. These networks are either identified as Intermittently Connected MANETs (ICMANETs) or Disconnected MANETs (DMANETs) that rely on the same store-carry-and-forward mechanism from DTN approach. Thus, when a node does not have a good next hop to forward the data, it simply stores the data locally without discarding it, as would occur in a MANET, and waits for an opportunity to forward it. Nevertheless, the Bundle Protocol is not a good match for all the various terrestrial scenarios.

Most part of OppNet studies are focused on routing algorithms. Routing algorithms in OppNet aim to two different objectives that are: maximize the probability of message delivery while minimizing the delay each message experiences during delivery. The probability of message delivery, i.e. the delivery ratio, is the ratio of received messages to emitted messages.

2.1.2.1 Forwarding Algorithms

The literature generally refers to forwarding algorithms rather than routing algorithm. This is due by the fact that no *route* is assumed to exist between any pair of nodes, hence algorithms determine which node is the best *forwarding target* opposed to routing algorithms that determine the shortest path to reach a packet's destination.

These algorithms are not only for OppNet but also DTNs. It is common for this kind of study to interchange the words 'OppNet' and 'DTN'. Several approaches have been adopted to achieve reliable communication in such networks. These various approaches are detailed below and classified as either deterministic,

[1]Mentioned as such in last DTNRG meeting at IETF-87 while DTNRG was closing down.
[2]See the mail at: https://archive.fo/08Hhm

content-based, stochastic, context-aware (where contexts are approximated by details extracted from location-based or social-based data) and mobility-based.

Deterministic Forwarding

Deterministic algorithms are a great fit with controlled environments or when the contacts are scheduled or predictable.

One algorithm that does not focus on any controlled environment but remains deterministic is the Epidemic Routing protocol proposed by Vahdat and Becker [21]. The simplistic forwarding algorithm is to forward a copy of every message to all its neighbors (except the one who sends it). If this protocol offers the best delivery ratio, it also consumes a lot of nodes storage and medium usage leading to high energy consumption and poor scalability. These downsides are studied by Tseng et al. in [22] who conclude that it can seriously degrade the network performance, especially in high-density regions or if the resources are scarce.

Spyropoulous et al. [23] proposed two single-copy based protocols: Direct Transmission and Randomized Routing Algorithm. With Direct Transmission, a node forwards a message only if it is in contact the destination. In Randomized Routing Algorithm, a message is forwarded based on a probability score. These protocols, unlike Epidemic Routing, produce very low network traffic but are convenient only if the node mobility is well known or highly predictable.

Time To Return (TTR) [24] is another forwarding algorithm that targets environment with predictable mobility. The context targeted by this work is rescue in emergency scenarios. The protocol takes advantage from the fact that all medical personnel in such context are coordinated by a leader that assigns actions and maximum time to return to the base for security reasons.

Content-Based Forwarding

Another class of forwarding protocol is based on the content messages contain and profile of interest of network nodes.

SocialCast [25] is a content-based forwarding algorithm providing a publish/subscribe interface. A publisher originally delivers a fixed number of copies of the message to carrier nodes. A message carrier will deliver a copy of the message to subscribers it meets or will delegate the message to another node that is selected as a more feasible carrier.

DODWAN [26] follows a different approach of content-based that relies on a publish/subscribe interface. Each node periodically informs its neighbors about the messages it is carrying and that match neighbors interest profiles. When a neighbor misses a message carried by another host, then the neighbor requests a copy of this message.

Stochastic Forwarding

A couple of studies aimed to limit the number of messages generated by the Epidemic Routing protocol by adopting a stochastic approach. This is the case of Spray-and-Wait [27] proposed by Spyrpoulos et al.. It is inspired from flooding-based forwarding algorithms since it makes no use of information on network topology or knowledge of the past encounters of nodes, however, it significantly reduces the transmission overhead by limiting the total number of copies that can be transmitted per single message.

The same authors proposed a follow-up of Spray-and-Wait: Spray-and-focus [28]. Its difference with the Spray-and-Wait algorithm lies in the Focus phase. Unlike Spray-and-Wait, a node, after forwarding all the allowed copies of a message, can delegate the message it carries to another node according to a given forwarding criterion.

Context-Aware Forwarding

There are also forwarding algorithms that base their forwarding decision on data extracted from their context. These contexts are location- (or movement-) based and social-based. It should be noted that the *social* interactions of network nodes differ from the *social* interactions of their carriers. From a networking point of view, a social interaction is represented by an intercontact of two networks nodes while the carriers of these network nodes may not communicate at all.

Location-Based Several location-based forwarding algorithms exist. MaxProp [29] is designed for vehicle-based networks. These networks are usually composed of network nodes with large storage capacity and energy source, but experience short contact duration. To overcome these short contact durations, MaxProp proposes a prioritization of packets to be forwarded, or dropped according to their priority.

In MobySpace [30], a node computes the closeness of its neighbors to the destination of a message it wants to forward. The closeness of two nodes is computed according to the probability of an intercontact to occur between these two nodes.

To overcome the moderate delivery ratio of TTR [24], due to the one-copy approach, the TTR algorithm has been modified in order to propose PropTTR and PropNTTR [31]. PropTTR uses MaxProp for the forwarding to direct neighbors and then the neighbors use TTR for the delivery. PropNTTR follows the same rules but uses MaxProp for neighbors and neighbors of neighbors and so forth until the hop count of the message is equal to N.

Social-Based PROPHET [32] is a social-based forwarding algorithm. PROPHET assumes that node movement is not random and that it is possible to identify mobility patterns. Thus, its forwarding decisions are taken according to delivery probabilities that are estimated from the frequency of encounters, i.e. the probability to deliver a message to a certain destination node increases whenever it comes within sight, and decreases overtime in case no meeting occurs.

Following a similar approach, HiBOp [33, 34] relies on the history of contact and the place where two nodes last met each other. An algorithm [35], by Vu et al., relies on encounters patterns between network nodes. After studying on real traces of Wi-Fi and Bluetooth traces over 6 months the authors concluded that the contact patterns are regular and predictable. TAO [36] also extracts details from contacts between mobile nodes and infostations since it targets hybrid networks.

Eiko et al. propose a forwarding algorithm [37] electing specific node as message broker. Mobile nodes run a community detection algorithm in order to determine which node has the shortest path to all other nodes in the community. Then, this node acts as the message broker within its community. A similar approach is proposed for SimBet [38] where two metrics are used: one to find best nodes to connect between communities and one to find best nodes to deliver messages within communities. BubbleRap [39] algorithm relies on communities. Specifically, authors propose a method to detect communities, then when a node has a message, it forwards it to a node closer of the destination's community.

Mobility-Based Forwarding

Finally, there are works on forwarding algorithms focusing on specific mobility. In [40], Goodman et al. propose to deploy infostations with strong radio signal in an area to provide one-hop forwarding. Opposed to this approach, of fixed network node, data-MULE system [41] and the message-ferrying approach [42] exploit mobile nodes that follow a determined route to provide a reliable link opportunity.

2.1.2.2 Communication Middleware

Despite a majority of studies on forwarding algorithms, there are also projects and works that propose middleware systems in order to ease application programming. A common addressing and routing schemes shared by middleware systems in DTNs and OppNet are content-based: the data being moved in the networks are forwarded and shared according to their content. In order words, the network enables its nodes to share content based on their interest.

Haggle [43] is a content-sharing system involving mobile users. Each node describes its interests and lists the content items it locally stores. When two nodes meet, they exchange their interests and then share items of content that match other interests.

C3PO framework [44] provides *ad hoc* communication means to smartphones relying on legacy Wi-Fi as well as Bluetooth to create an opportunistic network. This communication system aims to be deployed for specific social events (e.g., shows, sports, concert and so forth) to share content among people without Internet access.

DODWAN [45] middleware proposes cooperation of neighboring devices to deploy component-based applications. Each devices hosts a *deployment manager* that can retrieve *software components* (applications) from other nodes while sharing its own software components. The middleware also provides communication means adapted to OppNet and exploits multi-hop exchanges.

A middleware architecture is proposed in [46] by Helgason et al.. It supports the dissemination of contents within a wireless *ad hoc* domain and with the wired Internet. The access to service of their platform is done through a publish/subscribe interface. This architecture has been implemented and tested using their Android implementation. Finally, for a large scale evaluation, they simulated their system on OMNET++.

CAMEO, proposed in [47] and part of the SCAMPI project [48] (Service Platform for Social Aware Mobile and Pervasive Computing), is a context-aware middleware that collects and manages context information from physical and virtual worlds. Thanks to these data, the system determines user's profile and social behavior that applications can exploit. This middleware aims to ease the development of social applications in OppNet.

OppNETing is not a very good candidate to substitute to DTN since routing algorithms as well as middleware systems are based on many hypothesis. These hypothesis are made on various parameters that include node mobility, node density, links dynamicity, node knowledge (e.g., on its geolocation), storage capacity and so on. No similar hypothesis are made in this thesis. Furthermore, middleware systems are rarely based on standard protocols unlike contributions of this thesis. Here again, this work does not aim to contribute on OppNet, per say. Nevertheless, the overall solutions (BoaP and LILWENE) could be deployed in OppNet.

2.2 IoT Programming

The IoT interests various actors and workers such as researchers or companies, that all work to help the IoT becoming a reality. A considerable part of these works are focused on middleware in order to allows developers to build application for the IoT. As shown by several reviews [49, 50, 51, 52], many projects and studies on middleware adapted to the IoT exist. Several architectures and designs exist for middleware systems and these works categorize these systems in different approaches. The service oriented approach is a common category to all these works. Other categories include data-based (*database oriented*, or *data focused*), agent-based and actor-focused, event-based or even object-oriented. Another categorization is proposed in this thesis with only two approaches: resource-oriented approach (ROA) and service-oriented approach (SOA).

These two approaches are supported by various works. ROA has already been studied in the context of IoT [53, 54] and SOA too [55, 56, 57, 58]. Differentiating one approach from the other is not straightforward. Indeed, they are close to one another and are sometimes mistaken for one another. Both approaches involve clients that request either resources (ROA) or services (SOA). To allow these clients to request resources or services, each client needs to discover them beforehand. This is done thanks to a discovery mechanism. The first difference between ROA and SOA lies in this mechanism. In ROA systems, discovery is rather basic, simple and not included by default. On the contrary with SOA, discovery mechanism is a requirement that is carefully described and defined. It usually provides searching and filtering capabilities. The discovery relies on descriptors that contain information to request, or to invoke, a resource, or a service. In ROA, these descriptors commonly contain a very limited set of information that are elementary. Such information is for example their identification and location, with an optional field to inform what type of data the resource can return. On the opposite, service descriptors provide all functional and non-functional details of a service. In some cases, several message exchanges are required to gather all details of one service. Finally, ROA and SOA differ a lot on the invocation aspect. With SOA, the sets of invocation methods for each service can widely vary whereas ROA systems frequently exploit the four main HTTP methods (GET, POST, PUT and DELETE) for all resources. From a higher point of view, SOA can be seen as an abstraction layer put over an ROA system. As for ROA system, the Web is a well-known example: web page are the resources and web browsers are the clients.

In practice, according to Guinard et al. [8], service-oriented approaches are usually too heavy and complex for simple objects that populate the IoT. This is a challenge that this thesis aim to overcome by investigating the SOA contribution towards RESTful principles. Indeed, both ROA and SOA commonly follows a RESTful style that is considered as lighter than other styles. As presented in [59] by Pautoasse et al., there are two styles of services: "big" Web services and RESTful web services. Pautasso et al. quantitatively compare these two services styles in [59]. The article points out strengths and weakness of both styles. One of the strength for RESTful web services is its low hardware requirements that allows a simple system for a low cost. As stated before, this low cost characteristic steps out as an important aspect in this thesis since it is a DT-WoT context where things have constrained resources.

2.2.1 Resource-Oriented Approach

Works identifying themselves as resource-oriented in the IoT usually follow a RESTful approach. In [53], Guinard et al. propose to design the IoT with Web technologies (HTML, HTTP), suggest to follow best-practice based on REST and present several prototypes that would extend the World Wide Web according to their resource oriented architecture. REST defines five constraints that are: *1.* a client-server architectural style to enforce a loose coupling hence enabling independent evolution for both sides, *2.* stateless servers to ensure server scalability, *3.* a cache to improve network efficiency as well as client satisfaction since cached response are instantly available, *4.* a uniform interface to simplify the software architecture and the visibility of interactions, *5.* a layered-system for all its well-known advantages. One optional constraint is added to these five ones: the code-on-demand. This constraint enables clients to download code, e.g. applet or script. This reduces the number of required features by clients that can be extended after deployment. It can be noted that REST defines four interface constraints being: *1.* identification of resources: each resource should be identifiable and conceptually separated from the representation of the resource, *2.* manipulation of resources through representations: each resource should be completed by metadata providing required details to process the resource, *3.* self-descriptive messages: each message should inform on the representation format of the resource (MIME), *4.* and, hypermedia as the engine of application state (HATEOAS): paths to related resources and/or actions of a resource being requested should be available by following hyperlinks, this can be likened to the CoAP URI path *well-known*, that follows RFC 5785 [60] .

Three prototypes adopting these constraints are then presented. The first one is a smart gateway. Smart gateways are a common solution to fill the gap between traditional Internet and WSN or other networks composed of component not able to connect to the Internet. The second prototype is a proxy to transparently translate regular HTTP requests/responses from Internet into proprietary protocols to a sensor and actuator network. This proxy also keeps in a cache the responses in order to minimize the number of exchanges with sensors and actuators. This proxy also offers smarter features such as monitoring (a client subscribe to a resource and receives notifications when its states reach a threshold). Taking benefit of the previous prototype, the last is a Web dashboard to remotely monitor and control energy consumption. In [54], Dar et al. oppose ROA to SOA in the context of IoT. This study adopts the ROA and proposes an architecture inspired by REST to interconnect business processes with IoT resources.

CoAP

At the moment, one standard protocol can be seen as the archetype of ROA in IoT: CoAP. CoAP [11] offers an application layer protocol that allows resource-constrained devices to interact together asynchronously. It is designed for machine-to-machine use cases and is compliant with the REST architecture style. CoAP defines a simple messaging layer, with a compact format, that runs over UDP (or DTLS when security is enabled). Its low header overhead and low complexity simplify the processing of messages by constrained nodes. On top of this message layer, CoAP uses request/response interactions between clients and servers.

If a node needs to send a message in a reliable fashion, in spite of UDP unreliability, then the node will send the message and wait for an acknowledgment. If no acknowledgment is received, the node will retransmit the message several times with an exponential back-off. These messages are referred to as CON (confirmable), in contrast to NON messages (non-confirmable) that nodes can afford to lose.

CoAP applications and resources are identified by URIs following the *coap* scheme (or *coaps* with DTLS). Group messaging is also possible with CoAP, by specifying a multicast address in the URI host part. This allows several resources to be accessed with a single request.

CoAP requests are derived from the main HTTP methods (GET, PUT, POST or DELETE) and the responses from HTTP statuses. PUT creates a resource, GET retrieves it, POST updates it and DELETE deletes it. As for the responses, CoAP uses HTTP statuses with some slight semantic differences. Informational and redirection HTTP statuses are not used in CoAP. In addition to its UDP binding, CoAP differs from HTTP regarding its message options. Messages may have one or more options. The list of options includes Content-Format, Accept, Max-Age, Uri-Host, Uri-Path, Uri-Port, Uri-Query.

An original work worth citing is the Observe option, [61]. A client subscribes to resource updates by sending a GET request with the Observe option so the server sends notifications upon resource modifications.

UDP is the standard binding for CoAP. However, several other bindings have already been envisaged. The informational Internet Draft [62] examines the requirement of several alternative transport protocols for CoAP, and mentions the potential interest in using the BP. To our knowledge, only the SMS binding has led to an actual test implementation [63]. On the other hand, CoAP has also been used as a SOAP-binding itself described in the Internet Draft [64] and presented in the article [65] where this system is deployed in a WSN.

Different works relate CoAP usage. In [66], CoAP is deployed in a cargo to monitor the different containers (temperature, humidity, voltage). In [67], CoAP is used in a wide area for sensor and actuator use-case. It should be noted that in this last article, sensors and actuators are equipped with communication means relying on cellular network. This system is considered in this thesis to be too expensive, especially when the number of sensors and actuators increases.

CoAP also supports elementary description and discovery mechanism. It references [68] for description and [12] for discovery in its section defining resource discovery.

The description of CoAP resource is defined in [68] as Web Linking. A link is described as a typed connection between two resources. This typed connection contains attributes in the form of a set of key/value pairs. Web Linking offers different attributes that provide hint on different target, or link, characteristics such as its media type. The set of defined attributes is very limited and does not provide any means to define specific attribute of a resource, or service, that would be interesting in a DT-WoT context such as an interface, or an agenda. It should be noted that CoAP also defines a new Web Linking attribute to inform what Content-Format a resource can return.

Once the resource are described, then can be discovered. CoAP discovery involves of two steps. The first is to discover all CoAP nodes, the second is to discover hosted resources (if any) on these nodes. Discovering all CoAP nodes is done through a request sent to the multicast address of CoAP. These two steps can be merged into a single one, as explained below. A strong constraint on the default port number is imposed by the RFC of CoAP. Indeed, a CoAP server *must* listen to the default port to support the discovery. CoAP servers can host resource on an additional port if needed. The RFC6690, CoRE Link Format [12], defines how to use Web Linking to discover resources hosted on a server. When a client wants to discover hosted resource it needs to send a GET request to the server. A well-known URI is used as a default entry point to which the request are directed. A client can send its request either by using unicast or multicast to discover all resources of all CoAP nodes at once. CoRE Link Format also defines additional attributes that are claimed to "*describe information useful in accessing the target link*", unfortunately these attributes do not provide any means to defined meaningful attributes in a DT-WoT context.

2.2.2 Service-Oriented Approach

Many proposals of SOA middleware systems target the IoT, WSNs, or ubiquitous environments. While none of these works target DTN environments they actually target networks that share similarities with DTNs, e.g. devices with constrained resources, noncontinuous connectivity. It can be wondered if these works can be adapted into an SOA middleware for DTN. Unfortunately, very few studies actually propose systems that can be adapted as is in a DT-IoT context.

In the following, works on SOA middleware systems are categorized according to their characteristic that make them more or less adaptable to a DT-IoT context. Sometimes this characteristic lies in their design decisions, e.g. usage of *big* services [59]. There are works for which this characteristic lies in their objectives. Such works propose SOA middleware systems for the IoT while generally aiming, as a tacit secondary objective, to fill the gap between traditional Internet and WSN. However, this thesis does not consider Internet as being part of the context. Another characteristic that makes SOA middleware systems unadaptable for the DT-IoT context is the focus on semantic techniques while leaving design principles aside. Finally, there are studies and works that demonstrate appealing architecture and vision that could inspire this thesis contribution. These works present RESTful and decentralized architectures.

2.2.2.1 *Big* Services

Systems such as Hydra [69] or ubiCoAP [70] rely on protocols that are not considered as resource friendly, i.e. consuming too much CPU, RAM or battery. There are systems that require too many exchanges of messages. It is the case of MUSIC [71] for its discovery or a work by Guinard et al., [72] where the services are *scanned* in order to report their availability status to a central system whereas DTN architecture advises to minimize them. Being in a DT-IoT context heavy protocols such as SOAP should be avoided. Also, DTN advises to minimize messages exchanges so these approaches should not be adopted into this thesis context. Furthermore, it would be interesting to rely on other and decentralized systems in order to determine if a service is available since centralized architecture are not efficient at all in DTNs. It would also be interesting if clients were informed beforehand when a service is not available.

2.2.2.2 Integration with Internet

Studies such as KASOM [73], TinySOA [74] or the one proposed by Anastasi [75] aim to integrate WSNs into the traditional Internet in order to extends the Internet with such networks. KASOM [73] provides tools for the registration, discovery, composition, and orchestration of services. KASOM also proposes to found *Knowledge Management* services according to a semantic description of low and high level resources of the WSN. TinySOA [74] proposes a language-agnostic API to exploit WSN capabilities. They provided an API to a set of programmers to evaluate their perception and intention of use. Several simple applications were developed as well as a more complex one for a real-world problem (agricultural monitoring) that has been in use for several months. The system proposed in [75] helps to configure WSN functionalities and exploit them in the form of Web services. Industrial systems as Xively [76], Carriots [77], Echelon [78], are based on cloud platform to provide their services. It relies on a centralized architecture that cannot be adapted to DTN context without heavy modifications that would render the DTN systems very different from its original system.

These propositions rely on service-oriented approaches but their objectives differ from the goals of this thesis. One of their goal is to integrate, or exploit, Internet connectivity. This vision is not specific to these works. Indeed, as the research team of CHOReOS project [79] puts in their *perspective* of SOA for the IoT [55]: “the *Internet of Things will cooperate with the Internet of Services*”.

2.2.2.3 Semantic Style

ASAWoO [58] and SENSEI [57] are two projects adopting a semantic approach. In ASAWoO, reasoners process ontologies in order to determine how to achieve a goal. For more details on ASAWoO, see below. In SENSEI, some models are proposed to describe IoT components, data and capture relations between different data provider and data descriptor.

While semantic tools clearly help to build SOA middleware systems, especially for description of services, looking up services and composing of services, it is a research domain in which the main objectives of this thesis do not lie.

2.2.2.4 RESTful and Decentralized Style

Finally, there are works suggesting some appealing architecture characteristics to provide SOA middleware in IoT, that could be adapted in DT-IoT systems. In particular, ubiREST [80] that relies on a RESTful architecture. This work introduces a *network-agnostic connectivity layer* that can be likened to the BP convergence layer, and exploit both unicast and multicast, namely ubiREST point-to-point and ubiREST group. The middleware relies on a *refinement of the* REST *style*, called Pervasive-REST (P-REST), presented in [81]. It is an adaptation of REST at different level of abstraction to offer flexibility, genericity, and dynamism. The approach of this work supports the idea previously presented that REST is promising in the context of this thesis. MOSDEN [82] is a middleware providing a plugin architecture that enable to load applications in the form of plugins distributed through mobile application stores such as Google Play. MOSDEN is built on top of GSN [83] that relies on a decentralized P2P architecture. Servilla [84] is an SOA system that introduces a new binding model based on semantic that can be eager or lazy, and persistent or transient (any of the four combinations). It also offers three forms of service invocation: on-demand, periodic and event-based. In Servilla, each node hosts its own registry. Decentralized systems, RESTful architecture and distributed registries are valuable methods for DTN environments.

2.2.3 Other Approaches and Systems

The categorization of ROA and SOA is at a coarse-grained level. Some middleware systems do not entirely fit in these categories. These systems generally embed devices management, devices communication as well as application-layer programming support. They are, for the most part, industrial projects that mask out their real approach, that might be ROA or SOA.

Industrial Projects

Kura [85], by Eclipse, proposes devices communications, network and gateway management for an IoT integration through a Java/OSGi-based implementation. MicroEJ [86], an industrial middleware, follows the same approach and relies on microJVM that is a Java virtual machine for embedded systems. IoTivity [87], later merged into AllJoyn [88], are client-server oriented. AllJoyn purpose is to enable devices to communicate with other devices around them and also to ease device management operations. A device's abilities are advertised through an XML document. Z-Wave [89] and BACnet [90] protocols are compatible with AllJoyn thanks to third-party. LWM2M [91], by Open Mobile Alliance, is a protocol for device management. It is client-server oriented. It relies on UDP, and, like very few, on SMS too. WhereX, presented in [92], is an event-based middleware, proposes the identification of different objects in different contexts for enterprise systems.

Research Studies

In [93], a middleware is proposed to allow various ubiquitous devices to discover one another and link them with web resources or services. Perera et al. present MOSDEN [82] that suggests things collaboration where more-constrained devices request more-powerful nodes to execute data processing on their behalf.

Traditional programming supports for the IoT (ROA, SOA or other) cannot be ported as is in DTN context. Indeed, if DTN and IoT share similar constraints, assumptions made for IoT networks and DTNs vary a lot. Because of these differences, designs and architectures of IoT protocols or programming supports naturally make them unsuitable for DTN networks. Nonetheless, the work of this thesis lies in this specific challenge that is to port, adapt and transpose IoT protocols into DTN context.

2.3 DT and Opportunistic Computing

In addition to networking challenges, DTNs and OppNet also present programming challenges. The *Opportunistic Computing* aims to solve these challenges. Indeed, Conti et al. presented the concept of *Opportunistic Computing* in [94, 95] that considers a broader vision of OppNet encompassing autonomic computing and social networking aspects to enable collaborative computing tasks in such environments. Several programming styles that are used for traditional networks cannot be used to program application in DTNs and OppNet. Indeed, when a system requires a lot of message exchanges or very short delays, i.e. default time outs are impractical in DTNs, it cannot be easily adapted in such challenged networks.

Few implementations and prototype are actually available and they vary from each other in their decisions and assumptions on network nodes capacities, mobility patterns or objectives. These differences stem from the wide range of possible scenarios OppNet can be deployed in but also on what challenges and objectives challenged networks are deployed for.

Among studies focusing on application and programming in DTNs, and OppNet, two categories emerge and are review below.

2.3.1 Pragmatic Programming

A general manner to develop applications in both OppNet and DTNs is to adopt a pragmatic approach. As Ott puts it in [96]: DTNs are "*in nature and generally used by specifically developed applications*".

In other words, when studies deploy applications in such networks, the studies focus more on environmental context and objectives rather than design of application and programming principles. It is the case of works deployed to monitor wildlife (zebras [97] and whales [98]), or [99] that aims to be deployed in Vehicular Ad hoc Networks to help finding a place to park or inform drivers on weather conditions.

Thedu, the system proposed in [100], aim to use an Internet proxy to which mobile nodes can send web queries, likened to queries to a search engine. The results of these web queries are then stored by the proxy and several results are prefetched in order to maximize chances of providing the most useful responses first. The contribution of this study lies in their *aggressive but selective prefetching* practices.

In the SNC project [3], part of the N4C project [101], the objective is to provide an Internet access to remote areas. More specifically to the Laponia region in northern Sweden where there is no infrastructure and thus, no Internet connectivity. A gateway with Internet connectivity through a line-of-sight was used as an entrypoint of the mobile part of the network. Helicopters are also used as data-mules to carry bundles over the mountains. Three differents applications were tested: e-mail, *not so instant* messaging and web access.

A similar work is presented by Lindgren in [102]. A gateway allows its users to connect to Facebook from a DTN network. The system enables users to read their news feed, post status and photos, comment and like the posts of other people.

In [103], Pitkanen et al. propose to use WLAN hotspot and DTN messaging to reduce the load on cellular networks. Their proposal can also be used in order to provide Internet access to nodes without cellular connectivity. To test their proposition they implemented a prototype implementation of a mobile DTN-based web browsing. They conclude that the node density in the area of commercial and community hotspots is sufficient to satisfy the majority of the requests issued by pedestrians within 20 minutes.

Ott and Kutscher, in [96], propose an Internet access to mobile users using existing applications relying on HTTP. They suggest that prefetching techniques can be used in order to gather large resources into bundles in response to a single request, as is the case in [100]. Prefetching techniques not only reduce the number of requests/responses but also fasten the satisfaction of mobile users since the resource they may be interested in can be embedded in bundles enclosing response's data of their queries.

DODWAN, that is both an implementation of a forwarding protocol and a middleware, has been instantiate in an application suite [104]. Application suite include a news stream, a messaging (both text and voice) app, mail service and file sharing. All applications are contained in an opportunistic network.

2.3.2 Programming Paradigms

Despite a common pragmatic approach, there are also some proposition on protocol adaptations, and suggestions of design principles to develop DTN applications. Such studies, reviewed below, generally focus on adapting the Web (HTTP, MIME), or SOA into DTNs.

Web Approach

Ott and Kutscher [105] propose a protocol design and a system architecture to enable access to web pages in a delay-tolerant network thanks to a *slightly enhanced variant of* HTTP running on top of BP. Two architectures are envisioned. The first one involves only DTN nodes, Web servers and Web clients are not connected to Internet, whereas the second architecture suggests to use a proxy at the edge of the DTN network that is also connected to Internet. The second architecture aims to provide Internet-Web access.

In [106], Wood et al. decouple HTTP from TCP. The aim of their work is to enable the exploitation of HTTP and MIME in DTNs. They consider HTTP as *well-understood by applications* and as an enabler to deploy applications into DTNs as well as move content in DTNs.

Service-Oriented Approach

Service-Oriented Approach is common in DTNs and OppNet. In fact, the Opportunistic Computing [94] and service provisioning in OppNet [107] emerged at the same time. In [107], the authors consider service provisioning as a *key feature* of Opportunistic Computing. While some studies focus on the social aspect of services and Opportunistic Computing, there are also studies focused on service invocation mechanisms and service composition.

Added to the service-oriented approach, there exist a few studies that propose a different approach to develop application in DTNs such as the tuple space [108, 109] paradigm.

Social Oriented Social aspect has been exploited by several research studies. That is the case of the middleware named CAMEO [47] (presented above). The authors of [47] introduce the social-awareness as *a fundamental requirement to develop optimized systems and applications [in* OppNet*]*. They even qualify their networking context as *opportunistic mobile social networks*, which demonstrates their focus on the social aspect.

In [110], social structures are considered to extract a metric representing node's social significance. This metric is used to determine best paths between nodes and create a sub-graph of the network in order to find best placements of services within the network.

In [111], authors suggest to exploit social information to determine whereto and how data should be disseminated in order to optimize content availability. The system they propose and evaluate is named ContentPlace according to different social-oriented policies.

CASA Research team CASA presents a clear interest in services and OppNet. DODWAN [45, 104, 112] is one example of this interest.

Another study focuses on service invocation over a content-based communication [113]. This study details the invocation mechanisms, how the client satisfaction can be increased by exploiting the potential multiplicity of service providers and how techniques of network healing can reduce the network load. Network healing techniques consist to remove unnecessary messages from the network storage.

Finally, a work studies discovery and composition of services in OppNet [114]. It proposes, simulates and compares a choreography-based and an orchestration-based algorithms based. It concludes that orchestration-based algorithm reaches a better success ratio than the choreography-based algorithm. Nevertheless, it induces longer delays in the composition process.

ASAWoO ASAWoO [115] project is the closest project and most related to this thesis's objectives: it is a service-oriented system that uses RESTful or Web protocols and is tolerant to disconnection. ASAWoO stands for: Adaptive Supervision of Avatar/Object Links for the Web of Objects. ASAWoO proposes a whole architecture to deploy WoT applications where physical objects, things, are extended through virtualized avatars. Avatars, that may run on different objects, are context-aware and adapt to environment changes. Avatar architecture relies on multiagents systems based on RESTful and Web services. These avatars uses HTTP and CoAP to communicate. This Web platform follows eight requirements among which a disconnection tolerance. A proposal to fulfill this specific requirement is presented in [116]. It proposes a disruption-tolerant communication middleware between avatars, hosted in the cloud, and their objects when they are not connected to the cloud. This middleware exploits by two DTN adapters. The first is C3PO [44] (presented above), and the other one is IBR-DTN interfaced through a Java API (developed during this thesis).

ASAWoO services capabilities and functionalities are semantically described thanks to a Web ontology language. Service descriptors are then advertised through RESTful PUT requests sent to other nodes. These requests are either unicast or multicast. Another possibility to discover service descriptors is to request other nodes with a GET request to which nodes answer with their own service descriptors. Invocation mechanism relies on RESTful requests/responses and is improved by supporting four non-functional parameters dedicated to requests that are: caching (to allow intermediaries nodes can reply to the request if they carry a valid response), time (to inform a date at which the client does not need a response anymore), spatial (to circumscribe at a coarse grain the area in which the messages can be forwarded), and asynchronous communication (to allow a client to inform a specific endpoint to which the response should be sent).

Both programming styles (pragmatic development and based on paradigms) come with their own advantages and defaults. Pragmatic solutions are usually finetuned for specific scenarios or solve concretely identified problems but are not so optimized when deployed in other scenarios. Yet, this is not a blocking issue since they do not intend to do more than what they were developed to. On the other hand, solutions based on paradigms are not necessarily fully-optimized for all scenarios but they provide solid bases to developers for new applications. It is this orientation that this thesis contributions are made. More specifically in the line of work of [105, 106] for which objectives are to support a Web-enabled DTN by adapting HTTP.

2.4 Discussion

DTN and OppNet differentiate to one another not only on the challenges they are focused on but also on their approaches. Indeed thanks to its wound-down DTNRG, DTN researchers produced RFCs and Internet Drafts in order to formalize and standardize architecture and protocols. The challenges targeted by the DTN research field usually stem from scenarios like IPN or bringing Internet to remote areas. As for the opportunistic networking, challenges targeted are commonly include communication in a defined *local and wide* areas without communication infrastructure. Solutions, apart from the vast variety of routing algorithms, traditionally follow a pragmatic approach and there is, at my best knowledge, no RFC on opportunistic networking/computing. This thesis sits on both sides of research fields aiming to provide programming supports close to standard protocols while targeting infrastructure-less scenarios.

The IoT is an emerging paradigm that draws a lot of interest and investment. Despite all the work and means put into it and despite many works on middleware systems for the IoT very few protocols stood out as standards. Furthermore, even though IoT and DTN share similarities there is no standard protocols of IoT that can be deployed as is in a DTN context. This thesis aims to fill this gap by defining DTN-enabled protocols inspired from IoT standards, hence supporting a DT-IoT.

After reviewing DTN, OppNet and IoT programming models the programming styles of DTN and opportunistic contexts are presented. Far fewer studies focus on these contexts compared to the IoT, and, as shown, very few works target standard protocols adaptation. Such studies gain the benefice of relying on well-known, tested and approved protocols. In my opinion, more studies should adopt such approach and this is an approach that this thesis contributions follow in order to inherit these advantages.

To conclude, despite many works on middleware systems for the IoT, several application deployments in DTNs as well as a couple of works on programming paradigms in the context of DTN there is no emerging standard protocols or design principles for a DT-IoT context that bears both constraints of IoT context and DTN context. Nevertheless, several studies show appealing approaches and characteristics. A lot of approaches exist but ROA and SOA are considered in most of the works. The REST architecture is also a silver lining coupled with HTTP concepts (as the request/response codes).

Two protocols, BP, intended to be a standard, and CoAP already a standard, step out as promising. The former, BP, overcomes the communication challenge identified in the DT-IoT context while the latter can be transposed into a delay-tolerant protocol. Once transposed it could support an ROA middleware system. This ROA system could then support an SOA middleware systems. Both systems would then be adapted to both WoT and DTN contexts.

3 BoaP

Contents

The previous chapter presented CoAP and BP as the two most important protocols for this thesis. This chapter, being the core of the contributions, presents the work done to adapt CoAP over BP and a prototype of this adaptation named BoaP.

This chapter first reminds the designs of BP and CoAP. One implementation for each protocol is also presented from a technical point of view. The protocol correlations and dissociations are highlighted to: *a.* justify that they are a good match, and *b.* show the rising challenges when CoAP is put on top of BP as is. CoAP adaptations stem from those challenges and are detailed below. A prototype of an adapted CoAP over BP, named BoaP, has been implemented. It relies on a Java API to communicate with the IBR-DTN daemon. Both this Java API and then BoaP are presented. Their performances are measured in the next chapter in a real network and then in a simulated one with emulated nodes running BoaP.

3.1 Presentation of BP and CoAP

This section presents the most important aspects of both BP and CoAP. Besides, it should be noted that even if BP aims to provide an application layer for DTNs, it is considered as a communication layer in this work. In like manners, even though CoAP defines itself as a “web transfer protocol” it is considered in this work as an application layer protocol, as its own name implies.

3.1.1 Bundle Protocol (BP)

The BP is the standard protocol to exchange messages within the DTN architecture [5] networks. The BP forms a message-based overlay that follows the store-carry-and-forward principle. The BP defines the format of the messages, called bundles, and the logic layout to process them but does not define any routing algorithm yet many exist.

As a network overlay, the BP relies on subnet-specific protocols called Convergence Layers to transport bundles, e.g. TCP, UDP or even specific transport protocol such as Licklider Transmission Protocol (LTP) [117] that is already oriented as delay tolerant. In order to overcome network disruptions and high delays, the BP uses a cache to store bundles. These bundles are either processed by an application (if the destination is hosted on the node), or forwarded to other nodes towards the bundle destination. A BP endpoint, i.e. bundle destination or bundle source, is identified by an Endpoint Identification (EID) that takes the form of a URI. A BP endpoint can either be a singleton or a set of BP endpoints that register themselves with a shared EID, thus allowing multicast-like operations to be performed. Nevertheless, a bundle cannot be sent from such EID.

Bundles are constituted of one primary block (header), then zero or more Metadata Extension Blocks (MEB [7]), and one or more payload blocks. The primary block carries options that influence the treatment performed by the intermediary nodes forwarding the bundle as well as the treatment performed by the bundle destination(s) that receives the bundle. For example, enabling *Report-When-Bundle-Delivered* option will make the destination node emit an administrative bundle when receiving the bundle. Extension blocks, called MEB, can be used to make specific processing decisions regarding bundles, e.g. routing decisions.

The bundles have a TTL and will be deleted when it expires. With shorter TTL value the network storage, or nodes’ cache, shall not be saturated, on the contrary, with infinite TTL the bundles will never be deleted and all the nodes storage would rapidly be saturated. The optimized TTL value greatly depends on the deployed network: some may prefer a lower delivery ratio due to constrained storage nodes while other networks may involve nodes with bigger storage that would permit longer TTL.

The BP does not offer a reliable means of communication. Nevertheless, a built-in mechanism, named "custody transfer", aims to improve its reliability. The custody transfer requests that a BP node takes the responsibility for delivering a bundle to its destination. This responsibility is released when the node forwards the bundle to some other node accepting this responsibility. The message can be under custody only if the destination is a singleton.

3.1.2 IBR-DTN

Several BP implementations exist but IBR-DTN [18] stands out. Indeed, even if DTN2 was first adopted by the community its last release is four years old, on the contrary IBR-DTN is still under active development with at least one update or merged pull request every two months[1].

IBR-DTN choice is motivated by two facts: *1.* this thesis's context is the DT-IoT, formed by constrained nodes, and *2.* that IBR-DTN is a RAM-friendly implementation of BP. Indeed, as exposed in the introduction, the IoT is formed of many constrained devices that have very limited battery and communication means. These communications means, usually Bluetooth or Wi-Fi, offer limited range-radio and, due to their battery, the things are most likely to suffer from sleeping cycle: during their sleep their communication interface is turned off to save up energy. Furthermore the things are expected to be part of our environment but usually follow an architecture where their data are uploaded into far away servers. These data are not accessible when the infrastructure is damaged or ineffective making the system incompetent despite the fact that the data are present locally, on the things themselves.

The IBR-DTN implementation, just like DTN2, comes in the form of a daemon. The choice of using a daemon is motivated by the RFC defining DTN [6]. This daemon discovers its neighbors to which it forwards to and receives bundles from. The daemon also takes care of the bundle storage, receiving and delivering them to and from its registered endpoints.

A registered endpoint is an application able to send and receive bundles. Communication between endpoints and daemon is possible through a TCP socket by means of textual exchanges. The socket to which endpoints bind is owned by the daemon. This solution to communicate offers different advantages: it is language agnostic and it also permits very constrained nodes (not powerful enough to run their own daemon) to register endpoints on a more powerful node hosting an IBR-DTN daemon. Nevertheless it comes with its downsides too: due to the sequential nature of a socket it is not possible to parallelize bundle receptions (or emissions). Another downside is that the socket induces an evident additional delay that, even though it is negligible, might create a bottleneck.

As endpoints communicate with the daemon through a textual TCP socket an API is defined and available[2]. A BP endpoint sends commands and data to which

[1] https://archive.fo/ECSIX
[2] http://archive.fo/2KBwm

the daemon answers with one-line responses. If a command is successful then its response starts with a code: `200`, a `100`-code response inquires that more information is expected from the BP endpoint. Several endpoints can be connected to the daemon and registered. When the daemon receive a bundle it checks if the bundle destination. If the destination node is itself it checks if an application is registered as the destination URI. If one is then the daemon notify the endpoint by sending a `602 NOTIFY [...]`, if none is then it stores the bundle until an application registers as the destination URI or until the bundle lifetime expires. It is the charge of the daemon to deliver received bundles to their rightful local endpoints by respecting three steps that are: *1.* notifying the endpoint, *2.* storing the bundle, *3.* return metadata and data when requested by the endpoint.

An exchange between the daemon and an endpoint to register, send or receive a bundle requires several commands. The succession of commands and responses needed to perform these simple actions can be likened to conversations. Conversations always start with a command sent by the endpoint to register, send or receive a bundle. Upon reception of a bundle for a local endpoint, the daemon notifies the endpoint through the TCP socket by sending a `602 NOTIFY [...]`. These notifications may be sent at any time, even during a conversation to send a bundle between two successive commands. These notifications are not considered as a conversation initiation as it is the duty of the endpoint to fetch the notified bundles when the endpoint is ready. Conversations are always ended by a daemon response. This last response acknowledges that the last command is successful. When an error occurs the endpoint may continue the conversation until the daemon acknowledges the last command as successful. These conversations differ with sessions on their start and end. Sessions are expected to be started, or ended, with a same procedure call, e.g. command, being `open`, or `close`. However in the case of IBR-DTN, a conversation can be started by a specific command that is different for each action.

Three actions are detailed below with their full conversation logged and explained: registration, bundle emission and bundle reception. In the examples, to visually differentiate daemon outputs and endpoint inputs an artificial number sign (`#`) is added before each daemon response and its banner. The daemon can be requested to switch into its extended protocol mode. This mode allows more commands such as `neighbor list`, `neighbor protocols` and `neighbor connections` but these commands are not presented in this work even though following examples were executed in this mode.

3.1.2.1 Registration

An application becomes a BP endpoint once it is registered as an endpoint to the IBR-DTN daemon. An endpoint registers itself by sending the command `set endpoint <eid>`, where `<eid>` is replaced by the path part of the EID. An example of an endpoint registration with the daemon is shown in List. 1 where the endpoint register as *Mars*. If the node name is *dtn://node* then the whole EID will be *dtn://node/Mars*. It can be seen that the daemon answers to each command with a `200`-coded response.

```
#IBR-DTN 1.0.1 (build 1d1df7b) API 1.0
protocol extended
#200 SWITCHED TO EXTENDED
set endpoint Mars
#200 OK
```

Listing 1: IBR-DTN Conversation: Endpoint Registration

3.1.2.2 Bunde Emission

While registering an endpoint require only one command, once the daemon is in *protocol extended* mode, sending a bundle require, at least, three different steps, with two commands, illustrated below:

1. The first step is to enter the command `bundle put plain` that informs the daemon that a bundle needs to be created so the endpoint can provide metadata and data of the bundle in order to send it. To this, the daemon answers with a `100`-coded response acknowledging the request and notifying the endpoint that the daemon is ready to receive bundle details.
2. The second step is then to provide the bundle details that are the bundle metadata such as the source, the destination, the flag, the options, the timestamp, that will be put into the primary block, and also provide the bundle payload blocks (at least one). In the example presented in List. 2, the bundle has only one payload block representing the content `"Hello\n"` encoded in base64, see line 17 in List. 2. The end of this input is marked by a double empty lines.
3. Finally, the command `bundle send` informs that the bundle is complete, e.g. no further modifications are expected, and requests the daemon to disseminate the bundle as soon as it can.

3.1.2.3 Bundle Reception

When a bundle is received by the daemon, it verifies if the endpoint is locally registered. If it is, then the daemon sends a notification to the endpoint through the socket. In the example in List. 3 the notification is the first line: `602 NOTIFY BUNDLE <timestamp> <seq_nr> <source_eid>`, where `<timestamp>` marks the timestamp of the bundle creation, `<seq_nr>` is the sequence number and `<source_eid>` identifies the bundle source endpoint. After this notification, four commands are then required to fetch the bundle:

```
bundle put plain
#100 PUT BUNDLE PLAIN
Source: api:me
Destination: dtn://59/Phobos
Processing flags: 144
Timestamp: 1479823174254
Reportto: dtn:none
Custodian: dtn:none
Lifetime: 1800
Sequencenumber: 0
Blocks: 1

Block: 1
Flags: LAST_BLOCK
Length: 2

SGVsbG8K


#200 BUNDLE IN REGISTER
bundle send
#200 BUNDLE SENT
```

Listing 2: IBR-DTN Conversation: Bundle Emission

1. `bundle load <timestamp> <seq_nr> <source_eid>` requests the daemon to load the bundle into its bundle register. The bundle is identified by the three parameters `<timestamp> <seq_nr> <source_eid>` present in the notification.

2. `bundle info` requests the metadata of the bundle loaded in the register, such as the flags, timestamp, source, destination, lifetime, number of blocks and the length of each block.

3. `payload [<block_offset>] get [<data_offset> [<length>]]` requests the content of the payload block identified by the parameter `<block_offset>`. The other parameters, `[<data_offset> [<length>]]`, can be used to request a subset of the payload data.

4. `bundle delivered <timestamp> <seq_nr> <source_eid>` acknowledges the bundle so the daemon can unload it from its bundle register and delete it from its storage.

If a bundle is composed of several payload blocks, the line `Blocks: 1`, line 11 in List. 3, will show the actual number of payload blocks present. Metadata of each payload blocks (its number, its flags, its length) will then be sent by the daemon. The command `payload get` is needed one time for each of the payload blocks. The payload content may be split into several lines if it contains many data.

```
#602 NOTIFY BUNDLE 533138377 1 dtn://139/uxyLjauFpklhUsoA
bundle load 533138377 1 dtn://139/uxyLjauFpklhUsoA
#200 BUNDLE LOADED 533138377 1 dtn://139/uxyLjauFpklhUsoA
bundle info
#200 BUNDLE INFO 533138377 1 dtn://139/uxyLjauFpklhUsoA
Processing flags: 144
Timestamp: 533138377
Sequencenumber: 1
Source: dtn://139/uxyLjauFpklhUsoA
Destination: dtn://139/Sender
Reportto: dtn:none
Custodian: dtn:none
Lifetime: 3600
Blocks: 1

Block: 1
Flags: LAST_BLOCK
Length: 45
Encoding: skip

payload 0 get 0 0
#200 PAYLOAD GET
Length: 45
Encoding: base64

VHVlIE5vdiAyMiAxNDo1OTozNyBDRVQgMjAxNiBmcm9tIHBjLW1uYS0xMzkg

bundle delivered 533138377 1 dtn://139/uxyLjauFpklhUsoA
#200 BUNDLE DELIVERED ACCEPTED
```

Listing 3: IBR-DTN Conversation: Bundle Fetching

3.1.2.4 Neighbor Discovery

A beaconing system has been implemented in IBR-DTN in order to discover neighbors. This system relies on the Internet Draft IPND [20] considered as *unfinished business*[3]. A beacon is a small announcement usually sent via broadcast, or multicast, embedding canonical name of its emitter. When a neighbor receives this beacon it can then initiate exchanges, e.g. of bundles or routing informations, and add the node to its neighbor list. A beacon is composed the following fields:

- Version,
- Flags,
- Sequence number: increment at each beacon sent,
- EID length: byte length of the canonical EID,

[3]Mentioned as such in last DTNRG meeting at IETF-87 while DTNRG was closing down.

- Canonical EID: EID of the beacon's emitter,
- an optional service block since IPND also supports a basic service discovery,
- an optional value that informs on the beacon period.

Four flags are currently defined and informs on the presence of: *a.* Source EID should always be set, *b.* Service Block, *c.* Neighborhood Bloom Filter: this filter helps to determine if a link is bi-directional, *d.* Beacon Period.

IBR-DTN provides various options to customize neighbor discovery. These options include: *a.* Disable/enable, *b.* Version to be used, *c.* Address to which send beacons, *d.* Short beacon (to add or remove the service block), *e.* Cross-layering: if set the daemon do not advertise its own address and neighbors are expected to extract it from sender IP address.

It should be noted that the IBR-DTN Linux package also provides several tools that allow to send/receive bundle, synchronize a directory or ping a node.

3.1.3 Constrained Application Protocol (CoAP)

General Concept

CoAP [11] offers an application layer protocol that allows resource-constrained devices, i.e. the *things*, to interact together asynchronously. It is designed for machine-to-machine use-cases and is compliant with the Representational State Transfer (REST) architecture style. CoAP defines a complete messaging layer, with a compact format, that runs over UDP (or DTLS when security is enabled). CoAP requests are derived from the main HTTP methods (GET, PUT, POST or DELETE) and the responses from HTTP statuses. For the requests, PUT creates a resource, GET retrieves it, POST updates it and DELETE deletes it. As for the responses, CoAP uses HTTP statuses with some slight semantic differences. These features (methods and response codes) make CoAP easily translated into to HTTP making it seamlessly integrated into the already existing web. However, informational and redirection HTTP statuses are not used in CoAP. The format of CoAP messages is illustrated in Fig. 3.1[4]. Its low header overhead and low complexity make serializing and parsing simple for constrained nodes. On top of this message layer, CoAP uses request/response interactions between clients and servers.

[4]Figure 7 from the CoAP RFC

Message Format

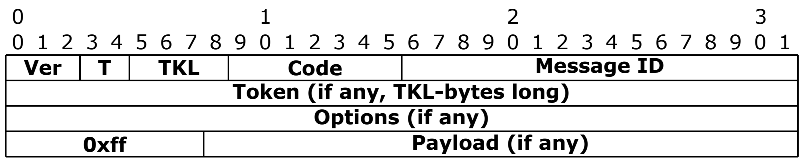


Figure 3.1: CoAP Message Format

All messages start with the version number of CoAP. This field, `Ver`, is a 2-bit unsigned integer. The current version of CoAP is 01. If a node needs to send a message in a reliable fashion, in spite of UDP unreliability, then the node will send the message and wait for an acknowledgment. If no acknowledgment is received after a customizable time out, the node will retransmit the message several times with an exponential back-off until the nodes reaches `max_retransmit` that is the maximum number of emission for a message. This retransmission mechanism aims to overcome the UDP unreliability. These messages are referred to as CON (confirmable), in contrast to NON messages (non-confirmable) that nodes can afford to lose. `T` field informs which type the message is. The type is a 2-bit unsigned integer and the four message types are: *(0)* Confirmable, *(1)* Non-confirmable, *(2)* Acknowledgement, *(3)* Reset. Reset messages are sent when a received message cannot be processed properly due to missing context, e.g. a node that just rebooted and receives a response could send back a reset message. `Code` field is an 8-bit unsigned integer that indicates whether the message is a request or a response and what is the request method (GET, POST, PUT, or DELETE) or the response status. `Message ID` field is an 16-bit unsigned integer used to detect message duplication and match messages of type Acknowledgement/Reset to messages of type Confirmable/Non-confirmable. `Token` field is used to match request and response. Since it is a variable length field, `TKL`, 4-bit unsigned integer, indicates how many byte(s) the token is. After the header, and the token if any, the message presents zero or more options. Finally, if the message embeds a payload, a marker `0xff` identifies the beginning of the payload data.

Options

Existing options are listed in Tab. 3.1. Some are repeatable. A repeatable option is an option that may appear more than once in a message. Repeatable options are identified by a cross in the column named R in Tab. 3.1. The semantic of `ETag` option differs whether it is in a request or a response. An `ETag` is a local identifier used to differentiate between distinct representations of the same resource. In a response, the `ETag` depicts the current representation of the resource. In a request, a client can provide multiple ETag values, since the option is repeatable, so the server can answer with *2.03 Valid* if the resource matches with one of these ETag values. This allows clients to ask if their resource presentation is still valid while preventing to the server to send the whole resource. `If-Match` and `If-None-Match` options are used for conditional request. If the condition(s) is not fulfilled then the server must answer with a *4.12 Precondition Failed* response. These option values are usually an ETag[5]. If an `If-Match` ETag matches a representation of a resource then the condition is fulfilled. `Uri-*` options, being `Uri-Host` , `Uri-Port` , `Uri-Path` and `Uri-Query` , are used to specify the target resource of a request. It is important to embed the `Uri-Host` within a request option as a server may have several hostnames and must be able to identify each resource. `Proxy-*` options, being `Proxy-Uri` and `Proxy-Scheme` , are used when the request is send through a forward-proxy. Proxies may have valid cached responses that they can send directly to clients. `Location-*` options, being `Proxy-Path` and `Proxy-Query` , are embedded in *2.01 Created* response to indicate where the resource requested by a POST is located. The path, and query, are relative to the URI of the request. `Size1` option is an integer used in *4.13 Request Entity Too Large* to indicate the maximum size in byte of request entity that the server is able and willing to handle. When a request, or a response, with a payload is sent the `Content-Format` option is used to identify the representation format of the payload. This option is related with the `Accept` option that a client sent in a request to inform the server which `Content-Format` is preferred. If the server cannot send the request resource in the representation format indicated by the `Accept` option then the server must answer with a *4.06 Not Acceptable*. For obvious reason `Content-Format` option is not repeatable, `Accept` option isn't neither yet a client may accept different `Content-Format` . Some resources are valid for a specific duration and thus may be cached. To specify how long a resource can be cached the `Max-Age` option is used. The lifetime of the resource is a number of second encoded in an integer.

Adressing and Multicast

CoAP applications and resources are identified by URIs following the *coap* scheme (or *coaps* with DTLS), defined as coap:// *host* [:*port*] / *path-abempty* [*?query*], e.g. coap://zeus.foo.bar:7800/museum/outside-light?number=3. The host part is compulsory, the path identifies the resource within the scope of the host and port (5683 by default), and the query part details the resource access. Other protocols than UDP are already considered as CoAP binding: section 3 of RFC 7252 [11] suggests that "other transports such as SMS, TCP, or SCTP" can be used as CoAP binding and also specifies that UDP-lite and UDP zero checksum are not supported by CoAP.

Group messaging is also possible with CoAP, by specifying a multicast address in the URI host part. This allows several resources to be accessed with a single request. Before answering a multicast request, the servers are expected to wait a *leisure time*. This wait aims to avoid network congestion in the case many servers would answer the request at the same time. Also the servers are encouraged to not answer at all "if it doesn't have anything useful to respond". The CoAP applications should be

[5]Entity-tag

No.	R	Name	Format	Length (byte)
1	x	If-Match	opaque	0-8
3		Uri-Host	string	1-255
4	x	ETag	opaque	1-8
5		If-None-Match	empty	0
7		Uri-Port	uint	0-2
8	x	Location-Path	string	0-255
11	x	Uri-Path	string	0-255
12		Content-Format	uint	0-2
14		Max-Age	uint	0-4
15	x	Uri-Query	string	0-255
17		Accept	uint	0-2
20	x	Location-Query	string	0-255
35		Proxy-Uri	string	1-1034
39		Proxy-Scheme	string	1-255
60		Size1	uint	0-4

Table 3.1: CoAP Options

aware that a request is received through multicast as their expected behavior should be adapted.

CoAP Extensions

In some cases a client may want to track a resource, either for history or to take action. Such client could be heater that needs to knows the temperature of the room, to stay up-to-date it may request a temperature sensor frequently to heat more or less in order to stabilize the room temperature. In order to minimize the number of request/response while optimizing this scenario a CoAP extension has been proposed: the Observe option, RFC 7641 [61]. This RFC allows a client, the heater in our case, to register as an observer to a resource, the temperature, by sending an extended GET request. The server, temperature sensor, returns the current state of this resource and add the client to a list of observers that will receive notifications upon resource modifications.

Because CoAP relies on UDP large resources (such as firmware update) cannot be sent through regular CoAP messages as they are not likely to fit a single UDP datagram. To overcome this issue, an update of CoAP has been proposed: the block-wise transfers, RFC 7959 [118]. This update provides the needed means to transfer rather large resources that could not fit into one UDP datagram while following CoAP constraints and architecture. It defines the requirement for the request, the notifications, the caching, the reordering and the transmission of such resources.

3.1.4 Californium

Many CoAP implementations exist. Some are oriented for constrained devices (for Contiki), other for smartphones, other for servers. Among these multiple implementations written in Go, C, C#, Java, and so forth, some come as web application, as Web Browser plug-in, framework or even as command line tool. For this work, Californium (Cf) is the CoAP implementation of reference as it is open-source and under very active development: more than 5 updates per month in the third quarter of 2016, there is almost 30 developers who contributed to the source code (that is about 74k lines) that has been migrated into Eclipse Foundation. However, being written in Java it is not oriented for constrained devices. Cf implementation does not only provide CoAP features, RFC 7252, but also the Observe option, RFC 7641, and the Block-wise Transfers, RFC 7959. Below some server and client examples are shown. These examples are based on the example from the source code of Cf itself.

To implement a CoAP server by using Cf a resource must be implemented first. A CoAP resource is already present: `CoapResource`. It replies with a *4.06 Not Allowed* response for every method. The easiest way to implement a new resource is to extend this class as shown in List. 4. The extended class overrides methods `handleGET`, `handlePOST`, `handlePUT`, `handleDELETE` according to its needs for the respective request methods GET, POST, PUT, and DELETE. If a resource only need to answer to GET request then it only overrides the method `handleGET`. Once, at least, one resource is implemented then it can be added to a `CoapServer` and, finally, the `CoapServer` can be started. After following these three steps: *1.* override `CoapResource` methods, *2.* add the resource to a server, *3.* start the server; the CoAP server is up and running.

Let's take a deeper look at the List. 4. In the example below only GET and POST method are overrode. Others methods (PUT, DELETE) will then replies with a *4.06 Not Allowed* response. The method `handleGET` is the callback executed when, as its name indicates, a GET request is received. In this call back, the method call `respond` of the class `CoapExchange` answers a *2.05 Content* directly to the client by embedding a payload set as the parameter. In the method `handlePOST` the exchange is marked as accepted. The method call `accept` of `CoapExchange` first tests if the request's type is CON and if the request has not yet been acknowledged and then, if the condition succeed, sends an acknowledgment to inform the client that the exchange has arrived and a separate message is being computed and sent soon. In the `main` method a new `CoapServer` is created to which is added a `CoapResourceExample` and then the server is started. To request such server, or resource, a client example is shown in List. 5 and 6.

```java
public class CoapResourceExample extends CoapResource {

    public CoapResourceExample(String name) {
        super(name);
    }

    @Override
    public void handleGET(CoapExchange exchange) {
        exchange.respond("hello world");
    }

    @Override
    public void handlePOST(CoapExchange exchange) {
        exchange.accept();
        // .. slow creation processing
        exchange.respond(CREATED);
    }

    public static void main(String[] args) {
        CoapServer server = new CoapServer();
        server.add(new CoapResourceExample("example"));
        server.start();
    }
}
```

Listing 4: Cf Resource Example

Cf provides two different ways to request CoAP resource. It can either be synchronous or asynchronous. In List. 5 the request is sent (see l.5), without any option, and the response is waited on this same line. Upon reception the payload is put into the `destination` variable and printed.

```java
1  String destination = "coap://californium.eclipse.org:5683/test";
2  CoapClient client = new CoapClient(destination);
3
4  // Blocking call:
5  String content = client.get().getResponseText();
6  System.out.println("Response: " + content);
```

Listing 5: Cf Client: Synchronous Example

In List. 6 an asynchronous client is presented. A nested class handles the received response (or the error).

```java
String destination = "coap://californium.eclipse.org:5683/test";
CoapClient client = new CoapClient(destination);

client.get(new CoapHandler() {
    @Override
    public void onLoad(CoapResponse response) {
        String content = response.getResponseText();
        System.out.println("Response: " + content);
    }

    @Override
    public void onError() {
        System.err.println("FAILED");
    }
});
```

Listing 6: Cf Client: Asynchronous Example

Cf API is easy to use and, being focused on scalability, offers high performance. Fig. 3.2 presents the Californium Architecture. To ease the reading, in the following description Java class names are used ignoring whether it is an actual class, abstract class or interface name.

Messages are received in their raw format by a `Connector`, usually a UDP or DTLS connector, and passed to `InboxImpl` that uses `DataParser` to parse messages. Once parsed, the messages are processed by the `Matcher` that matches incoming responses with the outgoing requests into the right `Exchange`. An `Exchange` is created when a request is received, or sent, and ends in the case: *a.* the last response has arrived and is acknowledged, *b.* a request or response has been rejected from the remote endpoint, *c.* the request has been canceled, or *d.* a request or response timed out. Working with the `Matcher`, the `Deduplicator` aims to detect, and optionally drop, the duplicated messages (both CON and NON). Once a message is matched in an `Exchange` it is sent to the `CoapStack` that actually retransmits messages when needed, keep track of the registered observers (RFC 7641) or reassembles the different blocks (RFC 7959). When the message (response or request) is ready and needs to be transfered to client or server, the `CoapStack` forwards through `MessageDeliver` it.

When a CoAP client sends a request, or a client produces a response, the message is sent by `CoapStack` through `StackTopAdapter` down to the `Outbox`. The `Outbox`, after some tests, i.e. *is the destination set?*, calls the `DataSerializer` to produce the raw data of the message that is then sent directly through the `Connector`.

Optionally a class can implement the interface `MessageInterceptor` to log or produce statistics on the number of sent/received CON/NON requests/responses.

An interested reader will find more details on Cf performances in [119]. This publication presents Californium as focused on scalability rather than resource-efficiency

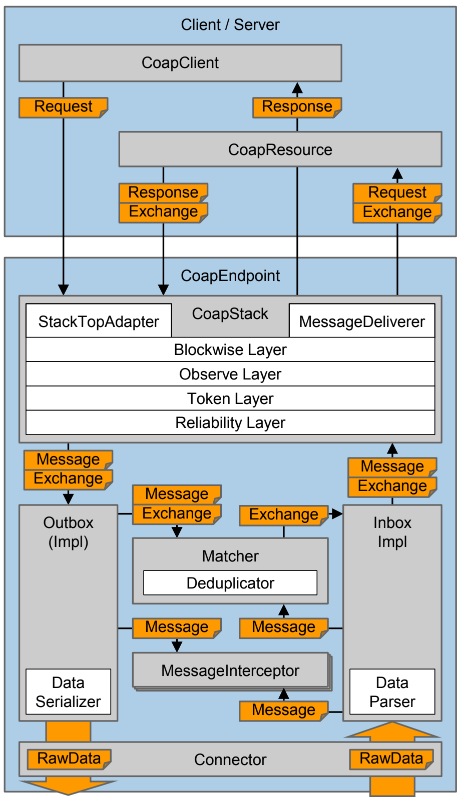


Figure 3.2: Californium Architecture (from M. Kovatsch's thesis reused with permission, main Cf developers)

and demonstrates its responsiveness by outperforming all other platforms Cf is benchmarked against. It should be remarked that one of the author of [119] is the first author of CoAP(RFC 7252) [11], Constrained RESTful Environments Link Format (RFC 6690) [12] and the second author of block-wise transfer (RFC 7959) [118]: Z. Shelby.

With the insight provided above on both BP and CoAP, the next subsection highlights their common and distinct concepts to first justify the choice of these protocols and then expose the challenges that arose.

3.1.5 BP and CoAP: Arose Challenges

As considered in RFC 7252, other protocols than UDP could be used as a binding. Prototyping a BP binding for CoAP could be done by putting each CoAP message into a BP bundle and preserving all the CoAP features that had been designed for the traditional UDP binding. Nevertheless this naive approach does not take in consideration DTN specificities and does not take advantage of the BP specifications. Indeed, the BP is not a transport-layer protocol and offers richer capabilities than UDP. Therefore several mechanisms of the BP can be exploited to enhance a CoAP/BP prototype and improve its performances. Coming along these enhancements there are also several challenges that arise. In this subsection the incompatibilities between BP and CoAP are highlighted and possible enhancements are discussed.

The first incompatibility is the endpoint addressing. Even if other protocols than UDP are mentioned in CoAP, its tightly reliance on concepts such as IP address and port number makes it challenging to be deployed over other kind of networks. An Internet Draft proposed SMS as CoAP transport [120], not active anymore, but tackles this issue by using a service center. Such solution is not feasible in DTN networks. Another solution would be to use IP address in BP endpoints but this constraint is too strong especially when a convergence layer may not lie on top of an IP transport protocol.

Another, yet related, incompatibility is the multicast support of CoAP. The communication group that the multicast provides can be used to request several CoAP resource at once. Again, as BP is not related to IP at any layer the multicast support cannot be simply put over BP.

In DTNs the number of messages must be minimized. Added to this, the meaning of CON is two fold: *a.* the message packet has not been lost during its transport, and *b.* the endpoint was active, so it received and started to process the request. These two reasons, minimum number of messages opposed to CON reemission and the dual semantic of acknowledgements, make the CON mechanism challenging and some choices are to be made. Indeed, a node may be active and receive a CON request for a local endpoint that is not active. Should an acknowledgement be sent in this case? Should the node wait for the endpoint to be active again? Any response to these questions are not trivial and deserve discussions as well as comparisons of different solutions.

CoAP uses a parameter in order to limit an endpoint data rate towards a non responding node. Its default value, an average, is 1 byte per second. While this mechanism avoids congestion in traditional network there are use-cases where it would be harmful to restraint a request (or response). For instance a node may send a request to a CoAP resource and may need to send another request, before the first one is answered, to a CoAP resource hosted on the same host. Waiting an artificial time because of this data rate would induce additional delay in a network that already suffer from delays and disruptions.

Last but not least, caching mechanism is an important aspect of CoAP protocol: several section defines why to use it, i.e. to limit network traffic, to improve performance, to access resources of sleeping devices, and for security reasons, and also how to used. It is mainly used through proxy servers. In DTNs nodes there is no node reliable enough that could provide proxy services. The CoAP caching mechanism cannot be ported as is into BP networks.

Also to conclude BP and CoAP presentation, it can be noticed a paradoxical way both protocols present themselves. The BP goal is to disseminate messages (bundles) in a challenged network but presents itself as an application layer protocol. In the mean time, CoAP presents itself as a RESTful transfer protocol while it actually provides a whole logic processing with deduplication means, caching mechanisms, considering exchange packet units as requests and responses (and not as neutral-kind messages) and even carries "Application Protocol" in its own name.

The next section presents the proposed solutions for the challenges presented below. Some of these challenges are answered in a practical way by being implemented in the BoaP prototype, while others were not implemented but proposed solutions are still detailed.

3.2 CoAP Transposition for a BP-binding

As seen just before, even if BP seems to be a good candidate as a CoAP binding, some incompatibilities between the two protocols remain. These incompatibilities need to be solved in order to provide a protocol as close as CoAP with BP binding. Solutions to these incompatibilities are presented below as both fundamental adjustments and adaptations.

3.2.1 Fundamental Adjustments

Fundamental adjustments stem from incompatibilities between CoAP and BP. These fundamental adjustments focus on addressing, multicast, CON meaning, and caching incompatibilities.

3.2.1.1 Addressing

As explained above, CoAP relies on concepts such as IP address and port number. This tight reliance makes it challenging to put a binding protocol that is not following IP, such as BP. A solution would be to force BP node identification to an IP address. But this solution is not feasible: some convergence layers do not use IP addresses, in the cases IP addresses are used it is usually dynamic, the node identification would be under a heavy constraint and finally node would lose the possibility of having multiple hostnames. Another solution, that actually seems natural, would be to merge CoAP and BP node identifications. Indeed, both identifications relies on URI: in CoAP the URI-* options are inserted into the CoAP message to describe the targeted resource, in BP URI details are in the bundle header as the string identifies the destination. Also, according to the BP, URI-query is not considered as a part of the endpoint identification. Besides IBR-DTN treats an endpoint with a query as a whole different endpoint, e.g. IBR-DTN daemon would consider that messages destined to `dtn://node-id/app?param=a`, `dtn://node-id/app?param=b` or `dtn://node-id/app` as three different endpoints. By merging these two identifications at the bundle layer, CoAP options URI-Host, URI-Port and URI-Path are therefore either meaningless or redundant. Indeed, the URI-Port option does not have any use in networks running the BP as the nodes do not have any port. As for the host and the path URI options, these are redundant with the bundle metadata and should not be added into the bundle payload because it would induce an unnecessary overweight. However, the URI-Query option must remain at CoAP level.

Added to the network addressing, the addressing scheme of the URI should be considered too. While the CoAP RFC specifies to use the URI scheme *coap* when UDP is used (or *coaps* with DTLS), there is no mention of alternative transport layers. As the registered *dtn* scheme is not yet precisely defined, it is suggested to use a CoAP-compatible scheme in which *host*[:*port*] is replaced by any alphanumeric string, that typically represents a node or a group of nodes. This scheme could be named *coap+dtn*, as suggested in [62].

3.2.1.2 Multicast

CoAP supports requests sent to a group of servers by using an IP multicast address in the URI. In the BP, an EID can be a group of different applications running on different nodes, in other words a set of BP endpoints. This feature can be used as a substitute to the IP multicast group, thus allowing requests to be sent to several destinations at once.

If a node needs to send an identical request to a list of N servers then it sends a single bundle to the EID to which the N servers registered to. The other alternative would be to produce N requests, one for each server, and disseminate each of these requests as if they were regular and different requests. The gain of multicast requests is clear: without multicast there would be N identical requests disseminated within the network instead, with multicast, only one single request is. The gain of storage in the network is equal to $N - 1$ multiplied by the number of nodes carrying the message multiplied by the size of the bundle. This gain also has an equivalent in energy gain as $N - 1$ bundles multiplied by the number of forwarding are saved up as transmission or reception.

Note that it can also be interesting to combine the multi-payload with the multicast. In short, one bundle can carry several responses (or requests) while being sent to several BP endpoints.

The Observe option could also be implemented using this group of BP endpoints. To register as a resource observer BP endpoints only need to register to a specific group EID to which the server would send the notifications.

3.2.1.3 Confirmable Messages

Added to the fact that DTN design principles constrain applications to minimize the number of round-trip exchanges and that the dynamic behavior of DTN networks induce long and very often very variable delays then DTN applications should not rely on a heavy reliability. This is why it can be expected that applications are most likely to use NON messages instead of CON.

However, CON messages may be useful in some circumstances, i.e. when a client sends a configuration update to ensure that the update occurred. CON messages are used to ensure two distinct facts: *a.* the request packet has not been lost during its transport, and *b.* the endpoint was active, so it received and started to process the request. The most important parameter to use CON messages in DTNs is the time-out before the first reemission. This time-out should be equal to the upper bound of the time required by a bundle to cross the network, plus the time for a destination BoaP node to resume from its potential sleep cycle (as node may sleep to save battery), plus the time to cross the network again. Calculating, or setting, the time a bundle needs to cross a network is mostly empirical. In fact the lack of real-world DTNs makes it hard to estimate this time duration.

As part of a another solution that is more in line with DTN, several features of BP can be used as CON substitution:

- *Custody Transfer*: it can be assumed that, by using the custody transfer, a message have smaller risk to be dropped from the network and is more likely to reach its destination.
- *Report-When-Bundle-Acknowledged-By-Application option*: this option requests that the BP daemon running on the destination node generates an administrative bundle once the request-bundle is received and taken in charge by the application (in our case the BoaP layer). This administrative bundle can be a substitute to the CoAP acknowledgment. Setting this option for CON messages avoids taking care of the acknowledgment at the CoAP level as it

is sent directly by the BP daemon. The similar option *Report-When-Bundle-Delivered* is not so convenient because it does not involve the BP application (i.e. BoaP in our case) and therefore prevents BoaP from piggybacking the acknowledgment to the response if it is quickly available. Added to this, the bundle may be delivered but the application may be stopped (or in sleeping mode) and the bundle could reach its lifetime before the application is awoken. This would be an issue as the acknowledgement would have been sent inspite of the fact the request may never be processed by the application.

- *Expedited class of service*: there are three classes of service for bundles. The classes are, from the least to the most important, bulk, normal and expedited. Higher-class bundles are forwarded with priority over others, as long as the source is the same. If bulk (or normal) is used for NON messages, it is suggested that expedited be used for CON messages in order to accelerate their transport, and, hence, reduce the probability of their lifetime expiring.

Again, applications are expected to take into account the unreliability of the underlying network so they should not require network reliability and thus use NON. This is also in the line with DTN constraints. In the meantime, if BP mechanisms are not sufficient to substitute CON occasionally re-emitting a request might be acceptable if the minimal back-off duration is at least equal to twice the TTL of the bundle, assuming that the TTL is well set to be equivalent to the duration a bundle requires to reach its destination, plus the time for a destination BoaP node to resume from its potential sleep cycle.

3.2.1.4 Caching

In CoAP, endpoints may cache responses of GET requests in order to reduce the response time and medium usage on future, equivalent requests. The option Max-Age is set on a response to assign a duration of freshness to it. CoAP endpoints are allowed to provide a cached response if it is sufficiently fresh. This caching mechanism is only performed locally, that is, by the sender of the request, or by a proxy (specified in the Proxy-URI option of the CoAP message) that issues the request on behalf of this client.

For CoAP, the caching mechanism imports a lot: in CoAP RFC [11] four whole different sections are dedicated to the caching mechanism. Whilst in the BP the caching is not designed to fulfill a request, as BP does not consider messages as request nor response anyway, it is still intrinsically present in its underlying layers. In fact, the BP follows the DTN architecture that relies on the store-and-forward concept, which can be likened to cache-and-forward provided that caching is close to storing.

The caching mechanism goal, as presented in the CoAP RFC, is to reduce latency, the network round-trips and the medium usage usage. In the context of the IoT, and even more in the DT-WoT, the network usage must be reduced to its minimum. Not only the network is usually characterized by a low bit rate but also using it requires energy that comes at high price.

In a DTN network, this caching mechanism could be extended so that relay nodes would also be allowed to provide a cached response, as if these relay nodes were proxies. The gain in response time could therefore be drastically reduced. However, some cross-layering is necessary to implement this mechanism: the BP relay node must pass the bundle payload to a local BoaP proxy code in function of some BoaP-level information. This information includes the URI but also the method type and other CoAP options. These are normally only available in the payload of the bundle, which should remain opaque.

Using an extension of the BP called "Metadata Extension Block" [7] could enable this cross-layering support. A Metadata Extension Block (MEB) is designed to carry additional information used by BP nodes to make processing decisions regarding bundles. In this case, the MEB would contain all the necessary data about the request so that the BP layer can trigger a local BoaP Proxy code that will potentially send a BoaP response.

It should be noted that a double storage issue stems from the caching. According to code design the cached responses may or may not be directly accessible from the BP storage. If this is not the case a solution is required to access, read and parse the stored bundles to detect valid responses and matching requests. Granted that the caching mechanism is at a higher layer of the bundle storage it may not access them directly, assuming the code respect the layered design. Additionally, once the bundles have been parsed and the messages of interest detected, these responses must be directly accessible so they can be sent back to the client. This implies to store twice the still-valid responses and waste node storage. A better solution would be to offer trusted applications the possibility to manage bundles (edit, send, remove, send as, store longer) but the challenge of determining which application is trustable rises with all the security concerns.

3.2.2 Enhancements

Added to the fundamental adjustments some enhancements are also proposed in order to optimize a BP-binding for CoAP. These enhancements rely on a feature of BP (e.g., multipayload), `accept`-option characteristic (e.g., that is not repeatable) and enabling PATCH method.

3.2.2.1 Multi-payload

CoAP suggests that each CoAP message should fit into a single UDP datagram. However, the BP allows several payload blocks to be included in the bundles. It is common for a node to stay isolated during a significant period, without the possibility to forward any message. In BoaP, it is thus recommended that all the BoaP messages sharing the same destination should be appended into a single bundle, each message being in a different payload block, until the bundle is ready to be forwarded. This avoids the transport of several identical primary blocks. A bundle header, e.g. primary block, is about 100 bytes therefore if ten requests or responses are to be sent from one node to another adding all of them into one sole bundle would spare 900 bytes. Not only energy will be saved up, as these 900 bytes won't be emitted, but storage in the network will be spared too. It must be reminded that most of the primary block fields do not have a fixed size so the header size can vary a lot.

The optimal size of bundles is defined as the biggest size for which the bundles have very low risk of being fragmented during the forwarding operations. This optimal size depends on the duration of contacts, the medium usage, the volume of data the nodes need to forward. All those metrics depend on the mobility pattern, the type of deployed network and the applications running. Nevertheless, it is safe enough to assume that if the bundles fit in a packet unit of the convergence layer there is low risk of being fragmented. If the convergence layer is TCP or UDP then according to IPv4, all hosts must be prepared to accept at least 576 bytes datagram. Depending on the specific convergence layer 20 bytes may be used for the header (TCP or UDP) and 100 bytes for the bundle primary block leaving about 450 bytes for the payload blocks to fit into a single packet unit. These values are just a magnitude, again the bundle fields size are variable.

A BoaP request size with a few options and a short payload is about 25 bytes so 18 requests can fit an IPv4 datagram. The responses are more likely to be bigger as there is a great chance of carrying a payload, so assuming a response weights twice a request size it is then 9 responses that can fit in an IPv4 datagram.

Here again, the layered design imposes some limitations. Indeed, the OSI model ensures that the upper layer do not have to deal with the lower layer parameters. In the case demonstrated here the maximal size of the transport layer datagram influences the upper layer and the number of BoaP messages that should fit into one bundle.

A good solution would be to set the limit of the bundle payload a BoaP implementation should not exceed and sum the messages sizes to check whether or not it can fit in the same bundle. A better solution would be to leave this task to the daemon that could automatically append bundle payload blocks to existing, and not already forwarded, bundle sharing the same destination. If a bundle becomes too big, and would imply fragmentation, then the daemon could split the payload blocks into two different bundles of equivalent size. Nevertheless, note that adopting an intermediary policy by dispatching the BoaP messages in more than one bundle is generally not feasible because no information is available on the actual convergence layers that will be used along the path, information that would be necessary to ensure a beneficial dispatching while avoiding packet fragmentation. There are also some networks relying on several convergence layers hardening even further this adaptation.

3.2.2.2 Option Accept

A CoAP client and a server may need several exchange to determine which `Content-Format` is both producible by the server and consumable by the client. Indeed, as stated in the RFC: "If the preferred Content-Format cannot be returned, then a 4.06 "Not Acceptable" MUST be sent as a response". In this case, if the client can process a different `Content-Format` then it sends another request with a different `Accept` value. This may lead to several exchange before a client finally find the right `Accept` value that the server can produce.

This kind of exchange is highly discouraged by the DTN architecture. Allowing the `Accept` option to be repeatable could offer a significant benefit by preventing these exchanges for negotiation. The advantage is not specific to BoaP but it is particularly important to prevent unnecessary end-to-end transmissions in a DTN network. Avoided exchanges would spare the network nodes energy consumption, as fewer exchange are performed, and storage usage, as fewer messages are disseminated in the network.

As a side note, the `Accept` option was, at first, repeatable. Indeed, the change occurred between the version 13 and 14 of the Internet Draft over a "rough consensus". The repeatability of this option caused some problems with intermediaries and caching[6].

3.2.2.3 PATCH Method

If a resource need to be updated, then the method PUT shall be used. It this resource happens to be large and the modification concerns only a couple of bit then the whole resource still need to be sent in the PUT request. Using the method PATCH [121] would allow the request to be far lighter as this method permits to only send the difference between the actual state of the resource and its updated state.

For instance, if a resource weights several kilobytes but only one byte has been modified then the request size would only be about several bytes and not kilobytes.

While this method would be great for DTNs, CoAP also could benefit from it too as some request could be lighter and may be easier to update as the block-wise transfer would not be required in cases several datagram are needed.

To conclude, some of the adaptations presented below are actual fundamental adjustments, such as the addressing and multicast, while others are enhancements, such as the multipayload, repeatable `Accept` option or PATCH method. The considerations about CON messaging are yet an open challenge. Indeed, as there is no deployed applications relying on the BP and even less using CoAP on top of that. This is why, due to the lack of practical experience and real-world deployments, it cannot be stated what is the best way of providing reliability in DTNs. Some of these adaptations are part of the prototype presented below, BoaP, that provides an

[6]See this first email of the discussion about this issue: http://archive.fo/XJiZm

adapted and redesigned CoAP with a BP binding.

3.3 BoaP Stack

To prototype CoAP with a BP binding three ways were considered. The first, as simple as naive, is to capture and encapsulate CoAP UDP datagrams into bundles but this is not feasible because datagram destinations are IP addresses so all BP endpoints should be identified by an IP address which is not a acceptable constraint. A second possibility would be to adapt an existing CoAP implementation nonetheless even in a Java implementations with a well structured layering, like Cf [119], replacing the entire UDP binding is not an option: many upper layers are too tightly coupled with the notions of IP address and port to perform a straightforward adaptation for the BP, in which these notions are irrelevant. A third solution would be to deploy traditional CoAP clients that sends their requests to a local proxy. This proxy embeds messages into bundles and disseminates it. This solution is being implemented and tested as part of a team project, it also has been published during the thesis in the article [116]. The last solution, the chosen one for this thesis, is to develop a new implementation. BoaP is the name of the prototype developed with the objective to firstly include a BP binding, and then potentially test some future extensions or modifications of CoAP that would be suited for DTNs. The BP binding of BoaP has been developed thanks to the IBR-DTN API [18] presented above.

To prototype BoaP an IBR-DTN Java API is required to interface a Java program with IBR-DTN. In the following, this API is illustrated then BoaP's architecture, API and CoAP compliant features are highlighted. The next chapter evaluate both the API and the prototype.

3.3.1 IBR-DTN Java API

To fit into the software ecosystem of our research team BoaP have been implemented in Java. However BoaP relies on IBR-DTN that has been implemented in C++. Fortunately, as presented above, IBR-DTN provides a full functional API that is language agnostic as it relies on a textual protocol through a network socket. This is why, before prototyping BoaP, an underlying layer has been developed: a Java IBR-DTN API that communicates with the IBR-DTN daemon through the textual TCP socket. While this adaptation layer could be put into the BoaP implementation it has been done as a self-content project for coupling reason. One of the main objective, added to the obvious one of providing an IBR-DTN Java API, is to offer an easy-to-use API. The source code is available online[7] and was welcomed by the community[8].

[7]https://github.com/auzias/ibrdtn-api

[8]Johannes Morgenroth, main developer of IBR-DTN, acknowledged that "[the API] looks really easy to use": http://archive.fo/mwtur

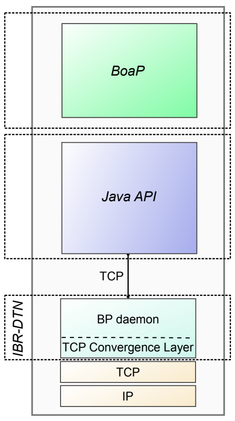


Figure 3.3: IBR-DTN, Java API and BoAP Stack

Fig. 3.3 presents, at high level, the three different software entities (IBR-DTN, the Java API, and BoaP prototype). The Java API is detailed in this subsection.

It seems important to remind that the API, enabling the communication with IBR-DTN daemon, passes through a network socket and that a whole conversation is needed to send, or fetch, a single bundle. Also, due to the sequential nature of a network socket, these conversations shall not be intermixed for endpoint: a bundle cannot be fetched while a conversation is on going to send a bundle, it goes the same way to respectively send, or fetch, a bundle while sending, or fetching, another bundle. Nevertheless, users of the IBR-DTN Java API should be able to concurrently send, or receive, several bundles. The conversations must be transparently sequentialized from the users point of view. To ensure this, the bundles to be sent are stacked up in a queue and are sent when there is no conversation ongoing. As for incoming messages, the daemon notifies new bundles through the socket so the BP can fetch them when there is no conversation ongoing.

The implementation architecture of this API is detailed below and shown in Fig. 3.4 in the dashed rectangle named *Java* API. For clarity reasons, some utility classes have been left out of the current presentation.

The input socket stream (from the daemon to the endpoint) is processed in a thread on its own by the class `CommunicatorInput`. It actually reads the input stream from the daemon socket, logs it, provides buffered bundles to fetch them and processes the bundle notifications to notify `Dispatcher`. The main goal of the `Dispatcher` is to keep track of the different conversations. For this, it uses states to exclusively fetch or send bundles, one at a time. A fetching procedure is illustrated in Fig. 3.5: it is assumed that the state of the endpoint is `Dispatcher` idle beforehand. This sequence diagram relates with the List. 3 as this is the same procedure. It can be seen on the Figure that the commands are sent from the `Fetcher`. The `Fetcher` class is threaded and sends commands to the daemon through the class `CommunicatorOutput`. Responses received by the `CommunicatorInput` updates the `Dispatcher`'s state. This procedure ends when the bundle is acknowledged by the

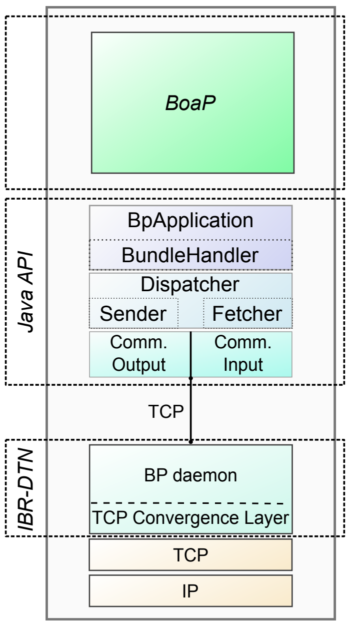


Figure 3.4: Java API Stack

application and that the daemon confirms that this acknowledgement is successful. The `Dispatcher` state is then set back to *idle*.

To send bundles, the class `Sender` is used. It also runs within its own thread. The `Fetcher` and `Sender` procedures are very close and both follow the example provided in List. 3 and List. 2.

From a higher level, the `BpApplication` provides means to create a BP endpoint, set its name (EID), set a specific bundle handler – implementation of the interface `BundleHandler` as the code example in List. 7 –, and sends bundles.

```java
public class PrintingHandler implements BundleHandler {
    @Override
    public void onReceive(Bundle bundle) {
        System.out.println("Received bundle:" + bundle.toString());
    }
}
```

Listing 7: BundleHandler Example

A full example of the implemented API is shown in List. 8 where a BP endpoint named *Phobos* set the `PrintingHandler` shown in the List. 7 and sends a bundle to another endpoint that is identified by the URI *dtn://mars/Deimos*.

3.3.2 BoaP Prototype

BoaP offers means to deploy both BoaP resources and BoaP clients. In the follow-

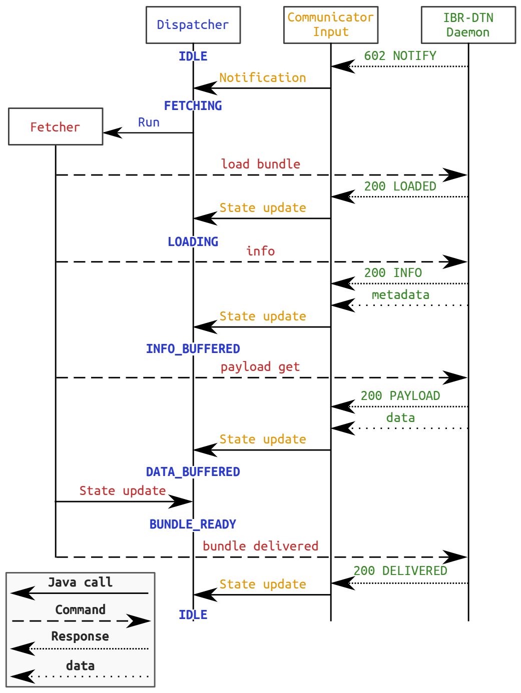


Figure 3.5: Sequence Diagram: Procedure to Fetch a Bundle from the IBR-DTN Daemon

ing both resource and client API are presented and illustrated with examples. The prototype implementation is compared to Cf implementation, as it is the implementation of reference for this work. The architecture of BoaP is then explained with a highlight of the caching mechanism and finally a table sums up what are the CoAP compliant features, what has been left out for future work and what has been adapted.

3.3.2.1 BoaP API

An interface named `MessageHandler`, presented in List. 9, is defined and provides both callback methods for requests and responses. A BoaP endpoint is expected to implement this interface in order to be able to process received requests and/or responses. If it is true that two different interfaces could have been defined, one for requests and one for responses, here a single interface is defined to allow resource requesting from resource providers. Indeed, if it seems unnatural for a client to receive a request a resource provider may need to send request to others resource providers. If the callbacks `*Method(Request rq)` are executed for each request the method `response(Response rp)` is executed only if the endpoint sent a request.

Above, when presenting Cf API, a resource example was presented, see List. 4. Here, a similar resource endpoint developed with BoaP is presented List. 10. Note that for clarity reason only the interface implementation is presented and unimportant methods are not shown. The creation of a BoaP endpoint is shown below in the

```java
public static void main(String[] args) {
    // Create BP endpoint with the uri-path being "Phobos"
    BpApplication bpApp = new BpApplication("Phobos");
    //Set the Handler that will process the received bundles
    bpApp.setHandler(new PrintingHandler());
    // Create a bundle
    final String destination = "dtn://mars/Deimos";
    Bundle bundle = new Bundle(destination);
    bundle.setPayload("Payload\n".getBytes());
    // Send the created bundle
    bpApp.send(bundle);
}
```

Listing 8: IBR-DTN Java API Example

```java
public interface MessageHandler {
    // Requests
    public void GETMethod(Request rq);
    public void PUTMethod(Request rq);
    public void POSTMethod(Request rq);
    public void DELETEMethod(Request rq);
    // Response
    public void response(Response rp);
}
```

Listing 9: MessageHandler Interface

client example and is the same for both client and resource. The `Response` constructor returns, if a request is sent as parameter, a response with the request's source set as response's destination, also if an integer is provided too then the response code is set to matching the integer, i.e. a 404 parameter would set the response code to *4.04 Not Found*.

To request this BoaP resource, a client must be created in a first time, then a request built and finally sent to the BoaP resource. This whole process is shown in List. 11. The class `ClientHandler` implements the interface `MessageHandler`, here again, for clarity reasons, only the important methods are shown: `Response(Response rp)` method just print the payload of every received responses. In the method `main` a `BoapEndpoint` is created with the URI-path part being *"client"*.

It should be noted that CoAP group messaging, provided by multicast for traditional networks, can be ported into DTNs by using non-singleton endpoint: every application registering at their own local daemon as, for instance, *dtn://group/boap* will be able to receive the same messages that are sent to this URI. This BP means can be used as a substitute of CoAP group messaging.

```java
public class ResourceExample implements MessageHandler {

    public void GETMethod(Request rq) {
        // Return a response with the response code "2.05 Content"
        Response response = new Response(rq, 205);
        // Charge the payload
        response.setPayload("hello world");
        // Send the response
        this.send(response);
    }

    public void POSTMethod(Request rq) {
        // Return a response with rq's source as destination
        Response response = new Response(rq);
        // ... creation (assumed successful)
        // Set the response code "2.01 Created"
        response.setCode(201);
        // Send the response
        this.send(response);
    }
}
```

Listing 10: BoaP Resource Example

3.3.2.2 BoaP and Cf API Comparison

With Cf, to create a resource endpoint, a class must be extended whereas in BoaP an interface must be implemented. The advantage of using an interface is that it allows a class to be extended while also implementing the interface, hence a BoaP resource can also extend a class. Cf resources cannot be extended without adding a sublayer of abstraction. This minor difference can be masked away in BoaP by developing a default resource endpoint and extended it.

As for client: BoaP does not provide native means to develop a synchronous one unlike Cf does, otherwise both Cf and BoaP asynchronous client are similar as they rely on an interface. An attentive reader may have notived that there is no `onError()` method for BoaP client. In fact, since DTN endpoints should be aware of the context they are in, missing responses processing should be common and a callback for this specific event is not required.

If BoaP prototype provides a single interface for both client and resource it is to allow Java inheritance without additional sublayer. Furthermore it also eases resource provider to request other resource providers as they need to implement only one interface: this can be likened to service composition by considering resources as services, more details on this are in the chapter 5.

```
1  public class ClientHandler implements MessageHandler {
2      public void response(Response rp) {
3          System.out.println("Response payload: " + rp.getPayload());
4      }
5
6      public static void main(String[] args) {
7          // Create BoaP endpoint with the uri-path being "client"
8          BoapEndpoint boapClient = new BoapEndpoint("client");
9          // Set the handler to the client
10         boapClient.setMessageHandler(new ClientHandler());
11         // Create the message:
12         final String destination = "dtn://callisto/resourceExample";
13         Request rq = new Message(destination, Code.GET);
14         rq.setPayload("payload".getBytes());
15         // Send the request
16         boapClient.send(rq);
17     }
18 }
```

Listing 11: BoaP Client Example

3.3.2.3 BoaP Architecture

The BoaP implementation is composed of three main classes that are `MessageReceiver`, the interface `MessageHandler` and `BoapEndpoint`. Again, for clarity reasons, some utility classes have been left out of this high level explanation.

The `MessageReceiver` class relies on a `BpApplication` of the IBR-DTN Java API to receive the bundles. Upon reception, the bundles are parsed as BoaP messages and tested to verify if they were already received (duplicate) or not. Duplicate messages are expected as most of routing protocol in DTNs uses multiple copies of bundles to increase the delivery ratio. If the message is not a duplicate, then the messages are forwarded to the application through the implementation of the interface `MessageHandler` presented above. It processes the requests and responses as defined in the implementation of the interface developed by the programmer. If the message is a request, it is processed and answered by sending back a response. If a default endpoint should be provided by the prototype then, by default, the responses would be *5.01 Unimplemented*.

The class `BoapEndpoint` is used to create a BoaP endpoint, set its name (path part of the URI), set a specific message handler – implementation of the interface `MessageHandler` – to receive messages, and sends requests. An example of this class is presented in List. 11 line 8 and below.

The implementation architecture of this API is illustrated in Fig. 3.6 in the dashed rectangle named *BoaP*. For clarity reasons, some utility classes have been left out of the current presentation. The BoaP client and server use the BoaP prototype to request or provide resources. In this example the BP convergence layer is TCP. The

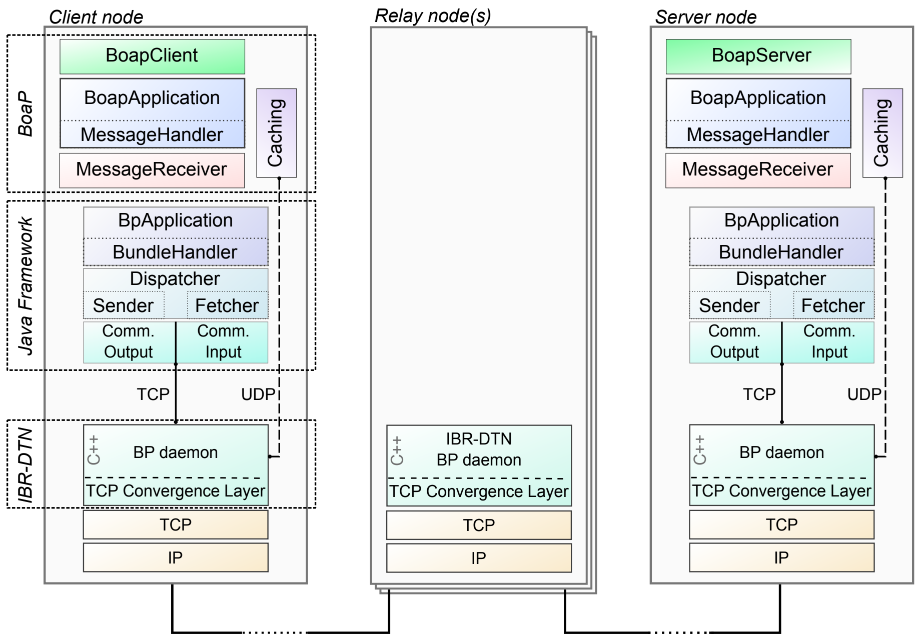


Figure 3.6: Connection between a BoAP Client and a BoAP server with full stack

client emits a request that will be forwarded to relay nodes, its neighbors, when possible. The request destination node may not be among these neighbors thus the request may need to be forwarded by several nodes before reaching its destination. When the request reaches the server it is processed and the response is sent back. Due to the node mobility, the relay nodes may be different and so is the number of hops needed to reach the source of the request. As a reminder, it is worth noting that the relay nodes do not require the application layer (BoaP) to disseminate the BoaP messages.

3.3.2.4 Caching in BoaP

Completing BoaP prototype a caching mechanism has also been developed. This caching, opposed to its representation on the Fig. 3.6 is composed of several classes plus a modification of the IBR-DTN daemon. Indeed, as exposed in 3.2.1.4, to provide a caching mechanism the needed cross-layering is provided by the modification of the IBR-DTN daemon. This modification is as small as possible and its only aims is to actually forward the stored bundles. Every bundle stored by the node are sent in a UDP datagram to a local port. This modification is represented in the Fig. 3.6 by the dashed line binding the IBR-DTN daemon and the caching tile. The caching mechanism, on BoaP side, relies on a UDP listener that creates a new `Bundle` each time the daemon forward one. This `Bundle` is passed through to the `Deserializer` of BoaP. If the message is not a BoaP message then the bundle is ignored; otherwise if the bundle either is a:

idempotent request then the caching service looks up for a matching still fresh response. If such a response is found then it is sent to the source of the request, otherwise the request is dropped.

successful response and still fresh then the response is stored. When its freshness times out, the response is deleted.

Of course, this caching mechanism is very simple as every node receiving an idempotent request will potentially answer. Without any other tests, i.e. *is the request already fulfilled ?*, a dense network could rapidly be flooded. To implement an optimized caching mechanism, heavy modifications on IBR-DTN are required. Indeed, the caching mechanism should be able to request the daemon to stop the dissemination of the answered request. In the actually state a BoaP node may answer a request while the IBR-DTN daemon still stores it and forwards it though it has been answered. Furthermore, by forwarding it to another node that have a cached response the network is flooded even more. Another IBR-DTN modification needed would be to forward bundle only once it has been tested for a cached response. These modifications are not possible without cross-layering that would unfortunately induces loosing all advantages that a well layered design provides.

A possible improvement to the caching mechanism would be a network healing means. A network healing mechanism aims to delete fulfilled requests from the network. Doing so not only free some storage but it also saves energy as requests and responses are not disseminated anymore. The implementation of the cache feature is naive: not only the node still disseminate the request to which it answered but it does not check if the response has already been sent by another node. As BoaP, and its caching feature, is just a prototype this network healing is the next step towards an improved implementation. This next implementation could reduce the number of sent responses when the caching feature is enabled.

3.3.3 Comparison of CoAP and BoaP Features

BoaP is mostly compliant with the RFC 7252: Tab. 3.2 lists the most important features and concepts of CoAP on one column and, on another column, this feature is marked as compliant, or adapted, in BoaP. Apart from some options considerations, the multicast substitute, the probing rate being ignored and the caching relying on a cross-layering BoaP prototype can be considered as a CoAP implementation with a BP binding.

Also, a difference between CoAP and BoaP caching lies in types of nodes that are part of the caching mechanism. In CoAP it is mainly proxies that act as cache enabled nodes, in fact there is few reasons other clients or service providers receive a request for which they are not the recipient and there is no reason such request would be parsed and answered. In DTNs messages are literally stored, carried and forwarded by any node in the network that may be, or not, a BoaP cache enabled node. In other words, any node in a DTN network may answer a cached response to any request.

CoAP Features	BoaP	
	Status	Comment
Asynchronous	Compliant	
Request/Response	Compliant	
REST	Compliant	
URI Identification	Compliant	
HTTP methods	Compliant	
Response Codes	Compliant	
CON/NON	Compliant	CON must be used as little as possible
Deduplication	Compliant	
Req/Resp Matching	Compliant	
Message Format	Compliant	
Options	Adapted	`Accept` : repeatable, `uri-*` : removed, expect `Uri-query`
Multicast	Adapted	Any non-singleton URI can be used as substitute
Probe rate	Adapted	Not implemented
Caching	Adapted	Any intermediaries may answer, rely on cross-layering

Table 3.2: Comparison of CoAP and BoaP Features

3.4 Discussion

A presentation of both BP and CoAP introduced this chapter. For each protocol an implementation is also presented at a technical level illustrated by examples of interactions with the IBR-DTN daemon (BP) and code usage of Californium (CoAP). This first part aimed to provide a good insight on the reasons these protocols can be a good fit.

However, CoAP remains unadapted for DTN context and cannot be used as is over BP. A transposition of CoAP is needed. The challenges arose for this transposition are detailed, including addressing, multicast, CON message and caching, and for each challenge several solutions are considered to finally propose the best one. These challenges are identified as fundamental adjustments but the transposition of CoAP over BP also consists of optimizations. These optimizations focus on exploiting BP multi-payload, making the option `Accept` repeatable and reintroducing the PATCH method.

Afterwards, two implementations were presented, a Java API[9] and a Java prototype: the first one provides an easy-to-use Java IBR-DTN interface and the other one, BoaP, prototypes a transposed CoAP over BP. BoaP main goal is to provide a proof-of-concept base to perform some tests and evaluate the feasibility of a BoaP deployment in larger networks. While it is functional, it cannot be assumed to be

[9]Referenced in the official documentation of IBR-DTN: https://archive.fo/7I6RY

fully-optimized: for instance, the simple caching mechanism would need a network-healing means.

Indeed, caching implementation is quite simple. Besides, the caching mechanism requires a cross-layering: the layered design is not respected anymore which takes away this solution from the standard protocol it is based on. This cross-layering can be masked away thanks to a MEB that could be used to inject specific code that can be triggered by the daemon. Such code could then follow the logic of answering a request if a matching and still valid response is locally stored. Then the code could delete the request from its storage since it has been fulfilled. Injecting code into a MEB has already been proposed by Borrego et al. [122] for routing purposes in order to move the routing algorithms from the host to the message.

Many features of BoaP are compliant with CoAP features. A proxy could be implemented to enable the cooperation between CoAP and BoaP nodes lying at the edge of two networks. Such proxy can be likened to gateways that are commonly used to integrate an ICN or a WSN to the Internet.

This prototype provides a platform to verify the feasibility of this protocol transposition and is evaluated in the following chapter. This prototype is also the core base of an SOA solution to enable a Delay-Tolerant Web of Things (DT-WoT) presented in chapter 5.

4 Evaluations

Contents

While the previous chapter presented the core base protocols of this work and their implementations, this chapter focuses on validating BoaP prototype through several tests and experimentations. This chapter aims to verify if the prototype BoaP is not nullified when deployed. To ensure this, BoaP is tested with few nodes in a small network involving no more than one client and one server, optionally an intermediate node too. These preliminary tests, considered as a proof-of-concept, are done to ensure that there is not any side effect or bug that would invalidate the prototype. These tests are fast to deploy and allow to quantify BoaP round-trip-times (RTT) between client's request and server's response. These RTTs are also compared to the theoretic ones. During these tests BoaP is also compared to Californium (Cf), a CoAP implementation. These tests are detailed in the first section.

Another aim of the current chapter is to determine if BoaP performs well in larger network. To answer this question BoaP is deployed in a network involving 50 network nodes. The nodes are emulated, using Docker, in a simulated wireless network. For these experimentations the network nodes follow two different mobility patterns: one simulated, based on a well-known mathematical model deployed in three different densities, and the other based on real GPS traces. The cache feature, that enables intermediate nodes to answer a request if they carry a response that is still fresh, is also introduced in these experiments to quantify its capabilities to fulfill exchanges that would not have been without it.

4.1 Preliminary Tests

Preliminary tests aim to verify if BoaP does not suffer from any bug or side effect that would discredit, impair or render unusable the prototype. Since BoaP lies on top of a Java API that provides means to communicate with IBR-DTN daemon it's important to attest that there is not any long delay added by this layer. In the following, the tests of Java API is presented followed by the deployment of BoaP in a small physical network.

4.1.1 Latency Between BP Endpoints and Daemon

Because a long delay at the Java API layer would negatively impact upper layers and more specifically BoaP, cross-layering latencies are measured. A cross-layering latency is the duration a message requires to cross the different layers. There are two different cross-layering latencies that are considered: top-down and bottom-up. In following measurements, the top-down one is the duration between the call of the method `send()` and the acknowledgement of the daemon, as opposed to the bottom-up one that is the duration between the notification of a new bundle from the daemon and the bundle being ready to be consumed.

To determine if the API performs well in bundle reception and emission cross-layering latencies are quantified according to the following procedure:

1. A timer called "emission" is started when a BP endpoint is created,
2. this endpoint produces 500 bundles, with small payload, and sends them (the daemon receives and stores them),
3. the timer "emission" is stopped once the daemon acknowledged the last bundle,
4. another timer, called "reception", is started when the destination endpoint is created,
5. the endpoint consumes the 500 bundles and dies right after,
6. the timer "reception" is stopped.

Average duration to process the 500 bundles is calculated as so: both durations processes spent in user and kernel space are quantified by using the Linux tool `time`. Then these values are summed up and finally divided by the number of bundles to determine an average value of cross-layering latencies.

The average latency to send (or fetch) a bundle to (or from) the daemon is about 3.5 ms with a standard deviation of 0.25 ms. These results were averaged over twenty runs. IBR-DTN tools, provided in the Linux package of IBR-DTN, are ten times faster. However this comparison is not really fair as these tools do not log the socket exchanges unlike the Java API. The latency of 3.5 ms may seem long, however it must be clear that this latency is not the duration that two nodes require to exchange a bundle. Indeed, these measurements were performed locally and are independent of connections or network dynamic.

To better understand the effect of this delay, let's assume a humidity sensor suffering from a long cross-layering latency of 10 seconds. The sensor makes a measurement put it into a bundle a sends it the to daemon. The bundle needs 10 seconds between its creation and the daemon's acknowledgement. If a mobile node passes close by and creates an intercontact duration of 3 seconds, then the two daemons may have enough time to exchange the bundle. The risk induced from a long cross-layering latency is to miss the opportunity to forward a bundle if this bundle is created 10 seconds, in this example, before the end of the intercontact duration. However BP applications are not expected to produce bundle according to the presence of neighbors because their behaviors must be totally independent from connections and disconnections events.

A delay of several seconds would have invalidated the prototype lying on top of this API. Nevertheless, whilst the Java API is slower than the IBR-DTN tools, it is not critical as the induced delay is negligible, does not undermine BoaP performance and does not slow down the communication procedure in a way that would make the BP endpoints lost opportunities of connections.

4.1.2 Validating BoaP in a small network

In the following, BoaP prototype is deployed in order to ascertain that it offers reasonable performance when the connectivity is good while supporting long disconnections, following tests were conducted in a small physical network in two scenarios named: intermittent connection and data-mule. To validate this prototype, BoaP RTTs of a request/response exchange between a client and a server, on both BoaP and Cf, have been assessed by these tests.

4.1.2.1 Testing Setup

In these tests BoaP used NON messages whereas CoAP implementation, Californium [119], used CON messages. In fact, since BoaP runs over BP it does not need to retransmit a new BoaP message as these are stored until they can be forwarded when a connection appears, on the contrary Cf runs over UDP that does not have any message buffer. Note that since the implementation of UDP is a part of the kernel therefore it is faster than an implementation of the BP that needs to run above its convergence layer and consequently induces delay. On the installation made for the tests, an ICMP ping is 50 times faster than a IBR-DTN ping (done with IBR-DTN tools). It is then expected that for a stable connection BoaP is slower that Cf. The following tests are deployed on Linux computers embedding 1 Go RAM, a dual 32-bit Atom processor cadenced at 1.60 GHz running Debian Wheezy (kernel 3.2).

The RTT is measured in function of the duration of disconnection. The RTT is defined as the duration between the time the client calls the `send()` method and the time when the response is received back.

4.1.2.2 Two Testing Scenarios

Two different scenarios were deployed for these tests:

intermittent connection This first scenario can be pictured as a sensor that follows a cycle of periodical sleeping and waking phases. The client and the sensor are always in ranged of each another, and the client goal is to log measurements of the sensor. Each time the sensor wakes up, the client sends a requests. Note that, for such a scenario, the option OBSERVE should be used instead as it better fit the needs when a resource need to be monitored. Fig. 4.1 represents this scenario. The link, represented by the dashed line, between the client, on the left, and the sensor, on the right, is cut off according to the sleeping cycle of the sensor. In this scenario the BoaP client sends a NON request while the CoAP one needs to send a CON requests. During the experiment, the requests are sent just after the link between the client and the server has been interrupted. This interruption lasts for what is called the disconnection duration. CoAP implementation is therefore forced to use its retransmission mechanism, which is triggered after a certain amount of time defined by the CoAP parameter ACK_TIMEOUT but the number of requests sent is limited by the MAX_RETRANSMIT value. Two values of ACK_TIMEOUT were tested (15 and 30 seconds). As for BoaP the request is sent to the daemon which waits for the reestablishment of the link to forward the request, hence no retransmission are needed as the request is stored (thus not lost nor dropped).

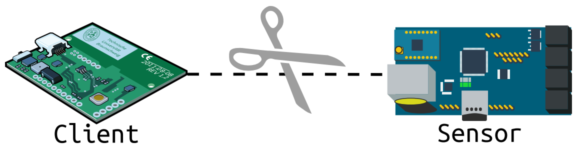


Figure 4.1: BoaP Test: Scenario 1

data-mule In this second scenario there is not any connected end-to-end path at any time between the client and the server. The client sends NON requests to the server periodically (every 30 seconds). In this context, "sends" means that requests are added locally to the IBR-DTN daemon queue every 30 seconds, whether the node is isolated or not. As shown in Figure 4.2, a third node, I, is used as an intermediary relay node (simulating a data-mule) that also runs IBR-DTN so that bundles can be transmitted between the client and the server. Note that the intermediary node does not need to have BoaP implementation to transport BoaP messages.

A cycle of connections/disconnections is enforced in the network, composed of four successive periods:

1. during Δ seconds, only the link between the client node and node I is active;
2. during $\Delta/2$ seconds, both links are inactive;
3. during Δ seconds, only the link between I and the server node is active;
4. during $\Delta/2$ seconds, both links are inactive again. Next period is period number *1*.

Fig. 4.2 represents the three different states throughout the four successive periods. The disconnection duration, Δ, displayed in abscissa in Figure 4.3b is the time during which BoaP nodes are isolated. Of course, this scenario has not been tested with Cf as UDP datagrams could not be transmitted to the server and would automatically be lost or dropped. Indeed, even with static routing rules the datagrams could, at best, be sent to the node I but as UDP does not store any datagram then the messages would not travel further.

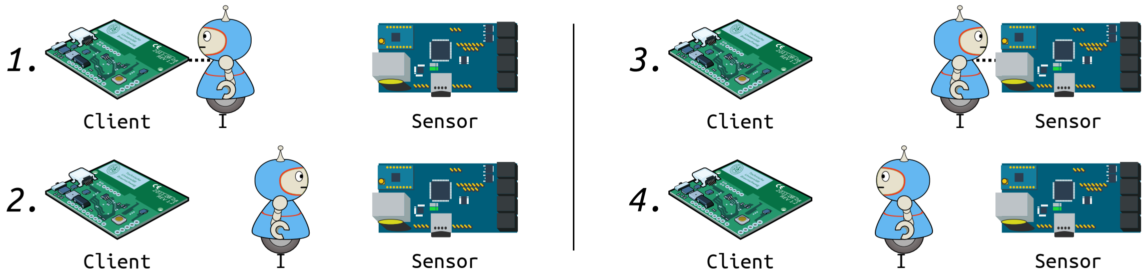


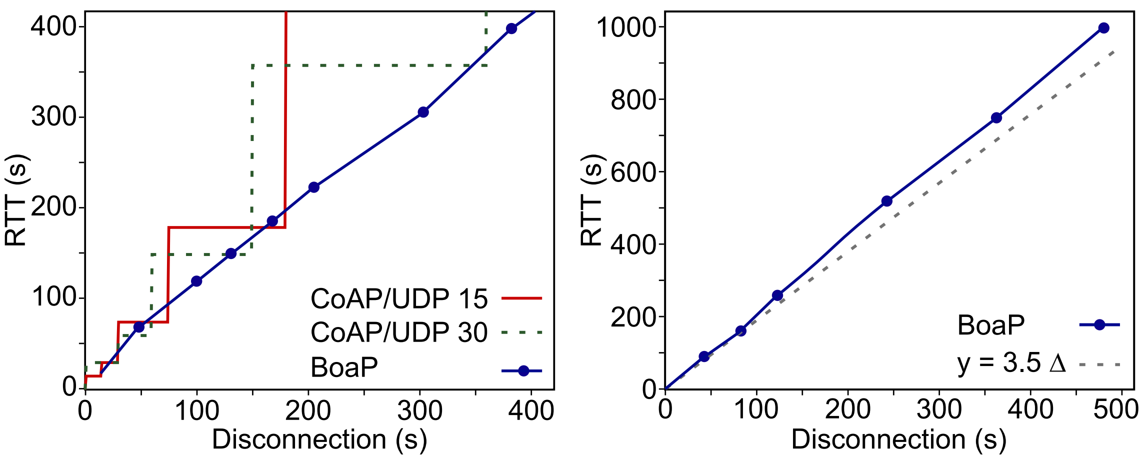
Figure 4.2: BoaP Test: Scenario 2

4.1.2.3 Results of Preliminary Tests

As for the first scenario, the obtained RTTs are displayed in Fig 4.3a. The two curves for Cf (one for each value of ACK_TIMEOUT begin 15 and 30 s) exhibit a stair-like shape, as expected, due to the exponential back-off. However, BoaP is almost linear, with a slope close to 1. BoaP is slightly slower than Cf when disconnections are very short or when the disconnection durations coincide with the retransmission back-off time, yet BoaP is definitively faster than Cf when disconnections stop between two dates of retransmission. That is because BoaP does not wait to transmit requests.

During this experiment the BP beaconing, used to discover the neighbors, was set to 1 second. For fairness, the CoAP parameter ACK_TIMEOUT could have been set to 1 second too. However if the disconnections are expected to be long then the CoAP client would rapidly flood the network each time it needs to send a request and the MAX_RETRANSMIT value would be reached before the link would be up again. In such circumstances the messages are most likely to never be received at all.

As for the second scenario, Fig 4.3b shows the obtained RTT values. With the emission dates uniformly distributed and with the two extreme RTT values being 2Δ seconds and 5Δ seconds then the expected average of the optimal and theoretic RTT is $y = 3.5\Delta$ seconds. In practice, the measured RTT s are slightly longer due to the reconnection cost (beaconing), and the processing of BoaP messages that are neglected in the optimal and theoretic RTT value. These results show that BoaP can perform reasonably even when the client and the server never meet.



(a) Intermittent direct connectivity

(b) No end-to-end connectivity

Figure 4.3: Measures of the RTT obtained in the two tested scenarios

As a conclusion, the Java API used to communicate with IBR-DTN provides reasonable performance and does not slow down the reactivity of BP endpoints that rely on it. As for the test in a small network, it can be said that Cf is fast and can easily overcome short disconnections. However, if long disconnections are to be expected or if a connected end-to-end path between client and server is unlikely to exist, then, BoaP is a better alternative. In other words, the BP can be an effective substitute to UDP as a CoAP binding: BoaP does not largely degrade transmission delays when disconnections are short, and, contrary to CoAP/UDP implementations, it continues to play its role when the connectivity is strongly intermittent or when no connected end-to-end path exist. These preliminary tests were performed on constrained devices: Atom 32-bit processors cadenced at 1.6 GHz with only 1 Go RAM. Even if these resources are a bit more than what the *things* can be expected to embed, RTTs are still very close to the theoretical ones even if the API and prototype are not focused on performance.

Finally, according to these results, it can be attested that BoaP tests verified that the prototype works as expected and offer good performance. Experiments, involving mobility and a bigger number of nodes were conducted and are presented below.

4.2 Experimentations

In the following, experimentations with BoaP deployed in a DTN network of 50 nodes in movements are presented. These movements forces the nodes to connect and disconnect with one another making the network suffers from delays and disruptions. These experimentations aim to attest whether or not BoaP is suitable for DTN networks and that there is not any side effect that would nullified it while keeping good performances. Validating BoaP is an important step for the work of this thesis as BoaP is only the first step to provide a RESTful and SOA middleware on top of it. If BoaP is not suitable for DTN networks there is no reason to design any middleware over it for the DT-WoT.

The experiments environment is presented in the following subsection, next subsection presents the results for each scenario, and more specifically for: *1.* Levy Walk, *2.* Levy Walk with cache, *3.* KAIST traces with and without cache.

4.2.1 Experimentations Platform

4.2.1.1 Swarm

Real-world experiments are financially expensive but provide evaluations of high quality. On the contrary, simulation tools are financially cheap but only offer to model nodes behavior and thus provide evaluations of lower quality. Since a prototype, BoaP, has already been implemented and that the measurements should be as close as possible to their real-world values then simulation tools are left aside. Indeed, simulations do not offer the possibility of running real code, opposed to modelized code. By using a modelized code the benefits of having a developed prototype are lost as measurements would not reflect the prototype code but the modelized ones. Furthermore, simulation tools are usually based on discrete time. Discrete time is both an advantage and a drawback: experiments of several hours can be done in minutes but it actually masks away long operations (such as queries to a database) by shorting them into a single time period. So, in such cases, discrete time actually twists the performance results for the better.

A solution between simulation and real-world experiment has been chosen: network nodes are *virtualize* with containers. Hence, the prototype code is reused, the deployment is cheaper than real-world experiments and also reproducible. Containers scalability lies in the fact that the container kernel is not virtualized nor emulated (opposed to what a virtual machine would do) but actually shared with the host. Thus, while it is not possible to seamlessly run another operating system in a container, a container is far lighter, in terms of RAM and CPU usage, than a virtual machine. The container solution used for these experiments was Docker and more specifically Swarm that is a native clustering tool for Docker offering to use the resources of a pool of Docker hosts abstracted as a sole one. Once coupled with a multi-host networking overlay, Swarm is a powerful, yet lightweight, tool that enables a whole experiment platform.

Figure 4.4 represents the architecture that was deployed for the experiments explained here in a top-down manner. The containers simulate the *thing*s. Each container runs an instance of BoaP and the IBR-DTN daemon, `dtnd`. Added to these two processes, the containers run an `iptable` script that allows, or disables, the communication with other containers according to the mobility pattern.

The plug-in Weave is a multi-host networking overlay that lets the containers share the same LAN and that supports multicast. It runs as a container on each Docker agent. A Docker agent is a host running a Docker daemon and being part of the Swarm.

Underneath the deployed containers is the Swarm environment. Swarm can be described as an orchestrator. The Swarm is composed of a cluster of hosts running Docker daemon pooled together. Then each time the Swarm is requested to start a container it will start it according to a strategy. In the case of the experiment, the strategy that was used is described on the Swarm documentation as: *optimizes for the node with the least number of containers*, i.e. the biggest number of containers running on a host is at most one more than the lowest number of containers running on any host. Thanks to this strategy, the Docker agents resources are then evenly distributed among the containers, and the containers are evenly hosted among the Docker agents.

Under the Swarm environment are the Docker daemons themselves. These ones run directly over the kernel of the cluster nodes. During the experiment each container ran several processes that were:

dtnd the IBR-DTN daemon.

BoaP either a client or server application. Optionally the BoaP cache feature was run on specific nodes for some experiments.

iptable a script that would first DROPs all packets, and then ACCEPTs packets from specific IP addresses according to the mobility pattern.

If Linux containerization has been used for a long time to experiment, using Docker is quite new. EmuStack [123], for instance, is a work that proposes an emulation platform for DTN networks based on OpenStack and Docker. Their platform, unlike the one proposed in this work, differentiates two types of nodes: one, that needs multiple network interface cards, manages network resources, CPU, memory, emulation network. Other nodes, connected to this manager, host emulated nodes. This solution being centralized is not as scalable as the one presented above. It should also be noted that, unlike the platform proposed above, the emulated nodes follow a discrete time which means that the experimentations are not as close as real-world deployment. NS-3[1], a discrete-event network simulator, would enable such architecture. Many parameters (almost a thousands) define the behavior of physical and network layers which make it too complex to be parameterized, for little benefit, according to what is usually done in the experiments. Indeed, the ONE, one of the network simulators accepted by the community, only model node behavior and the communication links are as simple as: when two nodes are connected they can exchange any data without any challenge. Another tool, Common Open Research Emulator (CORE) [124], provides similar features through containerization and emulations networking layer 3 and above. Besides being a very similar tool it provided no benefit to the Swarm solution. Also Swarm solution emulates networking layer 2 and above which bring emulation closer to real-world deployment.

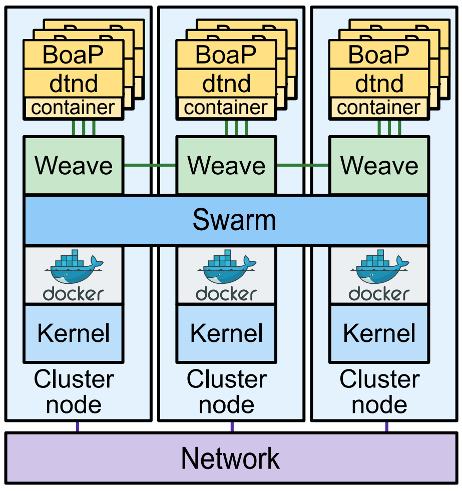


Figure 4.4: Swarm Architecture

4.2.1.2 Performances of the Network and the Application

In order to ascertain that no layer introduce a critical latency the performance of the different layers are assessed. In fact, just like for the Java IBR-DTN API, it is important to attest that the experiment platform does not critically degrade the networking performances. To do so, the following RTTs, presented in Table 4.1, are measured:

Agent Network The RTT named Agent Network is a simple ICMP RTT. There are two measurements presented: one sent from a Docker agent to itself and one sent from a Docker agent to another Docker agent.

[1] https://archive.fo/QLF8y

Swarm Overlay Network The Swarm Overlay Network RTT is also an ICMP RTT. It must be differentiated with the Agent Network one as it done over the Swarm environment through the plug-in Weave. Because a Docker agent hosts several containers then the RTT between two containers sharing the same Docker agent may be shorter than the RTT between two containers hosted on two different Docker agents. For this reason, RTTs are measured as both: loopback (a container pings itself), locally (a container pings another container on the same host), and remotely (a container pings another container hosted on a another host).

It should be noted that there is only two Agent Network RTTs as, at this level, the loopback and local RTTs are the same.

IBR-DTN Network The IBR-DTN Network RTT is measured by using the command `dtnping`, that sends a ping to a destination to which the daemon, on the destination, answers with a pong. Just like Swarm Overlay Network RTT, there is three measurements (loopback, local and remote).

BoaP Request/Response The last RTT measured is a BoaP request/response round-trip-time, again three measurements are presented.

These RTTs were runs ten times the standard deviations are below 13%. On average, a cluster node ICMP ping RTT to itself lasts 0.035 ms and lasts 0.226 ms when the destination is another cluster node. Within Swarm environment, on average, a container ICMP ping RTT to itself lasts 0.048 ms, lasts 0.065 ms when the destination is another container on the same host and 0.456 ms if the destination is hosted on another cluster node. On average, IBR-DTN endpoints ping RTT lasts 1.09 ms when the destination is itself, 2.22 ms when the destination is locally hosted and 2.39 ms when the destination is hosted on another cluster node. IBR-DTN daemons communicate over TCP which is an upper layer than ICMP needs to communicate. This latency is explained by the TCP handshakes and acknowledgements. The last latency measured is BoaP where a whole request is sent to a destination and is answered right away. The RTT is measured just before sending the request and just after receiving the response on the application layer. On average, a local request/response RTT lasts 68.9 ms, 74.6 ms when the container is on the same host and 79.4 ms when the container is hosted on a distant node. This latency can seem big but it can be explained by the communication between BP endpoints and IBR-DTN daemon that is done through a textual TCP stateful communication API.

	Loopback	Local	Remote
Agent Network	0.035		0.226
Swarm Overlay Network	0.048	0.065	0.456
IBR-DTN Network	1.09	2.22	2.39
BoaP Request/Response	68.9	74.6	79.4

Table 4.1: Platform Performance: RTTs Measurements (ms)

It should be noted that if the latencies are too short then the experiment platform is not realistic. Indeed, if the simulated network offers better performances than a real-world wireless network then the simulated network get away from a realistic one. Even though the BoaP RTT appears to be costly it is still acceptable and not critical as the radio interferences and the delay to connect two nodes are ignored.

Performance of the experiment platform, relying on Docker Swarm, is fast enough between interconnected nodes: about 70 ms for a request/response RTT. This RTT value is the minimal RTT in a connected environment. In the experiment platform, a contact duration of one second between two nodes can be utilized to fulfill a request, or at least forward it. In real-world, such fast contact may not be utilized: indeed delay for two nodes to connect to each other with Bluetooth or Wi-Fi is usually longer than one second. These utilized contacts could be part of a long chain of forwarding that allows a request (or a response) to be received. For this reason the platform might not be as realistic as possible. For a more realistic platform these short contacts should be ignored, for an even better realism all the contacts should be reduced of the connection delay. Another solution to this challenge of realism could be to modify the radio range. Indeed, with a shorter radio range then the short contacts would be deleted. Nevertheless this solution implies two other challenges: the contacts that were a bit longer than these short contacts become short contact themselves, and the radio interferences are still ignored.

4.2.2 Scenarios

Mobility patterns, node behaviors and processing of connections history are detailed here to better understand the following experiments. There are two different node mobilities used for these experiments:

Levy Walk Mobility Levy Walk, presented by Rhee *et al.* in [125], is a mathematical model based on statistic of human mobility and simulates it better than Random Way Point mobility. Indeed, as exposed in experiment reviews, see 4.2.2.2, several studies tried to correct, or at least compensate, its known flaws. In the experiments, 50 nodes move according to this mobility within an open squared area during one hour. Three different densities (high, medium and low) are deployed by modifying the size of the area while conserving the same number of node. These three densities were empirically chosen after a set of experiments with different values and are in the range of the various experiments reviewed in 4.2.2.2. The different densities actually force three different kind of networks. The different experiments are named:

- **LWHD** In the Levy Walk High Density scenario there are very rare disconnections.
- **LWMD** The Levy Walk Medium ario is characterized by more frequent disconnections.

LWLD As for the Levy Walk Low Density scenario, the connections opportunity are rarer and contact durations shorter.

KAIST Mobility KAIST mobility is based on real-world GPS traces, [126]. This dataset provides 92 movement GPS traces of students in the KAIST campus (Daejeon, Korea). This is one of the dataset of CRAWDAD that is not focused on the occurrence/duration of contacts but in effective human mobility. While most node speed is close to human walk speed, some have a higher speed and represent student mobility in cars. Unlike Levy Walk the nodes are not in an open area but follow defined paths, such as roads, and are not as uniformly distributed as Levy Walk in the area. The graphical representation of these traces, likened to a heat map, over the world map can be seen in the Fig. 4.5. This experiment scenario is named KAIST.

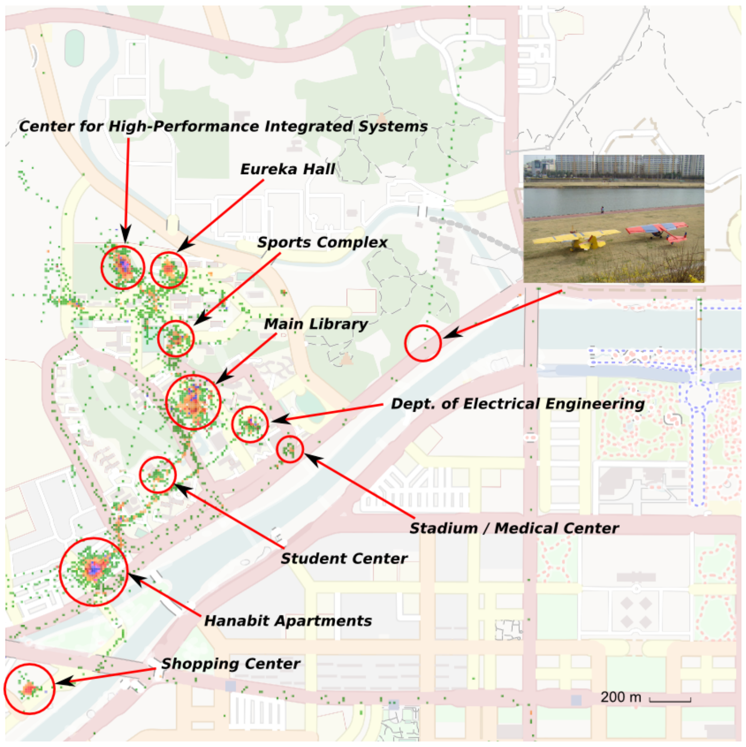


Figure 4.5: Graphical representation of KAIST traces

Before executing the experiments a processing is required to go from the nodes positions (either simulated with Levy Walk or GPS-based from KAIST) to the nodes connections and disconnections. From the nodes positions and their radio range, the connection history is calculated for every single node. For each node, identified by an IP address, and at each position, the reachable nodes are listed creating the history of connections and disconnections. When a node comes in range of another node then its IP address is enabled to accept the traffic from and to it. When a node leaves the range of another node then its IP address is set to drops the traffic from and to it. A script is created for each node, this script drops or accepts the packets from the specified IP addresses simulating the network as it would be if the nodes were really moving accordingly to the positions. This history of connections is put into a script using `iptable` Linux tool.

In the experiments, a node is a device being part of an intermittently connected network in which the BP is the transport protocol. Among the 50 nodes, there are clients, servers and regular nodes. The clients request resources hosted on the servers, the servers answer these requests. As for the regular nodes, they do not request nor answer message though they actually relay messages thanks to their cache and the store-carry-and-forward concept implemented by IBR-DTN. For the experiments, the percentage of clients and servers are set to 20% and the rest are regular nodes. Lacking of real-world DTN applications, these ratios were empirically chosen after a set of experiments performed with different percentages. Note that the results of these experiment are not presented in this document and match the range of client/server ratio of various experiments reviewed in 4.2.2.2.

The clients behavior aims to model a smartphone usage but the lack of study on the smartphone usage and network behavior[2] forces the model to be intuitively designed. Two behaviors aim at reproducing two different kind of applications usage that are automatic requesting, e.g. background requesting, and on demand requesting, e.g. web, email, instant message service. More specifically, the two behaviors are as follow:

Automatic At start, between 3 and 7 servers are randomly selected. Each client sequentially requests one server after another at a **fixed** period of P seconds, P being randomly picked **only** once between 30 and 300 seconds.

On demand (human action) At start, between 3 and 7 servers are randomly selected. Each client sequentially requests one server after another after a **variable** period of R seconds, R being randomly picked **each time** between 60 and 120 seconds.

All clients stop sending requests a while before the end of the experiment. In the Levy Walk scenarios, that last 60 min, the requests sending is stopped at 15 min before the end of the experiment. This is done to let time to the network to disseminate requests and responses to reach a stable state at the end of the experiment.

4.2.2.1 Quantitative Characterization of the Scenarios

A quantitative characterization of the scenarios is provided below. This characterization aims to put experiment results in perspective, to provide the key to understand

[2]Studies, such as [127], focus the psychological traits and compulsive behavior related to the smartphone usage but not the networking behavior of smartphones.

these results and to compare the four scenarios, e.g. especially finding a Lewy Walk scenario comparable to KAIST scenario. The Tab. 4.2 presents the four scenarios with six characteristics that are:

Square Side The square side defines the length, in meter, of the square enclosing the nodes in their mobility. While it can easily be determined for Levy Walk scenarios, since it is a parameter, it must be noted that for the KAIST scenario defining the exact limitations or the enclosed area is mostly empirical: the traces are not contained in a well defined square and removing areas where there is no traces is complicated as it tends the area to be only composed of few buildings and roads.

In the following experiments, the square sides vary between 500 and 2500 m. The KAIST area is estimated between 2000 and 2200 m.

Node density The node density is directly calculated according to the number of nodes in the experiment, which is a constant: 50, divided by the area of the experiment. Assuming that all nodes have the same radio range this metric seems a natural one to estimate the connectivity and stability of an ICN. Indeed, the number of network links logically increases with the node density.

Experiments densities go from 8.0 x 10^{-6} to 2.0 x 10^{-4} nodes/m^2 for Levy Walk experiments, as for the KAIST one it is estimated between 1.25 x 10^{-5} nodes/m^2 and 1.03 x 10^{-5} nodes/m^2 which places it between LWMD and LWLD.

Coverage Ratio Assuming that the nodes are uniformly positioned in the network area, the coverage ratio[3] aims to represent network sparseness. This ratio is calculated by dividing the $covered\_area$ by the $experiment\_area$, where: $covered\_area = number\_of\_nodes * (\pi * radio\_range^2)$. When this metric exceeds 1 it induces that all the experiment area can be covered by the wireless communication. This metric depends on the area size itself, the number of nodes in the network and the radio range.

In the experiments, the different node densities differ by two order of magnitudes. This directly affects the coverage ratio that goes from 157% down to 6.3% for Levy Walk scenarios. These values may seem very far away from each other but they were chosen after many experimentations to select three different kinds of networks being: stable/reliable, mostly disconnected and rarely connected. The KAIST coverage ratio is between 8.11 and 9.82%, as expected its value is within the range of LWMD and LWLD just like node density and square side values.

Average degree of node against time The average number of network connections per network node, or average number of edges per vertex, or average

[3]The coverage ratio presented in this document does not have any similarity with its financial homonym: the measure of a company's ability to meet its financial obligations.

degree of node, aims to give an overview of how much the network nodes are actually interconnected. If the average degree of node is high during all the experiments then it can be assumed that the network is meshed. On the contrary if this average is very low then it means that the nodes rarely meet one another.

Tab. 4.2 shows that a lower density allows fewer links per node. Indeed, in LWHD the number of network links does not fall behind 16 and goes above 25 which represents half of the nodes. In LWMD, the number of links is between 1.2 and 2.4, as for LWLD the number of links does not exceed 1.7 and sometimes falls under 1 which implies that most nodes are isolated. It should be noticed that for this characteristic KAIST is not between LWMD and LWLD values. KAIST average degree of node is not really comparable to other Levy Walk scenarios and is greater than LWMD values despite its lower node density and lower coverage ratio.

CDF of contacts durations The Cumulative Distribution Function (CDF) of contact duration shows the percentage of distribution of the contact durations. With longer contact durations, a network can be considered as more stable. In contrast with shorter contact durations, a network can be considered as more dynamic. Also, for the same reasonably low-density, the faster the nodes move, the shorter the contact durations will be but they are likely to be more frequent too.

CDF of contact durations are shown in Tab. 4.2. As it can be seen, the CDF of contacts durations do not greatly differ as both three scenarios use the Levy Walk model. As for KAIST scenario there are more long contact duration as the experiment lasts longer but, just like Levy Walk experiments, 90% of the contacts durations are below 10 seconds..

Contact matrices To better understand the difference between following results a means to visually compare the contact behaviors in the four experiments was sought. The contact behaviors can be measured and analyzed in many ways and a 2 dimensional graphic is proposed here. A contact matrix represents both the cumulative link duration of each node with every other node and the number of contacts of each node with other nodes. The matrices in Tab. 4.2 represents these different data in a single matrix as follow:

- the lower-left corner of a matrix represents the cumulative contact durations of each node with other ones: the longer two nodes are in contact, the whiter the color is. The contact durations are expressed in percentage of the whole experiment timespan, i.e. to reach 100% of contact duration two nodes must be in the range of each other during the whole experimentation. On the contrary two nodes that never meet will share a black color.
- the upper-right corner of a matrix represents the number of contact of each node with every other ones: if the number of contacts increases then

the gray-scale color becomes whiter. The number of contacts between the nodes are normalized: the biggest number of contacts between two nodes is represented as a white value (100%) while the least number of contact is represented as a black one (0%), therefore it cannot be used to relatively compare two different experiments but the emerging patterns still provide hints to understand the node interactions at a network level.

The LWHD matrix is undoubtedly lighter than the others. Since the Levy Walk mobility is supposed to model the human mobility and since the KAIST density and coverage ratio are enclosed in the range of LWMD and LWLD values it could be expected that same patterns are shown by the three matrices. Nevertheless KAIST matrix is clearly lighter than the others and specific patterns are shown. This difference can be explained by the average degree of node that already differs from Levy Walk experiments.

The experiments were performed at least twice to verify no error occurred during: thanks to the experiment platform the experiments are close to be reproducible and results do not differ over 3%.

4.2.2.2 Experiments Review

An experiments review has been done to define the experiment parameters in coherence with what the community usually does. 86 articles are referenced, see A.1. This review highlights the wide ranges of parameters values that are used among very different scenarios deployed for various, diverse objectives.

Very few real-world experiences occurred, two famous ones both involve animals, [97, 98], and very few emulations are done in this field. In fact, experiments are usually done with discrete-time simulations. An interested reader will find more details in the similar, yet deeper, work done in [128].

Tab. A.1 references 86 papers that presents simulations or emulations. In an attempt to compare and present what is usually done, some quantitative characteristics are extracted from theses papers. These quantitative characteristics are the area size of the simulations, the radio range of the devices, the number of nodes, the coverage ratio, the node mobility, the simulator used and the experimentation duration. The number of quantitative characteristics available for the different study varies a lot: some only name the simulator tool and others provide in depth details of the simulation set of parameters. Moreover, in the referenced works, some used CRAWDAD [129] traces where the number of nodes is not always written down in the paper, the radio range often not available and the area not well defined.

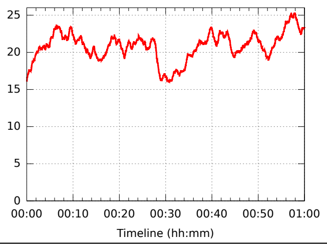
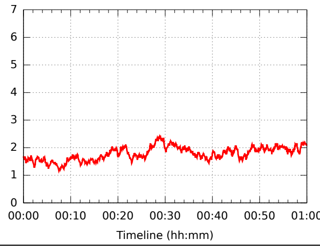
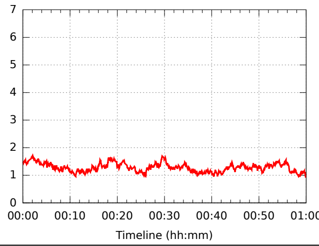
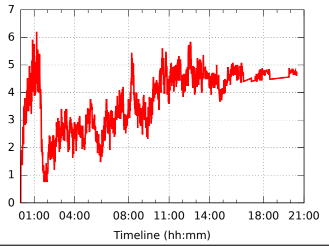
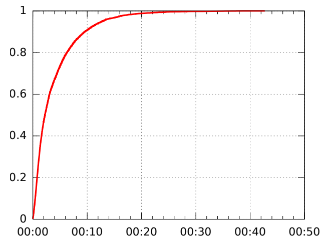
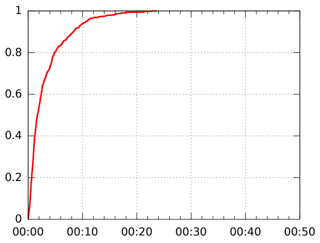
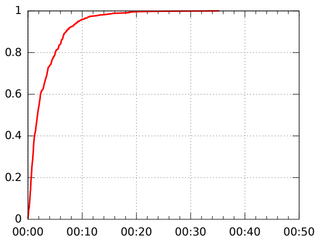
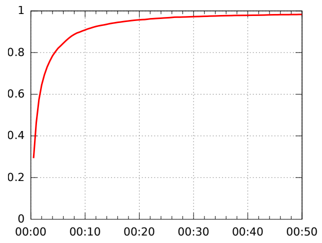
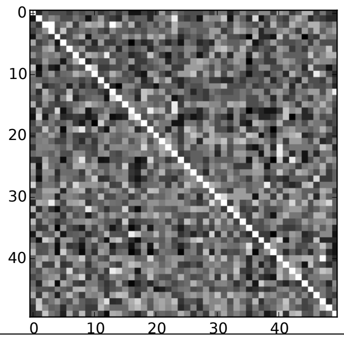
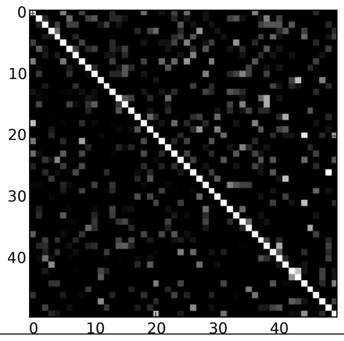
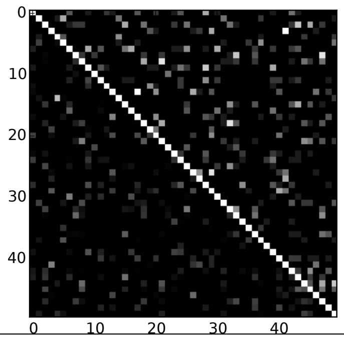
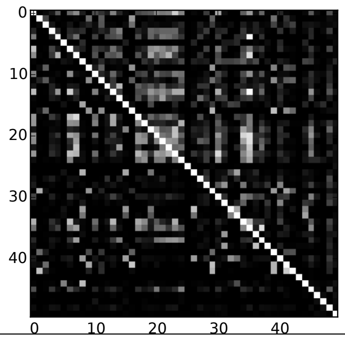
Experiment name	LWHD	LWMD	LWLD	KAIST
Square side (m)	500	1750	2500	2000 $\sim$ 2200
Square area (km^2)	0.25	3.0625	6.25	4 $\sim$ 4.84
Node density (nodes/m^2)	2.0 x 10^{-4}	1.6 x 10^{-5}	8.0 x 10^{-6}	1.25 x 10^{-5} $\sim$ 1.03 x 10^{-5}
Coverage ratio (%)	157	12.8	6.3	9.82 $\sim$ 8.11
Average degree of node against time				
CDF of contacts durations				
Contact matrices (lower-left corner: cumulative contact duration, upper-right corner: number of contact of each node with every other ones)				

Table 4.2: Quantitative Comparison of Scenarios

As stated in [128], only 15% of the published paper on experiment are repeatable and 12.1% mention the simulator version used. The reproducibility of the experiments presented in this thesis is in one of the focus of this work. The credibility of the experiments was also one of the objectives and Tab. A.1 aimed to give an overview of the parameters ranges usually chosen so the parameters selected for this work were not out of band and more likely to be realist. It is not claimed that the node mobility, the Levy Walk or the KAIST traces, node behavior and network link are all very realist but it is the closest this work could be despite the "poor state of simulation based studies in the network community", as wrote Kurkowski *et al.*.

Among the papers review in Tab. A.1 only one study did not use a discrete time simulator tool. Only 8.3% studies used several traces, and as few as 11.7% used CRAWDAD traces. Among the 75% studies that used mathematical models to generate the nodes mobility, 82.2% used the Random Way Point despite its known flaws. Several papers though applied some tweaks to correct these flaws.

4.2.3 Results of the Levy Walk Experiments

In Fig. 4.6 three metrics are represented against time on each graph. Each metric represents a number of messages:

- the number of sent requests (from the client to the servers),
- the number of received requests (that is also the number of sent responses),
- the number of received responses.

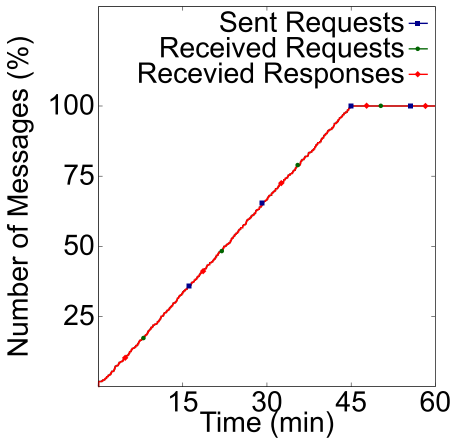
In the first experiment, see Fig. 4.6a, LWHD, all requests are received by the servers and all responses are received by the clients. In this experiment the density is so high that the network can be likened to a connected MANET within which a routed end to end path exist between every nodes in the network. The messages are actually received so fast that the three plots overlap.

In the second experiment, see Fig. 4.6b, medium density, 91% of the requests are received by the servers. Among these received requests, and produced responses, 12% will not be received by the clients. Overall 79% of the sent requests will receive a response.

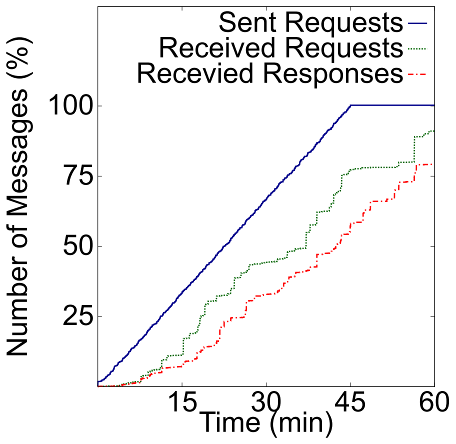
In the last experiment, see Fig. 4.6c, low-density, 67% of the requests are received by the servers. Among these received requests, and produced responses, 19% will not be received by the clients. Overall 49% of the sent requests will receive a response.

In Fig. 4.7 the two graphics represent the comparison between the different experiments. The Fig. 4.7a represents the number of received requests in the three different experiments, as for the Fig. 4.7b it represents the number of received responses in the different experiments. It is clear that with a lower the density the number of connections decreases and the message dissemination is challenged. This is why the number of received responses is higher when the density increases.

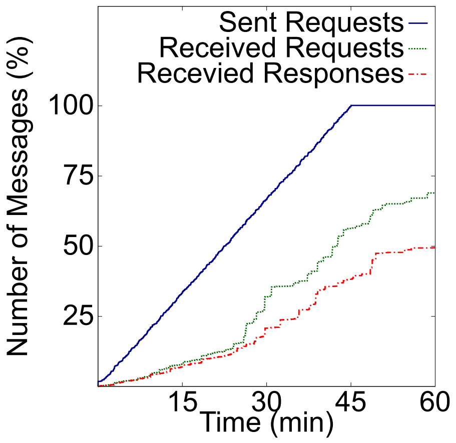
These results are coherent with the intuition given by the quantitative characteristics. In fact, as expected, if the area becomes bigger then the density lowers and with a lower density the links are rarer so the exchanges are less frequent due to the fewer opportunities. This is why the ratio of delivered responses decreases when the node density decreases. It can be noticed that the curves representing received requests and received responses share similar form with the received responses one being delayed. This is explained by the fact that a response cannot be received if the request was not received beforehand and that the more requests are received the more responses are received too. It can also be noticed some steps in received messages, both requests and responses. These steps are explained by the network nodes getting closer allowing messages to reach their destination.



(a) LWHD

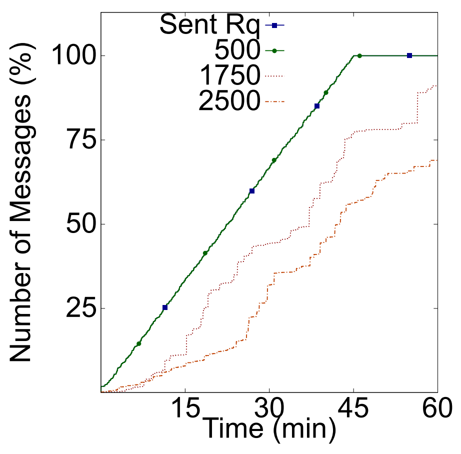


(b) LWMD

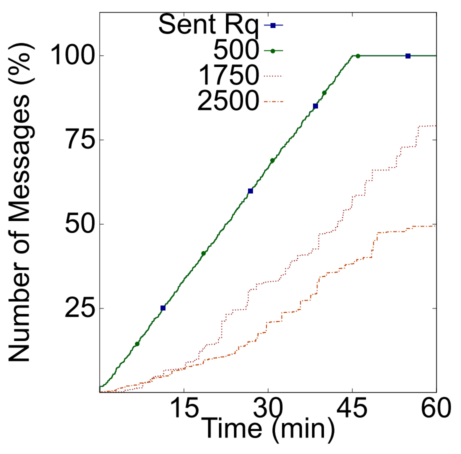


(c) LWLD

Figure 4.6: Levy Walk mobility with 50 m radio range, without BoaP caching



(a) Received Requests



(b) Received Responses

Figure 4.7: Levy Walk, 50 m radio range, without BoaP caching: Comparison

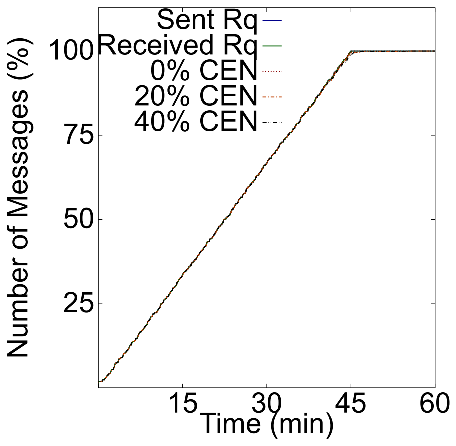
4.2.3.1 Enabling Cache Feature

In the experiment results presented below, the caching feature is enabled. The caching feature enables intermediate nodes to answer a request if they carry a response that is still fresh. Each of the previous experiment has been deployed twice again with some Cache Enabled Nodes (CEN). The first experiments had 0 CEN, the Fig. 4.8 presents these experiments with with 20% and 40% of CENs in the networks.

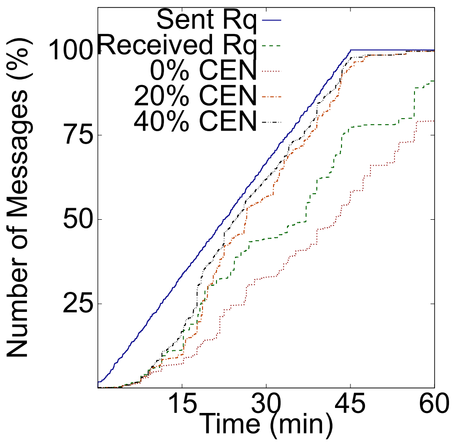
In Fig. 4.8a, as the network is closer to a fully connected MANET than an ICN, there is actually no difference with and without CEN: 100% of the requests are answered, all the lines overlap in the graphic and more than three fourth of the requests are answered in less than 0.5 second. Nevertheless, in the Fig. 4.8b and 4.8c the effect of the cache feature becomes clearer. In the LWMD, the percentage of received responses without any cache is 79%. Introducing 20% of nodes as CENs in the network brings this percentage up to 99.6%. The number of received requests stays the same while only 90% of requests has been received by the server. Having more responses received by the client than requests received by the server proves that the cache feature not only works but also improves the number of fulfilled requests. Indeed, 9% of the requests has not been received by the server but has been answered by the CENs directly.

It is clear than going from 0 to 20% of CENs improves the number of received responses. In LWMD and LWLD the number of received responses are respectively increased by 20.6% and 36.4%.

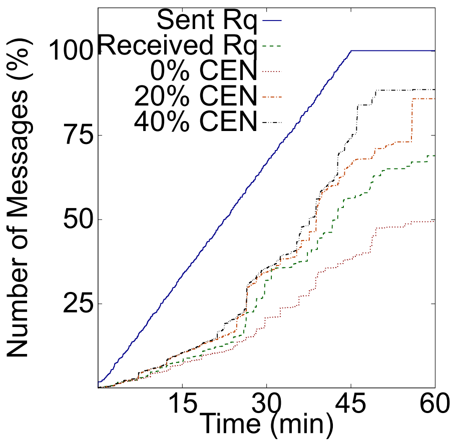
However, increasing from 20% to 40% of CENs does not always improve the number of received responses. For instance there is no improvement in the LWMD between 20% and 40% of CEN. However, as it can be seen in the Fig. 4.8b, the responses are received faster. In like manner, the LWLD does not show a big improvement on the number of received responses, yet again the responses are received



(a) LWHD

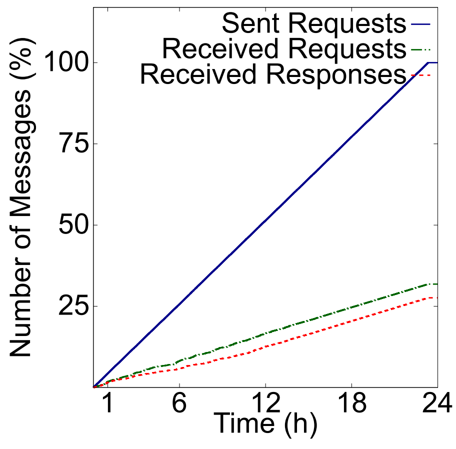


(b) LWMD

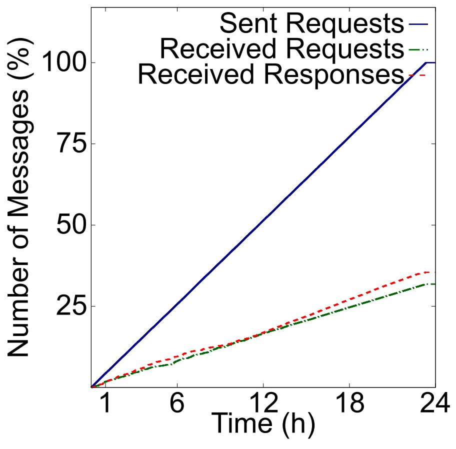


(c) LWLD

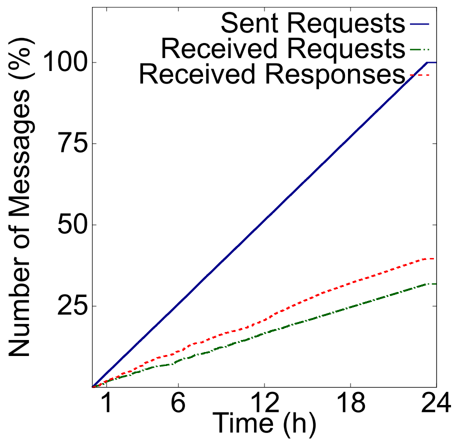
Figure 4.8: Levy Walk, 50 m radio range, with Cache Enabled Nodes



(a) 0 CEN



(b) 20% CEN



(c) 40% CEN

Figure 4.9: KAIST Experiments Results

faster.

4.2.4 Results of the KAIST Experiments

Just like the previous experiments with Levy Walk some nodes are clients, some are servers and others are regular nodes. Among the regular nodes a varying percentage are actually CEN, this percentage varies from 0 to 40%. For the same reasons as before, the clients stop sending any request 40 minutes before the end of the experiment to reach a stable state. Also, the bundles lifetime is set to 60 minutes, which is the default.

Fig. 4.10 depicts the results of KAIST experiment: GPS-based mobility, involving either 0, 20 or 40% CENs. In all experiments, a bit less than 32% of the requests have been received. In the first experiment that does not involve the cache feature, 27.6% of the response are received, which means that 4.2% emitted responses are not

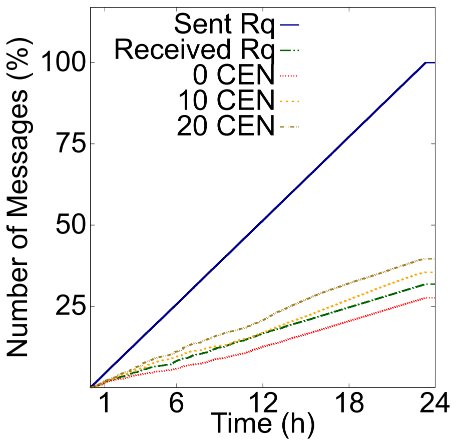


Figure 4.10: KAIST Comparison

received. The number of received responses corresponds to 86.72% of the received requests. In the second experiment, with 20% CENs, the percentage of received responses increases to 35.46%, which is 111.4% of the received requests. Indeed, thanks to the cache feature, the number of received response has been improved by 28.4%. As for the last experiment, with 40% CENs, the number of received responses increases to 39.6%, 124.42% of the received requests. The number of responses has been improved by 43% compared to when no CENs were in the network.

Fig. 4.10 shows the three different experiment merged into one graphic. It highlights the fact that the number of received responses is improved when the number of CENs increases.

The KAIST scenario is compared to the LWLD as their coverage ratio are the closest. Also, since the Levy Walk model reproduces the human mobility, similar results are expected between LWLD and the KAIST scenario. Because KAIST experiment lasts more than 20 hours, unlike Levy Walk ones that only last one hour, then to compare BoaP performance between Levy Walk and KAIST scenario it must be done through the first hour of experiment. Moreover it must be clear that bundles lifetime is set to one hour in all experiments, which means that no bundle expire during Levy Walk experiment but they expire in less than 1/20 of the KAIST lifespan. Focusing on the first hour of the experiment, with 40% of CEN, the KAIST scenario only reach 45% of received responses while the LWLD reaches about 75% of received responses.

Several facts explain this difference. First, differences between Levy Walk and KAIST experiments are highlighted by the quantitative characterization: even if the node density of KAIST is enclosed between the node density of LWMD and LWLD experiment, it must be noticed that the average degree of node differs a lot. Second, according to the matrices there are hints that KAIST nodes are part of groups that move together. This may be the social aspect of the mobility that might not be well reproduced in the Levy Walk model, or is enforced in KAIST traces. Finally, the Levy Walk model generates the node mobilities in an empty area while real-world traces follow paths such as the streets. These differences lower the chance of a message being disseminated over the network as it is more likely to stay in a group till its lifespan expires.

4.3 Conclusion

In this chapter all the results were treated from a point of view where clients send requests that are received, processed and answered by the servers. The number of sent responses, received responses and received responses are compared between one another. There is another way to analyze these results. If we assume that the clients are actually servers and that their requests are actually notifications (requested through the OBSERVE option even before the start of the experimentation) then the number of received requests in the figures can be considered as received responses or notifications. The number of received responses can either be ignored or considered as acknowledgements. This shift in the point of view of the data is easily explained, mobility put aside, in a real-world use-case: a smart city [130] where the different resources (noise, light, temperature, humidity pollution, and so forth) are logged into the city hall. It must be clear that with this shift of point of view the results, and experiment, with the cache feature enabled should be ignored. Indeed, the requests of the experiment are considered as OBSERVE-notifications and the cache feature do not cache the requests. Considering this shift, the number of received notifications in the LWMD scenario goes from 79% up to 91% and, in the LWLD scenario, goes from 49% up to 67%. In the LWHD scenario, it stays at 100%. As for the KAIST scenario, the number of received notifications goes up 27.6% to 32%.

After these experiments and results analysis, it can be concluded that BoaP, when the nodes mobility is at the advantage of message exchange, performs well in small and larger network. This conclusion seems a natural one since BoaP relies on BP and that nodes' mobility, with radio range, are the main two parameters that impact message exchange opportunities. Despite the cache mechanism there is no apparent risk to its scalability for even bigger network. With the insight provided by the experiment, BoaP do not seem as optimized as it should be. Its limits lies in the layered architecture of BP that do not allow bundle manipulation as precise as needed. For instance, the caching mechanism could be enhanced: no response would be sent if a node already carries a response for a request it was about to answer. Healing mechanisms could even be deployed. In other words, even if the layered architecture proved to bring many benefit it actually restricts applications above BP overlay. This is unfortunate since, in DTN networks, optimizations could save up power and battery and thus lengthens lifetime of *things*.

5
LILWENE

Contents

This thesis objective is twofold: how to provide both resource-oriented and service-oriented programming supports to enable a Delay-Tolerant Web of Things (DT-WoT). A resource-oriented approach has been proposed by adapting CoAP over BP. This proposition has been validated by its protocol evaluation and will be used as a core-base of the next proposal.

In order to fulfill the second objective of this thesis, a last contribution is presented in this chapter named LILWENE: Light services middleware for delay tolerant Web-of-things Networks. This proposal, RESTful and relying on BoaP, natively offers additional means such as service description, service discovery/advertisement and service invocation compared to a resource-oriented approach.

All theses mechanisms are presented in depth in this chapter that is introduced by a differentiation between resource and service. This contribution also contains four design principles, likened to the REST constraints, that aim to give guidelines when design a service-oriented framework. Finally, to give a technical understanding of LILWENE, an API as well as implementation elements are also presented.

5.1 Overview

LILWENE aims to provide more features than a resource-oriented framework, such as BoaP, by following a service-oriented approach. First of all, differences between resource and service must be cleared. W3C, in [10], defines a resource as an "*item of interest, in an information space, identified by a* URI". So far, with BoaP, the resources were considered as a simple self-content item of interest. Service-oriented approach covers additional features such as description of services (i.e., to inform on the needed details of their capacities and functionalities to request them), semantical descriptions of services, advertisement and discovery of services, and service composition.

To support these features, an abstraction layer can be put on top of BoaP to treat the resources as services in a service-oriented approach. LILWENE proposal provides these features, and more specifically focuses on the description, advertisement and discovery as well as requesting mechanisms. Semantical description and service composition are out of the scope of this work.

In other words, LILWENE answers to questions such as *how to describe the services?*, *how to discover them?*, and *how to invoke them?*. Several propositions are exposed to answer each one of these questions while remaining consistent with BoaP, staying aware of DTN constraints and continuing to be friendly with constrained devices that populates the IoT.

5.1.1 Proposal

As explained before, REST approach follows 6 constraints: loosely-coupled clients/servers, stateless servers, servers are accessed through uniform interfaces, the system must be layered, and responses can be cacheable. The REST-architecture constraints provide several advantages. LILWENE, just like BoaP on which it relies, follows these constraints to benefit from these advantages.

In like manners, four design principles were identified for the solution presented here. These design principles aim to optimize reactivity and efficiency of a SOA middleware in DTN environments such as LILWENE, by taking into account both DTN and SOA challenges. These principles are:

Local Registry Since there is no node in the network assumed to be well connected, stable enough or even reliable then each node must host its own local registry. This principle stems from network environment and constraints.

Full and Concise Description Being in a DTN environment, the number of exchanges must be kept to its minimum. For that reason, the discovery must be done with the minimal number of messages. Because of this, service descriptions must enclose all required details to issue successful request at first attempt. This principle prevents waste of energy and storage for disseminating unnecessary messages.

Correction Each client application must cooperate with its own local registry in order to modify service descriptors locally stored that are assumed to be incorrect. A descriptor can be considered incorrect according to its error-responses, for example if it answers with the code *5.01 Not Implemented* to a request sent with an HTTP method wrongly advertised as implemented. Doing so prevent clients to send future requests that would result as an error. This principle prevents waste of energy and storage for disseminating unsuccessful requests.

Update In the same way, registries must also cooperate with their local clients and ensure that clients are aware of any descriptors update. This principle enforces reactivity and can prevent unsuccessful requests too.

These four design principles aim to minimize the number of exchanges and errors by providing directives and guidelines to the registries, clients and service providers. These design principles ensure that local applications cooperate and correct themselves but also that the medium usage of the network and node battery are not wasted. These constraints are present throughout the whole chapter.

5.1.2 LILWENE Presentation

Fig. 5.1 illustrates the LILWENE overview presenting interactions between the different actors of the platform that are: client applications, service providers, their

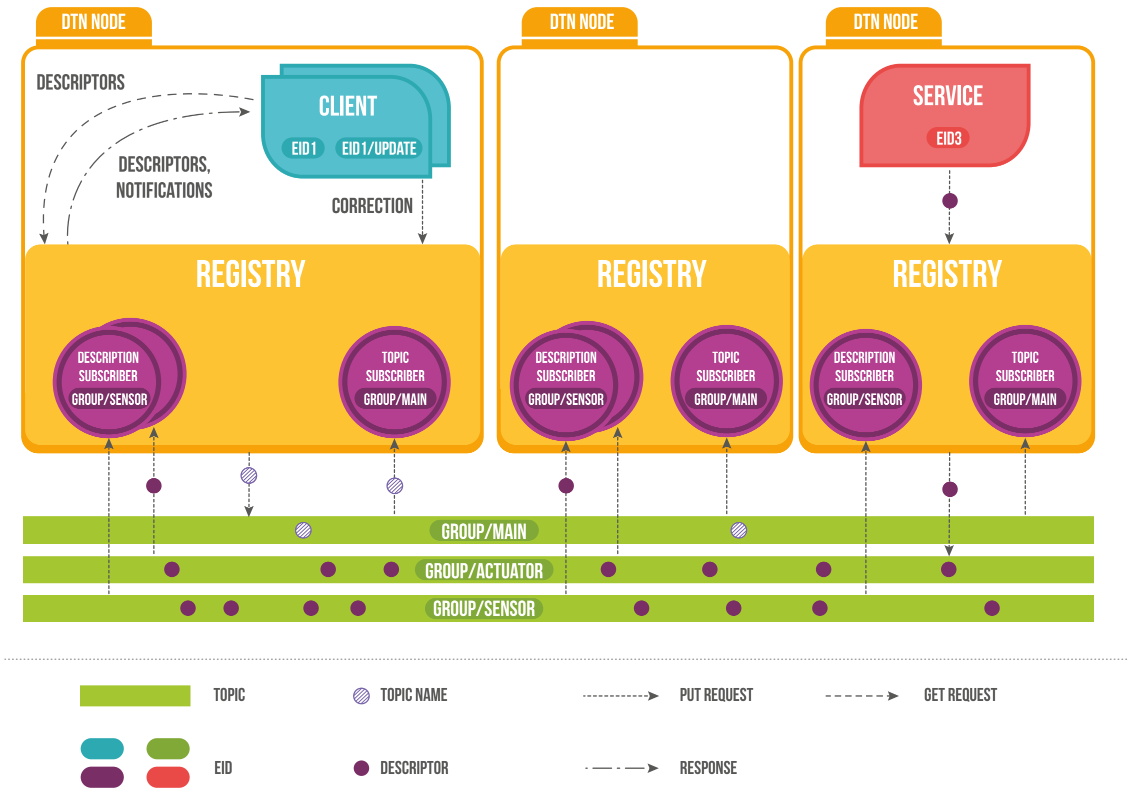


Figure 5.1: LILWENE Overview

description documents (descriptors), local registries, their subscribers and topics. The platform is described in depth in the remaining of this chapter. Nevertheless, a brief presentation at a high level may help the reader to apprehend the whole platform.

A LILWENE node can host zero or more client applications, zero or more services and must host one registry. The services providers, at their creation, send their descriptor to their local registry. Registries take full care of the advertisement and discovery mechanisms through a publish/subscribe interface relying on topics. Clients request their local registry to discover descriptors by expressing their needs. The clients can then request (e.g., invoke) the services providers.

Services

A service, or service provider, is an application that can be requested in order to produce an outcome. Services may be co-hosted on LILWENE nodes with client applications. In LILWENE, a service provider is identified by its unique BP EID.

At its creation, a service must send its descriptor to its local registry. Its local registry takes care of its advertisement. In Fig. 5.1, the service sends its descriptor to its local registry.

When a service plans to stop its activity, it must send a specific descriptor to its registry. The registry advertises it to inform clients that the service is not accessible and shall not be requested anymore. Additionally, if the service has been invoked by a little number of clients, the service may send this descriptor directly to these clients.

Clients

A client is an application that discovers service providers thanks to a registry and that invokes service providers. An application can be both client and service. Also, just like service providers, a client is identified by its unique BP EID. In Fig. 5.1, this EID is identified by `/eid1`.

If a client receives a service descriptor, invokes this service and receives an error-response then a test must be run to determine if this error could be prevented by modifying the service descriptor. If it can, then a correction must be sent to the local registry to modify the descriptor. This modification prevents local clients to send a similar request to the same service that would automatically result as an error-response.

Since a client can also be a service provider, it is important to differentiate messages for internal management (e.g., reception of service descriptors) and messages from other endpoints (e.g., either client's requests or service's responses). Processing messages for internal management is common to all LILWENE clients, whereas messages from other endpoints are specific for each client. To ensure that the messages for internal management are separated from other messages a second BP EID is built by appending `/registry` to the first BP EID. In other words, clients have then two different BP EIDs: the first is used to invoke the services, to receive services' responses and clients' requests, while the second one is used to communicate with the local registry and thus receive service descriptors or registry updates and to send corrections.

Topics

LILWENE topics are exploited by LILWENE registries to advertise and discover service descriptors through a publish/subscribe, or pub/sub, interface. A topic is a logical channels identified by, and named after, a non-singleton BP EID. A registry advertises a service by publishing a descriptor into a topic and can discover services by subscribing to one, or more, topic.

Topics can be dynamically created. Having several topics enables a categorization of service descriptors allowing a more flexible means to the discovery. Registries can choose which topics they subscribe to according to the needs of their client applications. However, registries must know all available topics. To ensure that all registries discover all topics then each new topic name is advertised into a main topic: `dtn://group/.well-known/discovery`. All registries must at least subscribe to this main topic. In a degraded mode, a list of available topics can be embedded on LILWENE nodes and registries could be limited to these topics. In such case, the main topic becomes useless. In a last resort, all topics can be merged into one, the main topic, that will be used to publish all service descriptors.

In Fig. 5.1, there are three topics. The main one, identified by `group/main`, contains the names of the two others identified by `group/actuator` and `group/sensor`. These two topics have several service descriptors that can be discovered by registries as long as they subscribe to these topics.

Registries, Advertisement and Discovery

LILWENE registries take in charge the advertisement and discovery of service providers. To do so they exploit the pub/sub interface to disseminate and gather service descriptors. Service providers interact with registries at two different moments. First, when a service is created and sends its descriptor to its local registry so the registry can advertise it, second when a service stops its activity and sends a specific descriptor to inform that the service should not be requested anymore. Such case is illustrated in Fig. 5.1, where a service sends its descriptor to the registry that publishes it into the topic `group/actuator`. Registries interact more with client applications. First, clients request registries to discover services, second clients send corrections of erroneous service descriptors to prevent future request that would result as an error-response. Finally, registries also notifies descriptors modifications, updates to clients as well as new descriptors that may interest a client. These interactions are illustrated in Fig. 5.1 on the left node.

A local registry is accessible to the BP EID that it must register in to its BP daemon: `dtn://node-id/.well-known/registry`. This EID is used to communicate with client applications. However, having only one EID is not enough, indeed topics are also BP EIDs and in order to subscribe to a topic a registry must have at least one other BP EID: `dtn://group/.well-known/discovery`, that is the main topic. Additionally, a register will have one more EID for each subscription to a topic. For instance, in Fig. 5.1 the two left-nodes registries have three EIDs: one singleton each and two subscriptions, whereas the right-node registry only subscribe to two topics: the main one and `group/sensor`.

LILWENE supports two kinds of discovery:

proactive Registries can publish/advertise, on behalf of services, descriptors into a topic. This is considered as a proactive discovery as these descriptors will be disseminated over time and other nodes only need to subscribe to a topic to receive these.

reactive Another kind of discovery is the reactive one. In contrast to the proactive discovery, registries in need for new descriptors must publish a request into the main topic. This request is received by other registries that can answer with their matching-stored descriptors, if any.

It is the duty of the registries to determine if a descriptor should be published and if a request should be sent.

Application Programming Interface

```java
public interface LilweneClient {
    public void create(String eid, ClientHandler handler);
    public void discoverServices(Needs needs);
    public void requestService(Descriptor descriptor, Request request);
}

public interface LilweneService {
    public void create(String eid, ServiceHandler handler, Descriptor
     ↪  descriptor);
    public void update(Descriptor descriptor);
    public void update();
    public void stop();
}

// The registry is a required tool and does offer an API since its processing is
 ↪  transparent.
```

Listing 12: LILWENE API

List. 12 shows the two java interfaces that developers must implement to program LILWENE clients and service providers. There is not any API method for the registry. Indeed, this entity is required and must remain transparent for the developers using LILWENE. Nevertheless, some of registry processings are shown in the section that presents the API and implementation elements.

On the client side there are three methods. One for creation, another to discover services and the last to request services through their REST interface. The first method, to create a client, requires the URI-path of the client URI and a handler to process service's responses. This method also creates a second endpoint, by appending `/registry`, with a specific handler, to communicate with the local registry. This second endpoint is used to request the registry, process the registry notifications and send descriptor corrections. To discover services, a client express its needs in the variable `needs`. It is an implementation matter to choose an ontology fitting the use-case needs of the LILWENE network. Finally, the last method is used to actually invoke the services through their REST interface. For this method only requires the descriptors and the request. This method extracts from the descriptor some details to determine if the request is valid.

At its creation a server needs the URI-path of the service, a handler to process client's requests and a descriptor that describes the service functionalities and capabilities. The method `update()`, that can optionally takes a `Descriptor` as argument, is used when the service modified its descriptor. It increments the descriptor number and sends it to the registry for advertisement. When a service stops, it must send to its local registry a specific and small descriptor that informs the service died so clients do not request it anymore.

5.2 Service Description

To be usable and useful a service middleware needs to provide all the means clients need to successfully request services. To request the services, the clients need to learn their presence, capacities, functionalities and semantic. These details are contained in a document called the service description, or descriptor.

Descriptors must be as concise and complete as possible to minimize the number of messages disseminated in the network. It is important to minimize the number of messages before a successful request can be issued. Indeed, DTN networks are deployed in contexts where the connectivity is usually a challenge, messages must be assumed as easily lost due to many reasons, e.g. storage shortage, nodes failure and so on. To maximize the services' discovery, the number of exchange between clients and service providers must be kept at a minimum. The best being to include all the needed services' details in a sole document to disseminate it over the network. Hence, once clients get this document, they can directly and successfully request the service without any extra exchange.

All descriptor fields are presented in this section. More details, binary format and parameters descriptions, can be found in Appendix B. Also, due to the networking environment specific details are added to traditional descriptive information of service providers. These details are an agenda, at which a service is not sleeping because of battery considerations, a descriptor-version number, to let registries know which descriptor is newer, a deadline, at which the service is not expected to be available anymore, and a geographic restriction defining an area from which the service can be requested.

After a general overview of descriptors fields, each field is detailed in depth. This section is concluded by two examples and completed by appendix B that presents the binary format of descriptors and optional URI parameters.

5.2.1 Descriptors Overview

The different descriptors fields, detailed below, that each service must provide in its own description document are:

- Identification: to allow clients to send their requests to the right endpoint,

- Interface of the implemented HTTP methods and enabled options: to prevent clients to request services through unimplemented HTTP methods,
- Accepted inputs and outputs: to prevent clients to send (or received) data that servers (or clients) could not process,
- Ontology document: to inform clients on the semantic of the service,
- Non-functional details: version, deadline, agenda, geographic restriction and optional free field,
- Parameters descriptions (if needed): to provide to the clients all needed details so they can properly use the service.

In this list, there are two different level of detail: mandatory fields and optional parameter description. All these details are embedded in a binary document following the format described in appendix B according to the syntax:

```
<identification> 0x20 <interface> <inputs> 0xffff <outputs> 0xffff
<ontology document id> <version> <deadline> <agenda> <conditional
requesting> 0x3b3b <parameters>
```

Appendix B also contains the details of the parameters description.

5.2.2 Mandatory Fields

The choices leading to the descriptor's content of the mandatory fields as well as their content are detailed here.

5.2.2.1 Identification

The first thing a client needs to request a service is an identification of its endpoint. In traditional networks it could be the network socket or a URI the service is hosted at. However, in DTN environments, nodes are not identified by their IP addresses as these ones are expected to be highly dynamic. This means that hosts of services cannot be identified through IP addresses so services cannot neither. In fact, with BP, entrypoints of both nodes and hosted applications are URIs.

All requests, and responses, for a service will go to, and come from, its specific URI. More than the communication entrypoint it is also the identification of the service as all singleton URI are unique. The entrypoint of service would look like `dtn://door-lock1C3AD/room12A/status`, where `dtn` is the scheme, `door-lock1C3AD` is the node identification, `/room12A/status` is the path of the service.

5.2.2.2 Interface and Observe Option

Following the REST architecture style and using the common interface of PUT, GET, POST, PATCH, DELETE and the option OBSERVE, is not enough to ensure that client do not request a service through an unimplemented method. For instance, it cannot be assumed that a door lock sensor would implement PUT, POST, PATCH or DELETE but only the GET, and optionally OBSERVE option. Electric shutters may also implement PUT but not POST nor PATCH.

This is why, in order to prevent clients requesting unimplemented HTTP methods, LILWENE descriptors must include the list of implemented HTTP methods.

5.2.2.3 Input and Output

Services sometimes require inputs and sometimes produce outputs. For example, a camera service can send either pictures or videos to its clients. Clients that can only process pictures just need to request the picture type of output from this service, and more specifically the types of pictures (png, jpg, tiff, raw, ...). However, if a client is interested in the sound of the recorded video then the service cannot fulfill the client's need. In this case clients must be informed that the service cannot provide the file type needed before a client request it.

Service usually need to receive or send resources. It may be sound, picture, video, text file or binary file in many of the content format known as MIME. These resources could be configuration files sent to services or recordings requested from clients. Clients and services are assumed to only process a subset of all the content formats available. In order to avoid a *4.06 Not Acceptable* error response or reception of files that cannot be processed, the clients need to know beforehand what are the content formats a service can receive in and send out.

In LILWENE, service providers must inform the clients of both the MIME they can consume and produce by including two lists in their service descriptor. This is why two lists, one for the input MIME and one for the output MIME, are included in the description document. It is assumed that a service provider can process same content formats regardless of the HTTP method it was invoked with.

5.2.2.4 Ontology Document

As there is not one ontology to rule them all and for more flexibility, the choice of ontology specifications is left to programmers. Works on ontologies for the IoT, such as [131], or the NASA JPL ontology [132] are good candidates for this proposition. As shown by [133, 134], the semantic in the IoT context is of interest. These works also provide means to determine how to semantically describe the services in LILWENE environment.

In the IoT a lot of similar small devices are expected to be present within a same small area. It could be door lock sensors, light bulbs or heaters. All these similar devices are likely to share the same ontology document.

In LILWENE, all ontology documents must be identified by a unique ID. The form of this ID is an implementation matter and out of the scope of this work, yet in the remaining of this document the ID is assumed to be the MD5 hash of the ontology document. If a service A discovered a service B's description and they both have the same semantic and ontology document then, to keep descriptions as small as possible, service A's description document can point to B's ontology. A local DB (that could be a `hashtable<id, document>`) of ontology documents must be kept up to date. A document is erased when no service description points it out anymore.

5.2.2.5 Non-Functional Details

Added to the functional details used to request services, there are other details that need to be taken in consideration. These details are the version of the description document, service deadline, agenda of sleeping cycles, optional free fields and geographical restriction. The needs of the clients to get these details are presented in the following with their means to fulfill these needs.

Version After an update a service may not process the same Input/Output or it could stop answering to specific method, e.g. POST implementation could have been merge into PUT after a refactoring. This update could lead to unsuccessful requests. To avoid such error, the clients must be informed of these changes and especially be able to determine which description is newer. The version field aims to solve this issue. Besides, if the details included in the description are still correct after a service update then there is no need to disseminate the same description of the service just for a newer version.

The version number does not identify the version of the service itself but the service description one. The version number is actually used to request the clients to delete the previous service description. By doing this, registries can clear their DB of unnecessary descriptors and prevent the clients to unsuccessfully request older version.

The zero value is restricted: it must be used only when a service stops its activity and will die. Publishing a service description $< id : 0 >$ informs that the service died so the clients do not request it anymore and its descriptor shall be deleted.

If the service had a bug or an error in its description (e.g. wrongly advertising the method POST to be implemented) and if this service had the bug corrected then it should send a PATCH request to its previous client to inform them of the bug fix. This request stems from the auto-correcting constraint: clients that receives an error response *5.01 Not Implemented* must have updated their local descriptor so they do not send a request with the same method. This PATCH request informs the clients that the error is corrected and that the stored descriptor must be updated.

Deadline Being in a DTN WoT environment, services are hosted by things. They may be deployed for a specific duration, i.e. a few days or hours, when they are deployed for specific occasions or events such as festivals, town markets or concerts.

In other words, there is no service that are expected to last indefinitely. To avoid the risk of clients requesting a non-existing service the description must include a deadline at which the service should not be requested anymore. This deadline is an epoch date. When this deadline is reached the service description must be removed from the registry.

Agenda The things could have a sleeping policy aiming for a longer battery duration. It could be to turn off its communication means at regular hours, such as every nights. The clients should be informed of such policy so they do not send any request that may fail because the service host is sleeping. That is why, added to its deadline, an agenda is added to inform the clients at which date the service is unreachable. The agenda takes the form of a `cron` entry with the last argument being a duration of the sleep in minutes instead of the command to execute.

Just like `cron`, the precision of the agenda is kept at the minute. Indeed, as the *things* may follow sleeping cycle and by lacking of stable connection their clock are likely to shift in the long run. This is why the agenda precision should be considered as a coarse-grained one.

Optional Free Field In some case, service providers could be required to evaluate their QoS, or provide location of fixed devices. In LILWENE descriptors, the field named *optional free frield* is free to be used and encoded as needed by the users as long as this field is terminated by a specific marker.

Geographic Restriction Services may provides their capabilities on screen or speakers. Their clients should then be restricted to a specific area. In order to inform LILWENE clients of the area constraints of such services a field is added into the descriptors. In order words, LILWENE services can set restrictions to be only requested by clients present within a defined area. Some works already use location of the nodes for specific decision in DMANET. For instance in [135] the node's location is used for routing decision.

Because the BP, and more specifically the implementation IBR-DTN, respects the layered model and because the lower layers involved in the dissemination should not have to treat data such as the location of the node itself, then the bundles cannot be restrained to a specified area. Hence, the description will be disseminated further and discovered by far away clients. Nonetheless, this is not a negative effect. Indeed, the clients discovering this service are aware that the service exists and know how, and from where, they can request it. Some nodes may even move themselves to this area to request the service. It is the client duty to check if it is allowed to emit a request.

5.2.3 Description Examples

In the following two examples illustrate the possibilities of LILWENE descriptor. These two example are presented in binary format in the appendix B.

Simple Temperature sensor

This service offers to respond the current temperature of the entity "fridge". Its identification is `dtn://fridge-sensor/temperature`. It implements the GET request and the OBSERVE option to periodically push the current temperature. The temperature is sent in the payload of the responses as text/plain (3). The ontology document is assumed to be in another bundle-payload block and the document-id here is the MD5 hash: `95 28 59 ... 77 7a fa`. The version of the service is `0x13`. Its deadline is calculated at boot time for a duration of ten years and the sensor never sleeps (as it is actually plugged). The parameter "variation" is used to set the temperature window within which the client do not want OBSERVE notifications, e.g. if set to 3, the client will not receive a notification unless the fridge temperature decreases or increases of 3 degrees. It must be restricted to positive value under ten. This service description would be:

See B.3 for the full binary content of this description.

Configurable Light Bulb

In this other example, a light bulb service manages a colored light bulb. Its identification is `dtn://light0a/access`. The light can be turned off and on through PUT request using the parameter "turn". It responds its state and color through a GET request as text/plain (3). The ontology document is not provided, but the MD5 hash is present: `a6 28 59 ... 77 7a fb`. The version is `0x10`. The deadline of the service is 4 years. It sleeps during day time, from 9:00 a.m. to 6:00 p.m. The color can optionally be set through PUT request using either a hexadecimal value with the parameter "hex" or send three parameters "r", "g" and "b". The parameter "hex" is in the parameter group number 1 while "r", "g" and "b" are in the parameter group 2 because, as they are redundant, they should not be used together. "hex" is a string constrained by the regular expression `[0-9a-fA-F]+`[1]. A real implementation of this service may use an `int` instead of a `string` for simplicity. The parameter "turn" is both in parameter groups 1 and 2. Its description would then be:

See B.3 for the full binary content of this description.

[1]Regular expression is one of the different contents a `string` can represents. These different contents are presented in B.2.

5.3 Service Advertisement and Discovery

As presented above the advertisement and discovery mechanisms are done by registries through topics with a pub/sub interface. The details and specificities of these mechanisms are exposed here.

5.3.1 Publish-Subscribe Interface

It actually makes sense in the WoT to offer pub/sub mechanism as lot of *things* are expected to share similar interests and produce or consume data from interests-based-topics fed by the *things* themselves. In [136], a middleware architecture for a mobile peer-to-peer content distribution is presented. This middleware disseminates data in a DTN network in a content-centric manner, opposed to the destination-based the BP follows. To do so the content is structured into logical topics that are accessed through a publish/subscribe interface. Among the examples presented in the paper [136] there is one about *relaying sensor data*, that can be likened to the WoT context. Another work on pub/sub mechanism for service support in the context of mobile devices with wireless connexion is presented by Caporuscio et al. in [137]. Relying on a decentralized pub/sub in a DTN network for advertisement/discovery of services offers opportunities of collaboration among nodes to help one another to discover new services.

In our case, the producers, and consumers, are respectively the registries of service-node and the registries of client-node (they may be both). As for the data relayed, they are not sensor's data but service descriptors. If the BP provides little means to natively support a publish/subscribe interface, some usage constraints can be set in order to support it.

The BP supports two different types of endpoints: singleton endpoint and non-singleton endpoint, where endpoints are identified by URIs. The non-singleton endpoints, called group URI, are URIs locally created by an application when it registers its endpoint as this URI. An example for this kind of URI could be `dtn://group/multicast`, if a node sends a message to this URI then all applications that locally registered as `dtn://group/multicast` can receive the same bundle. In like manners, if all nodes host an application that registered to the URI `dtn://group/bcast` then a bundle sent to this URI can be received by all nodes of the network.

Having similar applications hosted on all nodes of the network enables a pub/sub mechanism. The subscriptions are effective by the registration of these applications to the BP daemon. In our case, the identification of a topic is its URI itself. The publication of a message is done by setting the destination to the URI (topic) the node wants to publish in.

This pub/sub mechanism is used for the advertisement and discovery of services. Indeed, the service descriptions are published to a topic to which registries can subscribe to in order to discover the service descriptions.

An extension of CoAP exist, as Internet Draft [138], defining a pub/sub interface. This interface follows a centralized architecture by relying on a broker that acts as a CoAP server but also proposes a brokerless architecture within which each node hosts its own broker. A broker interface must implements several operations: discovery, create, publish, subscribe, unsubscribe, read, and remove. As an analogy, the pub/sub interface proposed in this thesis covers five operations that are:

create Topic creation, by publish its name into the main topic,

publish Publishing a service descriptor,

unpublish Publishing a service descriptor with a version number equals to zero, that has the effect of deleting this descriptor on registries (it has no equivalent in the CoAP pub/sub interface),

subscribe Subscribing to a topic, by registering a new BP EID to the BP daemon,

unsubscribe Unsubscribing to a topic, by unregistering an existing BP EID to the BP daemon,

5.3.2 Topics

Several topics coexist in LILWENE network. Topics can be dynamically created. Registries shall know all topics, so to discover new topic their name are advertised into a main topic: `dtn://group/.well-known/discovery`. Two kind of topics exist: the main one and others. Other topics are also called subtopics. Their specificities are presented in the following.

5.3.2.1 Main Topic

As explained above, the advertisement/discovery is done with a content-centric orientation through a publish/subscribe interface inherited from the non-singleton EID (or group URI) of BP. The advertisement is done by publishing a service descriptor into a topic. Topics are non-singleton BP EID and follow the construction of URIs, that actually identify them. There must be at least one topic in a LILWENE network that is considered as the main topic. This topic, assuming that the network scheme is `dtn`, is:

```
dtn://group/.well-known/discovery
```

If there is only one topic then all registries must subscribe to it and descriptors must be published into it. In LILWENE all nodes must have one registry that registers to its BP daemon as this URI and processes the received bundles to update the local DB of service descriptors.

5.3.2.2 Subtopics

When there are a lot of service providers, publishing all service descriptors into a single topic may flood the registries as they would process and store every single service descriptor whereas clients may only be interested in a small subset of these providers. For this reason, and to overcome this challenge, added to the main topic, `dtn://group/.well-known/discovery`, LILWENE nodes can use subtopics. Using subtopics allows to scatter the topics so the clients can only subscribe to the subtopics they are interested in. Henceforth, by selecting only a subset of subtopics they receive a subset of service descriptors, the ones they are interested in. As an illustration, a list of subtopics could be:

```
dtn://group/.well-known/discovery/sensor/temperature
dtn://group/.well-known/discovery/actuator/temperature
dtn://group/.well-known/discovery/sensor/light
dtn://group/.well-known/discovery/actuator/light
dtn://group/.well-known/discovery/security/sensor/door-lock
dtn://group/.well-known/discovery/security/actuator/door-lock
```

This scattering lessens the registries DB size as some clients may exclusively be interested in the temperature environment or the security. If a client requires both temperature- and light-related descriptors then its registry can subscribe to both subtopics. When subtopics are dynamically created, the main topic must not be used to publish service descriptor but rather to advertise subtopics.

It must be noticed that it is the duty of developers to not pollute subtopics and not create a large amount of subtopics tightly related.

5.3.3 Advertisement

Service descriptions are published with the HTTP method PUT and not POST. Indeed, PUT has been chosen as it is an idempotent method, unlike POST, and that a registry receiving multiple times the same service descriptor must stay in the same state.

Since the service providers only send their own descriptor to their local registry then it is the duty of the registries to determine if it is better to publish it or wait for a discovery request. To publish a descriptor into a topic, registries only have to set the descriptor bundles' destination to this topic's URI and send it. If there is only one topic, the main one, then the destination of the bundle must set to it, otherwise descriptors must be published in the best matching subtopic. A node should avoid to create new subtopic and use an already existing subtopic. In the case no matching subtopic exists, a LILWENE registry can advertise a service descriptor into a new subtopic. The decision to determine whether or not a descriptor should be advertised is a matter of implementation, such decision could take into consideration node storage estimated through the average number of bundles being disseminated, number of descriptor already existing, and so forth. A subtopic is considered created when its URI is advertised into the main topic. Advertising subtopics ensures that other registries can discover these subtopics and thus subscribe to these.

To summarize, in LILWENE, a descriptor is advertised by publishing a PUT request, embedding the document, into a (sub)topic. This request will not be acknowledged nor answered and must be published in at most one (sub)topic. If a subtopic is dynamically created then its name must be advertised into the main topic.

5.3.4 Discovery

Clients need to discover services before invoking, or requesting, them. Local registries store service descriptors. To access them a client application only needs to describe its needs and send a GET request to its local registry. The registry will then answer with known descriptors or try to discover new ones if none matched the client's needs.

There are two kinds of discovery: proactive and reactive. The proactive discovery offers means to the registries to discover descriptors by only subscribing to (sub)topics whereas with the reactive discovery a registry is required to sent a request beforehand.

5.3.4.1 Proactive Discovery

In LILWENE, the advertisement is done by publishing descriptors into one (sub)topic, that could be dynamically created. Each local registry have a unique EID that is `dtn://node-id/.well-known/registry`, where `node-id` is different for each node. This configuration is not enough to let registries receive the descriptors. Indeed, the descriptors are embedded in bundles that are sent to non-singleton BP EID. A registry, to receive these descriptors, must then register to its BP daemon as these non-singleton BP EIDs. This registration represents the action of topic subscription. Indeed, after registering as a topic (i.e., a non-singleton BP EID), the BP daemon will then forward every bundles that have this EID set as destination.

A LILWENE registry is then composed of at least two EIDs: its unique one and the main topic. In network with subtopics, registry can subscribe to one or more subtopic following the same procedure of registering as an additional BP endpoint. Handler for each descriptor receivers is the same as the processing of each descriptor is the same regardless of the subtopic it was received.

On a technical point of view, a registry can instantly discover new descriptors by subscribing to a new topic. Indeed, thanks to the store-carry-and-forward principle, a BP daemon stores descriptors in bundles. These bundles could be sent to a topic that the local registry of this node did not subscribe to. The local registry could receive a client request that would trigger a new subscription. This new subscription could be a topic that the BP daemon carries bundles for, triggering the daemon to sends all stored bundles to the registry new EID. To summarize, a node may store new descriptors in bundles that can be processed immediately as soon as the registry subscribes to a new topic since the BP daemon will then deliver bundles embedding descriptors to the registry through its subscription.

5.3.4.2 Reactive Discovery

In LILWENE the descriptors are not automatically published. This could harm a network wherein, for instance, a lot of various services are present but a few clients are. In such network clients' storage would be saturated with published descriptors while clients may be interested in just a small portion of these services. To solve this issue the registries can locally hold the descriptors by waiting for a request. Upon reception of a request from a registry for new descriptors, then registries that received this request can answer by sending the stored descriptors matching the needs expressed in the request.

To put it differently, a LILWENE registry can embed its client's needs into a GET request and publish it into the main topic so other registries will answer with their needs-matching descriptors. This second kind of discovery is called the reactive discovery since, unlike the proactive one, a request is required before any descriptor can be discovered.

5.3.5 Registry Notifications

Clients may keep in memory the service descriptions they received earlier. These descriptions may become out-dated and a client may request a service using an HTTP method that is not supported anymore while the local registry has an up to date service descriptor. These situations must be avoided. These situations waste network resources by disseminating requests that would result as error-responses disseminated while the information that may avoid such request was stored on the client node in the first place.

A solution to this issue would be to use the local registry as a proxy for the request. The clients would send their requests locally to the registry and then the registry would either sends an error to the client if the request would result as an error or emit the request itself. However this complicates the architecture because the clients would require the registry to preprocess all the requests, look-up the service descriptor into the DB, emit request on behalf of the client or, in some cases, answer back on behalf of the services. In addition, the source of the bundles would have to be forged so the responses are directed towards the clients themselves and not the registry, but any system relying on message forgery poses security concerns. Furthermore clients and registry would be tightly bound together which is something that must be avoided in software architecture, especially when these softwares are services.

Another solution would be to request the client a permission to the registry. The permission would be requested by sending the actual request to the registry that would preprocess it to authorize or not the client to send it. But then again, the registry and clients would be tightly bound together, the registry would have more DB look-ups to perform and this would solution would induce additional delay.

In LILWENE, another solution is chosen and can be described as push notifications, likened to OBSERVE additional responses, from the registry to the clients when a newer service descriptor is received. When a registry receives an updated descriptor then this new version is forwarded by the registry to the clients that requested this descriptor before. This is why in LILWENE the registry must track its clients and the descriptions it sends back to them. If a newer version of a service is received, the registry must notify this update to all the local clients that received this descriptor. This mechanism is a direct consequence of the fourth design principle named *update* and aims to improve LILWENE reactivity and efficiency. These updates, or notifications, must be sent to the client's EID used for internal management.

5.4 Service Invocation

As stated in LILWENE overview, invocation mechanisms lie on BoaP. Nevertheless, some specific behaviors must be respected by both clients and services to enhance the usability of the system as a whole.

5.4.1 Client Side

Request Testing Before Emission

Before a client sent a request, it must perform several tests to decrease its risk to receive an error-response. These tests are executed to ensure that the service is still alive (i.e., service's deadline not reached), not sleeping (i.e., according to its agenda), that the HTTP method is implemented by the service, and, if possible, if any, that the client is in the restricted geographic area.

If all these tests are passed, then it can be assumed that there is only few risk that the request will not be successfully answered. These risks are failures of network or intermediary nodes, or that an error was present in the service descriptor.

Error-Reponse Processing

Despite the wariness before sending a request, error-responses can still be sent by services. These messages, responses carrying an error, must be analyzed to further decrease unsuccessful exchange risk. Thus, if a client receives an error-response then a verification must be done in order to determine if the error could have been avoided with a modification on this service's descriptor. Such avoidable error could be a response with a code *5.01 Not Implemented*, this error can be avoided by tagging the HTTP method of the request as unimplemented. This modification is one of the four principle design: correction. Corrections aim to prevent local clients to send similar request that would automatically result as an error.

A correction is sent by a client to its local registry with a PATCH request. The request must contain the version number of the service description, so the local registry can ignore the patch if a newer version of the description was received, and the modification. Once the registry received a correction from a client then the registry must send a notification to all local clients that had received this service descriptor before. This is likened to the update principle design.

5.4.2 Service Provider Side

There are use-cases where a service may receive an ununderstandable request or a request that could automatically result as an error-response. Avoiding these requests enhances the system as a whole and prevents network resources to be wasted by disseminating both requests and error-responses. This is why, in such cases, services must check if these requests could be avoided either by respecting its current descriptor, either by updating its descriptor.

If the current descriptor would not have permit such request, then it can be assumed that the descriptor on the client's node is out-dated. To remedy this matter, the current descriptor can be directly sent to the client's registry that will be treated as an update, since the current stored descriptor is out-dated on this client node. On the contrary, if the current descriptor permits such request and that a descriptor update could solve this matter then the service must update its descriptor and disseminate it. It can also send a copy directly to the client's registry.

5.4.3 Group Invocation

With LILWENE, and BP, it is technically possible to invoke several service providers at once by using a non-singleton EID. If all similar service providers register to their BP daemon with the same non-singleton EID, then this EID can be used to invoke all of them at once. Doing so saves up network resources as each intermediary node would carry a single request instead of carrying one for each provider.

The optional free field of service descriptor can be used to inform on the non-singleton EID, assuming that the service providers have a means to determine what this common EID should be. Proposing such means is out of the scope of this thesis. Also, when a group is invoked, if a request results as an error-response, the service providers might not respond.

Different communication modes can be used when a group of service providers is invoked. These modes are:

Anycast When invoking a group, the client may be interested by a single response. For instance a client may want to check whether it is raining or not by requesting the group of humidity sensors from outside. A single response would be enough to deduce the weather yet all the sensors may answer. Requesting a single one from the group would lower the chance to get a response and slow down the process as it may not be the fastest exchange possible. Nevertheless if all sensors send back a response then the network resources may be wasted. The client could simply drop all responses coming after the first one, or a network healing mechanism could be deployed. Network healing mechanisms are exposed further in the conclusion chapter.

This communication mode is *anycast* as the client is interested in at least one response. It is the responsibility of the client to choose how to process responses coming after the first one, that could simply be dropped.

Multicast Opposed to the case above, there are situations when a client may want to receive as many responses as possible. For instance, a client could have the objective to log every room temperatures of a house. In this case, the client would then send a request to a group and as many responses as possible would be needed.

This communication mode is *multicast* as the client needs as many responses as possible. No healing mechanism is expected to be used as the client is interested in all the responses.

5.4.4 Request Options

Three options are proposed for LILWENE invocation requests:

Caching Option LILWENE requests can be marked as *public* by using the Bundle Metadata Extension Block (MEB [7]) so the intermediary nodes can read the request and answer it if they carry a cached and still fresh response for the public request. Responses of these *public* requests shall be marked as *public* too.

Time Option The second option takes the form of a date. This date is the expiration date at which the requesting client does not need the response anymore. Thus, if the requested service provider cannot compute its response fast enough then it shall not emit the response.

A lifetime bundle embedding a request with this option shall not share the same value, i.e. the bundle lifetime and the option date are not expected to coincide. Indeed, this option value can be a very far date and, in the case the bundle expiration coincides with it, the bundle will occupy node storage for a long, unnecessary duration. Nevertheless, for the response, the bundle lifetime must not exceeds this option value. Indeed, the client does not need the data after the date, so if the bundles carrying this data are deleted then it will not prevent the client to reach its goal.

Spatial Option The last option exploits node's location. When the nodes have access to their location (e.g., thanks to a system like GPS), this detail can be used by client to impose a spatial condition on the service providers. This option can be used to receive located responses.

This option defines an area within which service providers are allowed to answer. Outside of this area, provider's responses do not interest the client.

5.5 API and Implementation Elements

To support the architecture presented, this section shows different code examples of a possible LILWENE implementation. Implementation elements of the middleware relying on the BoaP prototype as well as the IBR-DTN Java framework are also presented in order to provide a technical understanding of this middleware. In the following, the implementation-elements examples are framed to visually differentiate them from the API-code examples.

```java
public interface LilweneClient {
    public void create(String eid, ClientHandler handler);
    public void discoverServices(Needs needs);
    public void requestService(Descriptor descriptor, Request request);
}

public interface LilweneService {
    public void create(String eid, ServiceHandler handler, Descriptor
     ↪   descriptor);
    public void update(Descriptor descriptor);
    public void update();
    public void stop();
}

// The registry is a required tool and does offer an API since its processing
 ↪   shall be transparent.
```

Listing 13: LILWENE API

As a reminder, List. 13 shows the LILWENE API that offers seven methods, three for clients and four for services. These methods are shown in API examples below with their implementation elements. Added to these API methods, some registry tasks are illustrated too through code examples.

5.5.1 Client Application API

Added to the three client API methods, code-elements are shown for accessing a service descriptor, processing a registry notification and processing an error-response with a correction sent to the registry.

5.5.1.1 Client Creation

Creation of a client application does not require many information, just the path-part of a URI and a message handler. The message handler can be likened to the `BundleHandler` (see 3.3.1) or `BoaPMessageHandler` (see 3.3.2.1), and is user defined. The code to create a client is then similar to the code presented in List. 14. In this example the variable `eid` represents the path-part of the URI and `MyHandler` represents the user-defined handler to process responses.

```java
final String eid = "client-endpoint";
LilweneClient lClient = new LilweneClient();
lClient.create(eid, new MyHandler());
```

Listing 14: LILWENE: Client Registration

Underlying mechanisms of this API methods are two BP endpoint-registrations. The first one, the main one, will be used to set the source of the service requests and will be used by the service provider to send back the responses to this endpoint. The second one, exclusively used locally with the registry for internal management, is used to both request, correct and receive service descriptors and registry updates.

```java
public void create(String eid, LilweneMessageHandler handler) {
    this.clientApp = new BoapApplication(eid);
    this.clientApp.setHandler(handler);
    this.descriptorReceiver = new BoapApplication(eid + "/registry");
    this.descriptorReceiver.setHandler(LILWENE.DescriptionHandler());
}
```

Listing 15: Client Registration

List. 15 shows the `create()` method implementation. A LILWENE client is an application able to request its local registry to access service descriptors and able to request distant services too. To create a `LilweneClient` two parameters are needed: a name, path-part of the client's EID, that is represented by the `eid` variable and a user-defined handler that is represented by the `handler` variable. It should be noticed that the constructor create two different `BoapApplication` to which correspond the main endpoint, identified by the `clientApp` variable, and the second one, `descriptorReceiver`, that is used to communicate with the local registry.

5.5.1.2 Descriptors Requesting

List. 16 presents an example of a client requesting its local registry. The process to express the client's needs, contained in the `needs` variable, is considered to be out of the scope of this work. Variable `lClient` represents a LILWENE client.

```
Needs needs = new Needs();
// ... Operations to express client's needs ...
lClient.discoverServices(needs);
// The response will be processed by the handler of the client endpoint that
↪  communicates with the registry
```

Listing 16: LILWENE: Local Registry Requesting

In List. 16, a LILWENE client requests all descriptors matching its needs that the local registry discovered. If the local registry discovered none, then the local registry can either create new topic-subscribers or request other registries; otherwise all matching descriptors are returned. Each future update on the service descriptors that were sent by the local registry will be notified to the client-endpoint that finishes by /`registry`.

This API example may rely on implementation element exposed in List. 17.

```
public void requestRegistry(Needs needs) {
    String localRegistry = "dtn://" + LOCAL_NODE_ID + "/.well-known/registry";
    Request registryRequest = RequestGET(localRegistry);
    registryRequest.setPayload(needs.serialize());
    this.descriptorReceiver.send(registryRequest);
}
```

Listing 17: Local Registry Requesting

List. 17 is quite simple to detail. It creates a BoaP request sent by the endpoint `descriptorReceiver`. The registry processing of such request can be explained as: the registry lookups in its DB, and sends back the matching service descriptors, if any. The client processing to treat the registry response is presented further below.

5.5.1.3 Selecting and Requesting a Service

LILWENE clients may live for a long time and send several requests during their lifetime. As an implementation choice, to send its requests, a client relies on a `RequestSender` objects. The objective of a `RequestSender` is to fulfill client's needs, represented in `needs` variable, by sending a request to a service matching client's needs. It is assumed that a `RequestSender` implements `Observer` interface to observe the `clientDB` variable to which it has a reading access. A `RequestSender` can use both client EID to send requests. The method `select(List<DescriptorPointer> descriptorPointers, Needs needs)` selects out and returns the `DescriptorPointer` matching at best the `needs`.

```
1  public RequestSender(LilweneClient owner, Needs needs) {
2      this.owner = owner;
3      this.needs = needs;
4  
5      Descriptor service = select(this.owner.getClientDB(), needs);
6      if (service == null) {
7          this.owner.getClientDB().addObserver(this);
8          requestRegistry(needs);
9      } else {
10         this.sendRequest(service);
11     }
12 }
13 
14 public void update(Observable o, Object arg) {
15 // Select the best match according to the needs:
16     Descriptor descriptor = select(descriptorPointers, needs);
17 // If a descriptor matches the needs, then request the service:
18     if (descriptor != null) {
19         this.sendRequest(descriptor);
20         this.owner.getClientDB().deleteObserver(this);
21     }
22 // Otherwise, wait for next update
23 }
24 
25 private void sendRequest(Descriptor descriptor) {
26     this.owner.requestService(descriptor, needs.getRequest());
27 }
```

Listing 18: Selecting a Service to Request

List. 18 shows an example of a `RequestSender` with its different methods. Upon creation, by a LILWENE client, it is added by its owner as an observer to the owner `clientDB`. At its creation, the `RequestSender` verifies if there is an existing descriptor in the `clientDB` matching its needs. If there is not any, then it request the registry with the method `requestRegistry(needs)` (l.7) and wait for an update. Upon `clientDB` update, all `RequestSender`s are notified and their `update()` methods are executed. During an update, a `RequestSender` verify if there is an existing descriptor matching its needs, if there is not any, it wait for the next update ; if there is one then `RequestSender` requests it and then stops observing.

5.5.1.4 Service Requesting

List. 19 shows an example of the code to request a service.

```
// Request creation
    byte[] requestPayload = payload.serialize();
    Request serviceRequest = Request(METHOD.GET, payload);

    lClient.requestService(descriptor, serviceRequest);
```

Listing 19: LILWENE:Requesting a Service

To request a service two conditions must be checked: *is the service still alive?* and *is the service awake?*. This code is simple and very similar to the request sent to the local registry during the discovery phase with a few tests. List. 20 shows what the implementation could be.

```
public void requestService(Descriptor descriptor, Request request) {
// First of all, check if the service is still alive
    if (descriptor.getDeadline() > now()) return;
// Then check if the service is awake:
    if(! document.getAgenda().isAwake()) {
        this.rescheduleRequest(descriptor, request)
    } else {
        this.boapApp.send(descriptor.getDestination(), request);
    }
}
```

Listing 20: Requesting a Service

5.5.1.5 Processing Registry Response

In this LILWENE implementation, each client manages a DB, identified in the code by the `clientDB` variable. This DB can be likened to a hash-table for which the keys are services' EID and values are pointers to services' descriptors contained in the registry DB. Clients are granted read-only access to the registry DB. Thus, in this implementation, When a registry sends descriptors, it actually sends pointers to its own DB.

In List. 21, shows the processing done to memorize pointers to descriptors sent by the registry. The client DB is assumed to be observed, thus if any process is waiting for a descriptor to send a request, it can be woke up as soon as the `clientDB` is updated thanks to the registry response.

```java
private void processRegistryResponse(List<DescriptorPointer> descriptorPointers)
 ↪  {
    // Add all descriptor pointers into the client DB:
    clientDB.add(descriptorPointers);

    // Notify observers:
    clientDB.notifyObservers();
}
```

Listing 21: Processing Registry Response

5.5.1.6 Processing a Registry Notification

In these implementation examples, descriptors are assumed to be stored in a DB shared between client applications (in read-only) and the registry. Client applications have a direct access in order to minimize the number of requests with the local registry: clients have access to previously requested descriptors so they do not need to request it again. In List. 22 we review what happens when the registry received a newer service descriptor. In this case, as stated in 5.3.5: upon service descriptor update, the local registry must notify the clients that requested this descriptor. This notification contains the service descriptor-document (or its pointer) so the client does not need to request it, thus avoiding another local exchange.

The variable `update` carries the pointer of the updated service descriptor. It is assumed that the client DB is a hashtable where the keys are the service endpoints and the values are the descriptor-pointers.

```java
public void updateProcessing(DescriptorPointer update) {
    // If the service descriptor is not memorized, exit the method although,
     ↪  optionally, the descriptor can also be memorized if it is not a PATCH
     ↪  request (fixing a faulty descriptor) but if the descriptor is complete
    if (!clientDB.contains(update.getEndpoint())) {
        return;
    }
    // Otherwise, replace the service descriptor:
    clientDB.replace(update.getEndpoint(), update);
    // Notify observers:
    clientDB.notifyObservers();
}
```

Listing 22: Registry Update Processing

5.5.1.7 Processing an Error Response

As explained in 5.4.1, clients must correct service descriptors if an error-response is received and that this error could be avoided with a corrected service descriptor. In List. 23 the method `corrigible()` return the object `Descriptor` containing the corrected service descriptor if the response error code is among the corrigible ones, otherwise the returned value is `null`. This code is executed for every single response with an error code.

```
    Descriptor correctedDocument = null;
    correctedDocument = serviceDocument.corrigible(response);
// If the error do not permit any correction, exit.
    if (correctedDocument == null) return;
// Otherwise request the local registry to update the document
    localRegistry.requestCorrection(correctedDocument);
```

Listing 23: Processing Error Response

5.5.2 Service API

The three methods API of service applications, being `create(..)`, `update()` and `stop()` are presented below.

5.5.2.1 Service Creation and Descriptor Update

The service creation process is very similar to the client creation one since both require an EID and a handler. List. 24 illustrates this creation as well as a descriptor update.

```
final String eid = "service-endpoint";
Descriptor descriptor = new Descriptor();
// ... Operations to set descriptor fields ...
LilweneService lService = new LilweneService();
lService.create(eid, new MyServiceHandler(), descriptor);
this.descriptorVersion = 0; // Incremented in update()

// Descriptor update
lService.update(descriptor);
```

Listing 24: LILWENE: Service Creation

In List. 26, two different `update()` methods can be seen. One uses an optional argument: the descriptor. The descriptor-fields setters can be used in combination with the method `update()` instead of using `update(Descriptor descriptor)`. The process of update is as simple as sending a PUT request to the local registry. The registry determines if the descriptor should be directly published into a topic, proactive discovery, or if it should wait for a request, reactive discovery.

```java
public void create(String eid, LilweneMessageHandler handler, Descriptor
 ↪  descriptor) {
    this.boapApp = new BoapApplication(eid);
    this.boapApp.setHandler(handler);

    // Descriptor advertisement (to the registry)
    this.update(descriptor);
}

public void update(Descriptor descriptor) {
    this.descriptor = descriptor;
    this.update();
}

// Increment the descriptor version and send the descriptor to the local
 ↪  registry:
public void update() {
    this.descriptorVersion++;
    this.descriptor.setVersion(this.descriptorVersion);
    byte[] payload = this.descriptor.serialize();
    Request descriptorRq = Request(LOCAL_REGISTRY_EID, METHOD.PUT, payload);
    this.boapApp.send(descriptorRq);
}
```

Listing 25: Service Creation

5.5.2.2 Dying Service

When a service stops, its descriptor should be advertised. This descriptor must have only two fields: the service identification (EID) and the version number set to zero. A registry receiving such descriptor can then remove the service descriptor from its DB. List. 26 shows the implementation of the method `stop()` that should be used to advertise a dying service.

```java
public void stop() {
    // Clear all the fields except the identification
    this.descriptor.clear();
    // Set the version to zero.
    this.descriptorVersion = -1; // Incremented to zero in update()
    this.update();
}
```

Listing 26: Service Creation

5.5.3 Registry Processing

The registry must be as transparent as possible for developers using LILWENE. This is why there is not any API call available to the developers. Nevertheless, some implementation elements are exposed below for a better understanding of the registry entity.

5.5.3.1 Registry Initiation

As explained in 5.3.3, each local registry may subscribe to one (or more, if available) topics. In the following code, List. 27, the local registry subscribes to three different topics that are:

- `dtn://group/.well-known/discovery/light`,
- `dtn://group/.well-known/discovery/shutter`, and
- `dtn://group/.well-known/discovery`, the main topic.

All messages received through these topics are processed by the same handler as explained before.

```java
// Local registry singleton-endpoint creation
    String eid = ".well-known/registry";
    BpApplication registry = new BpApplication(eid, new SubscriptionHandler());
// Subscribing to the topics
    String[] topics = {"dtn://group/.well-known/discovery",
                       "dtn://group/.well-known/discovery/shutter",
                       "dtn://group/.well-known/discovery/light"};

    for (int i = 0; i < topics.length; i++) {
        DescriptorReceiver.subscribe(topics[i]);
    }
```

Listing 27: Registry Creation

5.5.3.2 Descriptor Reception

A registry, just like client applications, have several EIDs: one singleton EID for its identification and communication with local client and one, or more, non-singleton EID for each topic subscription. These non-singleton EIDs receive descriptor and are represented by variables `DescriptorReceiver` in the following code examples. At `DescriptorReceiver` creation a handler is set. This handler is common to all `DescriptorReceiver`: each time a descriptor is received the handler checks if a descriptor for the same service provider is already stored, then the handler checks the version number and if it is zero then the descriptor is removed from the DB, else if it is lower or equal then the registry ignores the descriptor, else if the version number is greater then it replaces the descriptor. If no descriptor was stored for this service, the registry simply adds it in the DB. Then, the registry must check whether clients' needs match this new descriptors and notify them of this new descriptor. Furthermore, the registry must notify all clients to which the registry sent an older version of this descriptor previously.

```
for(Iterator i = documents.iterator(); i.hasNext(); ) {
    ServiceDescription document = i.next();
// If the document is already stored ..
    if(!documentsDb.contains(document)) {
        String id = document.getEndpoint;
        int storedDocumentVersion = documentsDb.get(id).getVersion();
// .. and if the version is zero:
        if(document.getVersion() == 0) {
            // Delete the document, as the service is stopped
            documentsDb.remove(document);
        }
// .. otherwise if the version is newer:
        else if(storedDocumentVersion < document.getVersion()) {
            // Notify the client of the descriptor update
            List<Client> clients = documentsDb.getClientsOf(document);
            for(Client c : clients) {
                c.notify(document);
            }
            // Replace the document
            documentsDb.replace(document);
        }
// Otherwise store it.
    } else {
        documentsDb.store(document);
    }
}
```

Listing 28: Descriptor Reception

Code shown in List. 28 presents the processing that a registry is expected to perform when it receives a message containing one or more descriptors. The variable `documents` is assumed to contain all the parsed service descriptors sent in the message. The object `descriptorsDb` is the entity that stores all the descriptors. This code may be contained within a method `descriptorsDb.update(documents)` called by the common handler of `DescriptorReceiver`, `DescriptorReceiverHandler`, upon reception of a new descriptors.

5.6 Conclusion

LILWENE is a Service-Oriented Approach middleware provided by an abstraction layer on top of BoaP. It enables a services-style programming instead of a resource-style one. LILWENE features presented here respect the four design principles identified and exposed at the beginning of this chapter (full and concise description, local registry, correction and update) while following DTN and REST constraints and taking into account DTN and WoT challenges. While this middleware architecture have not been implemented, it has been thought through from abstract concepts down to an API with implementation elements. The natural next step for LILWENE would be to actually develop it by relying on the architecture, design principles, guidelines and keys provided in this document.

The proposition of this service platform has been designed for ICN. With that in mind, it may be worth questioning its applicability in the traditional networks using the classic CoAP over UDP. Does porting this proposition would have drawbacks? What would they be? How to overcome them?

With traditional networking in mind it is assumed that CoAP and UDP are deployable, thus the network must be routed and end-to-end path are assumed to exist with sparse disconnections. Inasmuch as the store-carry-and-forward principle does not apply anymore, then the messages are not stored on nodes that are not the destination.

As a result the nodes will not be able to receive the requests that are not sent towards them directly. Under those circumstances the intermediate nodes cannot send the cached responses to other nodes, as there is not any intermediate anymore, apart from proxies themselves but then it falls in the regular CoAP caching mechanism. This implies that the cache feature will be useful only locally, as described in [11] or through proxies.

The discovery is also impacted. The topics must take a different form, fortunately CoAP supports multicast request, see [11, section 8], therefore the discovery can be made through this multicast requesting by embedding the topic name, or URI, when subtopics are used. The pub/sub CoAP interface [138] could also be used as a substitute. All clients would then receive these but they may drop the description of subtopics they did not subscribe to.

Apart for these two modifications, cache feature and discovery mechanism, all the aforementioned propositions subsist and remain valid. That is to say, the described platform is also a good candidate for the traditional networking environment.

Other solutions that tackles the identified challenges usually do not rely on standard and known protocols but rather provide a full stack middleware solution. My solution provides a high-level programming interface while respecting standards and well-known architectures. Nevertheless, there are still limitations and open challenges such as service composition and optimizations, that would be enabled by stepping away from the layered design although this one greatly helps in traditional networks.

6 Conclusion

Contents

6.1 Summary

The emerging paradigm of Internet of Things (IoT) is usually considered in the literature as an extension of Internet. Things are often equipped with short-range radio interfaces, undergo energy-saving constraints that shutdown their radio interfaces, can be mobile (e.g., when embedded on robot or carried by humans) or can be deployed in area lacking of network infrastructure. For these reasons, they are likely to suffer from frequent and long delays and disconnections. This context, related to IoT can be described as the DT-IoT: Delay-Tolerant Internet of Things to which the Delay-Tolerant Networking (DTN) architecture as well as Bundle Protocol (BP) provide means to overcome communication challenges.

This thesis studied programming-support systems in this DT-IoT. According to [59], one of the strength of Web principles is its low hardware requirements. This requirement characteristic befits well in DT-IoT context. For this reason, programming-supports of this thesis were investigated to be Web-oriented and RESTful. Furthermore, contributions of this thesis were willingly researched to be as close as possible to existing standards since they are well known, reliable, tested and approved. Web-oriented solutions for the IoT are part of the so-called Web of Things (WoT). The programming-supports of this thesis were then focused on a Delay-Tolerant Web of Things (DT-WoT). Two different objectives were aimed during this thesis: proposing one resource-oriented and one service-oriented programming-support for a DT-WoT.

Naturally, two solutions were proposed. The first one, resource-oriented, is inspired by Constrained Application Protocol (CoAP) and was extensively evaluated. The second solution is a service-oriented one relying on a publish/subscribe mechanism to fulfill discovery needs. These solutions are both RESTful, follow design principles of DTN and were designed to be adapted to the IoT context.

BoaP

The first programming-support is resource-oriented. A transposition of CoAP for a BP-binding has been proposed. This transposition is composed of fundamental adjustments (addressing, multicast, confirmable message and caching), and enhancements (multi-payload, accept option, PATCH method). Stemming from these fundamental adjustments and enhancements a prototype, named BoaP, has been implemented. BoaP is compliant with many CoAP features and concepts such as its serialization, message deduplication, request/response matching, URI identification, CON/NON particularities, options. Being in a BP network, and according to the store-carry-and-forward principle, messages are stored on and carried by intermediary nodes. Taking benefit of this, a caching mechanism has been implemented too. This mechanism enables intermediary nodes to answer a request they carry if they also carry a fresh response for a similar request. This caching mechanism has two consequences: it increases the number of fulfilled requests and it shortens their round-trip-time.

Because of the design principles of DTN, and more precisely the fact that the number of round-trip exchanges should be minimized, timeout of retransmissions of CON messages must receive particular considerations. Retransmissions should be avoided, and default values of time-out for CoAP shall not be reused in a DTN. This proposal natively inherits all CoAP characteristics (such as low overhead, straightforward mapping to HTTP, multicast support) and respects all DTN design principles in the same time.

It can be wondered if the Bundle Protocol is the best match for a transposition of CoAP into an Intermittently Connected Network (ICN). Indeed, if BP is the *de facto* standard protocol to exchange messages in DTNs, it is not adopted by the whole community [139]. Nevertheless, and despite its known flaws [140], BP is currently the only protocol for opportunistic networks that is technically defined in depth, adopted by a large community according to the number of BP implementations and considered to be deployed by national agencies such as the NASA [141]. Additionally, the contributions of this thesis were investigated to be as close as existing standard, and BP is the closest protocol to reach a standard status that supports message exchanges in ICN.

Experiments

BoaP performance has been evaluated: it has been deployed in a small physical network, then emulated in a network of 50 nodes following two different mobility patterns: an artificial one (Levy Walk) and reality-based one (from CRAWDAD).

The emulations relied on an original deployment using Docker Swarm and focused on reproducibility. Unlike simulations, emulations are not usual to evaluate DTN systems. These experiments aimed to ensure that the prototype BoaP worked well in DTN networks as well as validating the proposal of CoAP transposition. These evaluations provided the conclusion that it is reliable enough for the networks it was deployed in.

During emulations, network links were simulated in a binary manner: they were either null (disconnected), either virtually not limited (connected). The realism of network links can then be questioned: there is no radio-interferences, no delay reproducing the connection between two nodes, and the data-rate is not limited at all. Nevertheless, modelizing network-links that way is still acceptable in the research community within which evaluations are commonly done through simulations. Moreover, since the current literature shows no consensus on experimenting methods and approximations, it is unclear if this specific approximation is a real threat to validity.

LILWENE

The second programming-support proposed was a service-oriented solution. Benefiting from a tested, reliable resource-oriented middleware, BoaP served has a base of this solution named LILWENE[1]. This solution is RESTful and follows a service-oriented approach (SOA).

In this proposal, four principles are suggested to help designing an optimized, reactive and efficient SOA middleware in DTN environments. These four principles fully take into considerations the design principles of DTN, are adapted to IoT context and are conceived with service-oriented approach in mind. They are: *1.* Local Registry (each node takes part of the advertisement/discovery aspect by hosting its own registry), *2.* Full and Concise Description (so clients can successfully request the services while keeping at a minimum the number of round-trip exchanges), *3.* Correction (client applications can correct service descriptor stored locally believed to contain errors), and *4.* Update (upon reception of a new service descriptor, local registries notify all local client that requested similar descriptions).

LILWENE proposal contains service description guidelines. All mandatory and optional fields are listed. These fields enclose traditional information such as identification, entrypoint and requesting interface, as well as particular information related to the specific environment. This particular information includes service deadline (date at which a service is not expected to be available anymore), service agenda (that inquires when a service's host is sleeping), and a geographic restriction (for services that provide their capabilities in a restricted area).

These documents are advertised, and discovered, through a publish/subscribe mechanism relying on BP non-singleton EID. This pub/sub mechanism supports

[1]Light services middleware for delay tolerant Web-of-things Networks

several topics, and defines how to discover topics that can be dynamically created. A client application can request description documents through regular requests to its local registry that will then send back matching descriptions. Service invocation is also presented and some options are also suggested. Finally, this proposal contains some implementation elements as well as an API to provide all required details to implements LILWENE.

6.2 Perspectives

Many open challenges that would be interesting to solve remain. Solving those would undoubtedly enhance the contributions of this thesis but are not developped in this document as they are considered to be out of the scope of this current work. These open challenges affect for instance security aspect, service composition in LILWENE, real-world experiments and a BoaP/CoAP proxy. Added to these open challenges, and as a continuity of this thesis, three specific works may offer valuable outcomes, and are presented below. They are: improving BoaP caching feature, implementing network healing means and assessing performances in DTNs relatively to idealistic ones.

Improving Caching Feature

The current implementation of BoaP-caching feature can be optimized in order to reduce its impact on the volume of data emitted by nodes. Indeed, in its current state a node automatically answers with a cached response whenever a matching-request is received by this node. Meanwhile, even if the request was answered, the bundle containing this request is not deleted. Worse than not deleted, this bundle is still disseminated in the network as any other regular bundle. Since the request was fulfilled, it should not be disseminated in the network anymore as it is just waste, pollution. The consequence is that when another node receives this bundle, containing a request that has already been fulfilled, the receiving node may also answer to it. This simplistic implementation does not optimize medium usage at all. Another issue with the current implementation of BoaP-caching feature is that bundles of interest (valid and fresh responses) are stored twice on the node: one in the BP daemon storage and another at the application level. Indeed, since IBR-DTN does not support a direct access to its bundles database, these bundles, and more especially their payload, need to be stored on a different location. This means that the node's storage is not optimized since there is a duplication of data.

An optimization can be implemented. Assuming that the caching endpoint on a node has a read-access to all stored bundles then an additional verification can be implemented. When a request is received and that the node has a matching and fresh response then, before sending this response, the caching endpoint should verify whether a locally stored bundle already carries a response for the request's source or not. If the node has one, it implies that another node already answered to the request, hence no more response is needed. In other words, before a node answers to a request, it should check if a response is already stored locally. This would optimize medium usage and save up both energy and storage.

This optimization could be further enhanced. In the case a request is received and that this request has already been fulfilled (i.e., a bundle locally stored is sent to the request's source with a matching response) then the bundle that contains the request could be dropped. Dropping this bundle would stop its dissemination hence optimizing medium usage and saving energy and storage again. However, IBR-DTN does not permit such bundle manipulation that does not respect a strict layered design. Nonetheless, the Metadata Extension Block (MEB [7]) could be exploited to implement this feature with BP. This extension block can be used to carry additional details related to the content of the payload. These details can then be used to trigger specific actions on these bundles. In this case, there could be two details: the token of the message (that is shared between request and response), and the message type (either a request or a response). By exploiting these two information, a IBR-DTN daemon can then detect and drop the bundle containing a request when it carries both matching request and response.

A MEB embedding a code to execute has also been proposed by Borrego et al. [122]. In their proposal, the MEB embeds routing-logic. This not only release the daemon to take care of the routing algorithm but also enable to dynamically support new routing algorithms without nodes update.

Network Healing

Network healing mechanisms aim to cure network from unwanted messages by disseminating vaccines. A vaccine is a short message that informs that specific messages can be removed from the cache as their destination already received it or do not need it anymore. Messages that can be removed are all messages that have been received and all messages that are not needed anymore. For instance, in LILWENE, when a service dies it publishes a service description with a version-number equals to zero. This specific value, zero, is explicitly used to request registries to delete this service description so it is not requested anymore hence avoiding to disseminate requests that would not be answered. This mechanism, likened to healing ones, could be enhanced: node storing both this service description and requests for this service could drop the requests to save resources (i.e., medium, energy and storage usage). Network healing challenge has already been identified and tackled in [142]. In this publications, two means are proposed to reduce the network waste of "*invocation leftovers*". The first means is named *safe healing*. Once a client received the first response, it produces a specific control message. This control message requests the relay nodes to delete its request and all responses. The second means is named *aggressive healing*. This healing is produced by the provider when it emits its response. The control message requests other nodes to delete the matching request.

Other mechanisms that explicitly request to cancel the dissemination of a message exist, see [143, 144, 145].

Adding network healing mechanism, that could be implement by exploiting MEB, to both BoaP and LILWENE would surely benefit both of them by improving the medium usage hence saving up both energy and storage.

Assessing Idealistic Performance in DTNs

DTNs can be modelized as Time-Varying Graphs (TVG [146]), or Temporal Networks [147], and theoretical studies on TVG can be applied in DTN environments. In other words, some known algorithms for TVG are transposable in DTNs, and some quantitative characteristics can be extracted from it. Quantitative characteristics presented along the evaluations of BoaP (node density, coverage ratio, average degree of node against time, CDF of contacts durations) can then be completed. For instance, as presented by Casteigts [146], horizons can be calculated. The horizon of a node is the set of all nodes that, according to the evolution of the network, are reachable in the future. In other words, if a node emits a message, its horizon is all nodes that can receive this message.

TVG can help to better characterize inter-contacts and the general communicability of the network. TVG can also be a reference to precisely evaluate a DTN system performance. Indeed, by exploiting horizons, it is possible to determine, with precision, when a message can be received by another node. For instance, in the context of BoaP or LILWENE deployment, it is then possible to determine the exact and earliest date a request can be received by the server(s), and when their response(s) can be received by the client. It is also possible to determine if such message cannot be received at all. Therefore, for every message it is possible to determine if it can be received (delivery ratio), and to calculate idealistic duration needed to be received (RTT). Thus, these idealistic delivery ratio and durations can be correlated with experimental ones in order to evaluate the performance of a communication middleware (such as BoaP). Nowadays, evaluations are not cross-checked and usually done in an absolute manner, i.e. without comparing it to anything else.

The research community shows an emerging interest for experiment fidelity. Investigating further horizons and combining it with an open, emulation-based experimenting platform focused on reproducibility would enable a new way of testing algorithms, and protocol implementations. These tools would allow more realistic evaluations and help the community to reach a consensus on experimenting methods.

Appendices

A
Experiments Review: Table

In section 4.2.2.2, an experiment review references a table of 86 articles that were reviewed to gather quantitative characterizations and to justify chosen parameter value (density, radio-range, mobility) for this thesis's experiments. This table is presented below.

Table A.1: Experiment Parameters Review

Ref	**Area (m^2)**	**Radio Range**	**# of nodes**	**Coverage Ratio**	**Mobility**	**Simulator**	**Duration**
[148]	1.23E+5	100					100 s
[149]	1.95E+6		100		RWP	GloMoSim	1000 s
[150]	1.00E+6	150			RWP	GloMoSim	1000 s
[151]	1.00E+6	150			RWP	GloMoSim	1000 s
[152]	1.00E+6	100			RWP	SWANS	1000 s
[153]	4.00E+2	5	20	3.93	Close to none		100000 s
[154]	1.58E+7		126	0.009	map based	ONE	12 h
[155]	1.00E+8				custom RWP		12 h
[156]	1.80E+6	250	120	13.089		ns-2	120 s
[157]			34			DTNSIM	160 d
[158]	2.50E+9	250	450	0.035	Traffic simulator	OMNET++	2 h
[159]	2.50E+3	300	15	1696	RWP	Qualnet	200 s
[160]			92		KAIST with AP	ONE	24 h
[161]	3.00E+4	10	100, 2000	1.047, 20.94		ONE	24 h
[162]	2.50E+9	1000, 5000	158	2.37			300 s
[163]	6.00E+4, 1.00E+6	50			RWP	ns-2	3000 s

Ref	Area (m^2)	Radio Range	# of nodes	Coverage Ratio	Mobility	Simulator	Duration
[32]	4.50E+6	50, 100	50	0.087, 0.349	custom	custom	3600 s
[164]	2.06E+4		3540		Bologna car traffic	custom ONE-like	3600 s
[165]	1.00E+6	150	80	5.65	RWP	ns-2	400 s
[166]	2.10E+4, 4.00E+4	30	50, 200	3.53, 6.72	RWP	GloMoSim	4500 s
[167]	1.00E+6		64		RWP, Manhattan Grid		4500 s
[168]	3.60E+5	200	10, 90	3.49, 31.41		ns-2	500 s
[169]	8.37E+5, 1.68E+6		30, 60			ns-2	500 s
[170]	9.00E+4		12			OMNET++	60 min
[171]	1.21E+4, 6.25E+4	30	25, 64	1.13, 14.95	RWP	GloMoSim	600 s
[172]	2.50E+5	90			RWP	ONE	70000 s
[25]	1.60E+7	250	100	1.23	Social network baed		8 h
[173]	1.56E+6	125			HCMM [174]		90000 s
[175]						contiki, cooja	
[176]		275	4			CORE	
[177]	9.00E+4	10	30, 60	0.10, 0.21	RWP	custom	
[178]	9.00E+4	10	50	0.17	RWP	custom	
[41]	4.00E+6	25	100		RWP, random walk	custom	
[179]	1.44E+6	250	10	1.36		custom	
[143]	4.00E+4	15	16	0.28		custom	
[180]			120	1		custom	
[181]			100		CBM[182]	DTN Java	
[183]						Exata	
[184]	2.25E+6	250	10, 400	0.87, 34.9	RWP	GloMoSim	
[144]	1.00E+5, 5.00E+4	250	10, 250	39.27, 490.87	RWP	GloMoSim	
[185]	9.00E+4	50	50	4.36	RWP	GloMoSim	

Ref	Area (m^2)	Radio Range	# of nodes	Coverage Ratio	Mobility	Simulator	Duration
[186]	1.00E+6	250	50	9.81	Steady-State RWP	GloMoSim	
[187]	1.00E+6	250	50	9.81	RWP	J-Sim	
[188]	1.00E+6	100				J-Sim	
[189]	2.50E+5	30 60	2, 2500	0.022, 28.27	RWP	Madhoc	
[190]	1.20E+2	30	20	471.24	Random Walk	MATES[191]	
[192]	4.50E+5	10 250	50	0.035, 21.81		Monarch	
[42]	2.50E+7		40		RWP	ns-2	
[193]	1.00E+6, 4.50E+5	250	50	9.82, 21.82	RWP	ns-2	
[194]	4.50E+5	250	50	21.82	RWP	ns-2	
[195]	5.00E+5		50		RWP	ns-2	
[196]	1.20E+6	200	10	1.0471975512		ns-2	
[197]			42, 70			ns-2	
[198]	2.56E+4	40	50, 250	9.82, 49.09		ns-2	
[199]	9.00E+4	10	100	0.35		ns-2	
[200]	4.00E+6	250	200		Random Direction Model	OMNET++	
[107]	1.00E+6	20	20	0.025	RWP	OMNET++	
[201]	1.00E+6	200	50	6.28	RWP	OMNET++	
[201]	4.00E+6	200	100	3.14	RWP	OMNET++	
[202]	1.00E+6		100		RWP	OMNET++	
[203]	5.00E+2		50			OMNET++	
[204]	4.00E+8	250	260000	127.63		OMNET++	
[205]	4.00E+6	100	70	0.55	[206]	ONE	
[207]	1.00E+6	20	18, 66	0.023, 0.083	RWP	ONE	
[208]	1.53E+7		140		RWP, Helsinki City Scenario	ONE	
[207]	1.53E+7	20 50	10, 200	8.2E-004, 0.10	RWP, SPMBM, OPP	ONE	

Ref	Area (m^2)	Radio Range	# of nodes	Coverage Ratio	Mobility	Simulator	Duration
[209]	4.00E+6	10	70	0.0055		ONE	
[210]	2.50E+7, 1.50E+8		51			OPNET	
[211]		100	600			OPNET	
[212]						PlanetLab	
[213]	6.40E+9	500	500	0.061	RWP	STRAW	
[214]	1.00E+6	200	50	6.28	RWP	SWANS	
[215]	8.10E+5	150			RWP	SWANS	
[216]	4.00E+4	10	50, 200	0.39	Simplified Way Point	SWANS	
[217]			100			Waxman	
[218]	1.00E+3	200	16, 32	2010, 4021	[219]		
[220]					Dartmouth		
[221]					Generated via commercial tool		
[222]					H06, MIT, ETH		
[223]	1.00E+6	10	44	0.014	HCMM		
[224]					KAIST		
[225]	9.00E+6	100			Localized Random Walk		
[226]			4				
[227]	6.25E+4	20	8, 72	0.16, 1.45			
[228]	1.00E+4		20, 100				
[229]	3.00E+5, 7.68E+5	50	53	1.39			

B

Service Descriptor: Binary Format and Parameters

In section 5.2, LILWENE service descriptor is presented. This appendix details the binary format of this descriptor for all its fields as well as service parameters.

B.1 LILWENE Descriptors

The different fields of service descriptors were details in section 5.2 and are :

- Identification,
- Interface of the implemented methods and enabled options,
- Accepted inputs and outputs,
- Ontology document,
- Non-functional details:
  - version,
  - deadline,
  - agenda,
  - geographic restriction,
  - optional free field.

All these details are embedded in a binary document following the format described below according to the syntax:

```
<identification> 0x20 <interface> <inputs> 0xffff <outputs> 0xffff
<ontology document id> <version> <deadline> <agenda> <geographic
restriction> 0x3b3b <parameters>
```

The field `<identification>` is encoded in ASCII and the value `0x20` marks the end of this field. `<interface>` a fixed-size field of 1 byte. The lists of inputs and outputs fields are both ended by the value `0xffff`. These fields are a list of `unsigned short` values with the special value `0xefff` used as the dash-optimization: if a subset of a list is constituted of successive `unsigned short`, then a dash '-' can be used between the first and last integer of this subset of successive values. This shortens lists, for example the list `3,5,8,9,10,11` becomes `3,5,8-11`. The field `<ontology document id>` is a fixed-size field of 8 `int` that contains the MD5 sum of the ontology document embedded in the next payload block of a bundle containing its descriptor. The field `<version>`, and `<deadline>`, are both fixed-size fields of respectively 1 byte and 1 long. Then comes the field `<agenda>`: it is encoded in ASCII, the value `0x3b` marks the ends of an agenda entry as it may need several ones. Following the `<agenda>`, the field `<geographic restriction>`, when present, is contained in the remaining bytes between the last agenda entry and the final mark of `0x3b3b`. Because the end of this field is marked by `0x3b3b` there is a possibility of error in the parsing. Indeed, there is no assumption that this value is not in the `<geographic restriction>` field. To avoid parsing error, the first byte of this field must represent the size in octets this fields needs. Its size may differs according to the complexity of the described area. Thanks to this, the parser knows beforehand how many octets it should read. Removing the `0x3b3b`-marker would be possible if the field `<geographic restriction>` was not optional. However, without the marker the parser cannot determine if it reads this field or the `<parameter>` one. As for the parameter format it is detailed below.

B.2 Parameters

Some services may require parameter as input. It can be simple query parameters to set output options, e.g. `GET dtn://temp-sensor41/status?unit=celsius`, or a more complicated one such as `GET dtn://camera-a94/picture?grayscale=1&resize_h=256&resize_w=256&epoch_date=624245100`. A sub-document of the service description details the available parameters constraints. Each parameter must be described through its different fields that are: id, type, flags/group and constraint.

In order to ease the reading of the following a human readable representation is used according to the syntax:

`<id>:<usage>:<argument>`, where `<usage>` include type and flags/group. For deeper details on the parameter description format see B.2 where it is fully described.

Parameter Id

First of all, the parameters, just like the service providers, must be identified. To ensure this, the variable id is given and is the means of identification. This is this id that must be used in the request to set this parameter. Each parameter id must be unique within a service description, an example could be:

```
unit:  [..]
```

This field is limited to alphanumeric values and must start with a letter.

Parameter Types

The different types of parameters are the well-known: `boolean`, `byte`, `int`, `long`, `float`, `string` and `enum`. After its id, the document entry must inform on the type of the parameter:

```
unit:string [..]
```

This parameter is encoded in a byte, see Tab. B.1. Only four bits are used to encoded this information. The used bits are the LSBs, as for the four MSBs: they are used for parameter constraints, explained further.

Table B.1: Parameters Types

Value	Type
0b0000	boolean
0b0001	signed byte
0b0010	signed int
0b0011	signed long
0b0101	signed float
0b0110	string
0b0111	enum

According to Tab. B.1, the previous description becomes:

```
unit:7 [..]
```

Parameter Methods

Parameters may not be used, or useful, for every method. For instance, a fan service could be requested by `GET dtn://car/fan13/speed?unit=rpm` to get the current speed of the fan and by `POST -data "unit=rpm&speed=120" dtn://car/fan13/speed` to set the speed.

In the previous examples, the parameter `speed` is used with the POST but method not the GET one. In the meantime, the parameter `unit` is used for both methods. In order to inform the clients of the matching between parameters and methods, the parameter field named `methods` must be set. For instance, with the previous example, the parameters description would be:

```
unit:7,GET-POST [..]
speed:2,POST [..]
```

Same as the parameter type, this field is encoded as a byte. The two MSB are left cleared, while the bits from the 6th down to the 1st are: OBSERVE(5), DELETE(4), PATCH(3), POST(2), PUT(1), GET(0). Hence the previous examples become:

```
unit:7,5 [..]
speed:2,4 [..]
```

Assuming that the ontology describes the returned speed unit as `rpm` by default then this parameter is optional with the GET method yet remaining required for the POST. Following this byte, a second one informs if the field is optional. The two bytes are encoding in the same format. The previous examples become:

```
unit:7,5,4 [..]
speed:2,4,4 [..]
```

Parameter Groups

In the previous POST example there are two different parameters: `unit=rpm` and `speed=120`. If the fan service understands revolutions per minute (`rpm`) as much as Hertz (`H`) as much as radiant per second (`w`) units, then these requests: `"unit=rpm&speed=120"`, `"unit=H&speed=2"` and `"unit=w&speed=13"` produce the same result. Other requests with other set of parameters that this service could received are: `"rpm=120"`, `"H=2"` and `"w=13"` producing the same result again. So far, this is a matter of ontology, but let's take a look at these parameters descriptions:

```
unit:7,5,4 [..]
speed:2,4,4 [..]
rpm:2,4,4 [..]
h:2,4,4 [..]
w:2,4,4 [..]
```

According to these parameters' descriptions a client cannot determine which subset of parameters it should use for a POST request. These descriptions only inform that these parameters can be used with POST requests, nevertheless the client should not use all of them. Indeed, sending a request with `"unit=H&speed=2&H=2"` would be redundant, so the clients need a means to determine which parameter(s) need(s) to be used with which other parameter(s). This issue can be solved by providing a `byte` within which each bit represents a unique group. A group is a subset of parameters that must be used together. A group is represented by a bit of the parameter groups field and all parameters sharing a common bit represent a group, and thus must be used together. With the previous example, this field would be:

```
unit:7,5,4,1 [..]
speed:2,4,4,1 [..]
rpm:2,4,4,2 [..]
h:2,4,4,4 [..]
w:2,4,4,8 [..]
```

Meaning that `unit` and `speed` must be used together while `speed`, `rpm`, `h` and `w` should be used alone.

Parameter Constraints

Parameters may be constrained, i.e., `byte`, `int`, `short`, `long` and `float` could have some range restrictions. `string` type parameters could be constrained as alpha-only or alphanumeric values. To inform clients of the parameter restrictions it may have the three MSBs of the byte defining the type of the parameter are used as detailed on the Tab. B.2.

Table B.2: Parameters Constraints

Value	boolean	byte, int, long, float	string	enum
`0b0000`	Default is false	Max. value defined	Alpha only	Single value
`0b0001`	Default is true	Min. value defined	Integers only	List
`0b0010`	Reserved	Range defined	Decimals only	Sorted list
`0b0011`	Reserved	Multi-range defined	Alphanum. only	Reserved
`0b0100`	Reserved	Reserved	Alphanum. and space only	Reserved
`0b0101`	Reserved	Reserved	URL encoded	Reserved
`0b0101`	Reserved	Reserved	Regexp defined	Reserved
`0b1000`	No constraint			

`boolean` constraints

Default value When a `boolean` is optional the service may inform whether, by default, the value is true or false.

`byte, int, long, float` constraints

Maximum/Minimum value defined There are many reasons why a number should be restricted to a maximum or minimum value. An example could be the maximum speed of a fan or the minimum temperature an AC can aim for. The clients being informed of these extreme values can then respect these constraints and avoid getting *4.09 Conflict* error response.

In the following examples the `speed` parameter has a maximum value defined that is `360` and the `ac` parameter has a minimum value defined that is `10`.

```
speed:2,4,4,1:360
ac:0x12,4,4,1:10
```

Range defined For the same reasons explained above a number should be restricted between two extreme values. A motor may be restricted between $-5V < tension < +5V$. The following example demonstrate such a case:

```
tension:0x22,4,4,1:-5_5
```

Multi-range defined Some service may not support whole range of values or may allow several range of values. For instance a motor may not be able to apply the tension $0V$ as residual electronic tension may persist so the service would not accept the full range $-5V < tension < +5V$ but two ranges that would be $-5V < tension < 0, 0 < tension < +5V$, hence excluding the zero volt tension using another parameter to stop the motor. Multi-ranged parameters can be defined as follow:

```
tension:0x32,4,4,1:-5_0,0_5
```

`string` **constraints** As for the `string` parameters the constraints *Alpha only*, *Integers only*, *Decimals only*, *Alphanumeric only*, *Alphanumeric and space only* and URL *encoded* speak for themselves and do not require anymore details. Nevertheless, the *Regexp defined* constraint must include the specific regular expression that the `string` must follow. The regular expression syntax must follow the Extended Regular Expressions syntax. If the string is expected to be a valid IPv4 address the parameter could be described as follow:

```
    tension:0x32,4,4,1:(25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)
\.(25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)
\.(25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)
\.(25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)
```

Note that instead of using complex regexps it is wiser to use more parameters with simpler constraints. The regular expression for an IP address is really complex for resource-constrained things. Using four `bytes` would be far simpler and would save some CPU processing and thus energy too.

`enum` **constraints and specificity** The constraint *Single value expected*, *List expected* and *Sorted List expected* speak for themselves and do not require any additional detail. However, it should be noticed that `enum` type is the only parameter that must include some details. These details are in the form of a list of the possible values of the `enum`. An enumeration of the different CoAP request types would be:

```
cpRqType:7,4,0,1:con,non,ack,rst
```

No constraint If ever the MSB of the parameter type field is set then there is no constraint on the parameter and the clients are free to use them as needed.

Parameter Description Format

So far, representation of the parameter descriptions is made to be human readable with the syntax:

```
<id>:<usage>:<argument>
```

Nevertheless, the sub-document itself is in a binary format. The `<id>` field must be composed of alphanumeric ASCII values. A separator after the first field marks the beginning of the `<usage>` field, its value is `0x20`. The `<usage>` field is a fixed size field of 4 bytes long. The remaining bytes encode the `<argument>` field, if any. When a range or a multi-range is defined, there is not any separator as the length of each number is known beforehand. If the argument represents a regular expression, then it is encoded in base64 ASCII values. In the case of an `enum` the same restrictions of the `<id>` field must be respected and a separator marks the beginning of the next `enum` value, the separator value is `0x2c`. To mark the end of a parameter description the value `0x20` is written.

In the following table, Tab. B.3, parameter examples are encoded in hexadecimal:

```
tension:0x22,4,4,1:-5_5
cpRqType:7,4,0,1:con,non,ack,rst
```

Table B.3: Binary Overview

id	sep.	usage					argument
		constraint	type	methods	optional	group	
tension 74 65 6e 73 69 6f 6e	: 20	range 2	byte 2	POST 4	POST 4	1 1	from -5 to 5 B5 20
cpRqType 63 70 52 71 54 79 70 65	: 20	single value 0	enum 7	POST 4	 0	 1	con,non,ack,rst 63 6f 6e 2c 6e 6f 6e 2c 61 63 6b 2c 72 73 74 20

B.3 Description Examples: Binary Content

Simple Temperature sensor

Following hexadecimal values represent the service descriptor presented in 5.2.3.

```
    64 74 6e 3a 2f 2f 66 72 69 64 67 65 2d 73 65 6e 73 6f 72 2f 74 65 6d
70 65 72 61 74 75 72 65 20 41 ff ff 03 ff ff 95 28 59 ec 92 e9 f3 da 8b
57 75 0e ee 77 7a fa 13 6a 85 70 2d 3b 3b 76 61 72 69 61 74 69 6f 6e 20
22 40 00 01 00 0a 20
```

Configurable Light Bulb

Following hexadecimal values represent the service descriptor presented in 5.2.3.

```
   64 74 6e 3a 2f 2f 6c 69 67 68 74 30 61 2f 61 63 63 65 73 73 20 03 ff
ff 03 ff ff 95 28 59 ec 92 e9 f3 da 8b 57 75 0e ee 77 7a fa 10 5f 3d 30
29 30 20 39 20 2a 20 2a 20 2a 20 33 32 34 30 30 3b 3b 74 75 72 6e 20 04
20 03 6f 6e 2C 6f 66 66 20 68 65 78 20 56 22 01 5b 30 2d 39 61 2d 66 41
2d 46 5d 2b 20 72 20 82 22 02 20 67 20 82 22 02 20 62 20 82 22 02 20
```

Publications

[1] N. Le Sommer, L. Touseau, Y. Mahéo, M. Auzias, and F. Raimbault. “A Disruption-Tolerant RESTful Support for the Web of Things”, in 4rd International Conference on Future Internet of Things and Cloud, Vienna, Austria, August 2016, IEEE.

[2] M. Auzias, Y. Mahéo and F. Raimbault. “CoAP over BP for a Delay-Tolerant Internet of Things”, in 3rd International Conference on Future Internet of Things and Cloud, Rome, Italy, August 2015, IEEE.

Bibliography

[1] Luigi Atzori, Antonio Iera, and Giacomo Morabito. The Internet of Things: A survey. *Computer Networks*, 54(15):2787–2805, 2010.

[2] Jayavardhana Gubbi, Rajkumar Buyya, Slaven Marusic, and Marimuthu Palaniswami. Internet of things (iot): A vision, architectural elements, and future directions. *Future generation computer systems*, 29(7):1645–1660, 2013.

[3] Anders Lindgren, Avri Doria, Jan Lindblom, and Mattias Ek. Networking in the land of northern lights: two years of experiences from DTN system deployments. In *Proceedings of the 2008 ACM workshop on Wireless networks and systems for developing regions*, pages 1–8. ACM, 2008.

[4] Vinton G. Cerf, Scott C. Burleigh, Robert C. Durst, Kevin Fall, Adrian J. Hooke, Keith L. Scott, Leigh Torgerson, and Howard S. Weiss. Delay-Tolerant Networking Architecture. IETF RFC 4838, November 2007.

[5] Kevin Fall. A Delay-Tolerant Network Architecture for Challenged Internets. In *Proceedings of ACM SIGCOMM03*, August 2003.

[6] Keith Scott and Scott Burleigh. Bundle Protocol Specification. IETF RFC 5050, April 2007.

[7] Susan Flynn Symington. Delay-Tolerant Networking Metadata Extension Block, May 2011.

[8] Dominique Guinard, Vlad Trifa, Friedemann Mattern, and Erik Wilde. From the internet of things to the web of things: Resource-oriented architecture and best practices. In *Architecting the Internet of Things*, pages 97–129. Springer, 2011.

[9] Roy Thomas Fielding. *Architectural styles and the design of network-based software architectures*. PhD thesis, University of California, 2000.

[10] W3C. Architecture of the world wide web, volume one.

[11] Zach Shelby, Klaus Hartke, and Carsten Bormann. Constrained Application Protocol (CoAP). IETF Internet Draft, June 2014.

[12] Zach Shelby. Constrained RESTful Environments (CoRE) Link Format, August 2012.

[13] Luciana Pelusi, Andrea Passarella, and Marco Conti. Opportunistic Networking: Data Forwarding in Disconnected Mobile Ad Hoc Networks. *IEEE Communications Magazine*, 44(11):134–141, November 2006.

[14] Vince Cerf, Scott Burleigh, Adrian Hooke, Leigh Togerson, Robert Durst, Keith Scott, Eric Travis, and Howie Weiss. Interplanetary Internet (IPN): Architectural Definition. IETF Internet Draft, May 2001.

[15] Zhensheng Zhang. Routing in intermittently connected mobile ad hoc networks and delay tolerant networks: overview and challenges. *IEEE Communications Surveys & Tutorials*, 8(1):24–37, 2006.

[16] Scott Burleigh. Interplanetary overlay network. In *Consumer Communications and Networking Conference (ieeeccnc), Proceedings of the 4th*, pages 222–226. IEEE, 2007.

[17] Marc Blanchet. Postellation: an enhanced delay-tolerant network (dtn) implementation with video streaming and automated network attachment. In *SpaceOps 2012*, page 1279621. 2012.

[18] Michael Doering, Sven Lahde, Johannes Morgenroth, and Lars Wolf. IBR-DTN: an efficient implementation for embedded systems. In *Proceedings of the 3rd ACM workshop on Challenged networks*, pages 117–120. ACM, September 2008.

[19] Dtn2 documentation.

[20] Daniel Brown, Richard Altmann, Daniel Brown, and Ronald in 't Velt. DTN IP Neighbor Discovery (IPND). IETF Draft, May 2016.

[21] Amin Vahdat and David Becker. Epidemic Routing for Partially Connected Ad Hoc Networks. Technical report, Duke University, April 2000.

[22] S. Y. Ni, Y. C. Tseng, Y. S. Chen, and J. P. Sheu. The Broadcast Storm Problem in a Mobile Ad Hoc Network. In *Proceedings of the 5th annual ACM/IEEE international conference on Mobile computing and networking*, pages 151–162. ACM/IEEE, 1999.

[23] Thrasyvoulos Spyropoulos, Konstantinos Psounis, and Cauligi S Raghavendra. Single-copy routing in intermittently connected mobile networks. In *Sensor and Ad Hoc Communications and Networks, 2004. IEEE SECON 2004. 2004 First Annual IEEE Communications Society Conference on*, pages 235–244. IEEE, 2004.

[24] Ramon Martí, Sergi Robles, Abraham Martín-Campillo, and J Cucurull. Providing early resource allocation during emergencies: The mobile triage tag. *Journal of Network and Computer Applications*, 32(6):1167–1182, 2009.

[25] Paolo Costa, Cecilia Mascolo, Mirco Musolesi, and Gian Pietro Picco. Socially-Aware Routing for Publish-Subscribe in Delay-Tolerant Mobile Ad Hoc Networks. *IEEE Journal On Selected Areas In Communications (JSAC)*, 26(5):748–760, June 2008.

[26] Julien Haillot and Frédéric Guidec. A Protocol for Content-Based Communication in Disconnected Mobile Ad Hoc Networks. *Journal of Mobile Information Systems*, 6(2):123–154, 2010.

[27] Thrasyvoulos Spyropoulos, Konstantinos Psounis, and Cauligi S. Raghavendra. Spray and Wait: an Efficient Routing Scheme for Intermittently Connected Mobile Networks. In *2005 ACM SIGCOMM workshop on Delay-tolerant networking (WDTN'05)*, pages 252–259, Philadelphia, PA, USA, 2005. ACM.

[28] Thrasyvoulos Spyropoulos, K. Psounis, and C.S. Raghavendra. Spray and focus: Efficient mobility-assisted routing for heterogeneous and correlated mobility. In *Pervasive Computing and Communications Workshops, 2007. PerCom Workshops '07. Fifth Annual IEEE International Conference on*, pages 79–85, March 2007.

[29] John Burgess, Brian Gallagher, David Jensen, and Brian Neil Levine. MaxProp: Routing for Vehicle-Based Disruption-Tolerant Networks. In *25th International Conference on Computer Communications (INFOCOM'06)*, pages 1–11, Barcelona, Spain, April 2006. IEEE CS.

[30] Jérémie Leguay, Timur Friedman, and Vania Conan. Dtn routing in a mobility pattern space. In *Proceedings of the 2005 ACM SIGCOMM workshop on Delay-tolerant networking*, pages 276–283. ACM, 2005.

[31] Abraham Martín-Campillo and Ramon Martí. Energy-efficient forwarding mechanism for wireless opportunistic networks in emergency scenarios. *Computer Communications*, 35(14):1715–1724, 2012.

[32] Anders F. Lindgren, Avri Doria, and Olov Schelen. Probabilistic Routing in Intermittently Connected Networks. In *Proceedings of the The 1st International Workshop on Service Assurance with Partial and Intermittent Resources (SAPIR 2004)*, Fortaleza, Brazil, August 2004.

[33] Chiara Boldrini, Marco Conti, Jacopo Jacopini, and Andrea Passarella. HiBOP: a History Based Routing Protocol for Opportunistic Networks. In Marco Conti, editor, *International Symposium on a World of Wireless, Mobile and Multimedia Networks (WoWMoM 2007)*, pages 1–12, Helsinky, Finland, 2007. IEEE CS.

[34] Chiara Boldrini, Marco Conti, and Andrea Passarella. Exploiting users' social relations to forward data in opportunistic networks: The hibop solution. *Pervasive and Mobile Computing*, 4(5):633–657, 2008.

[35] Long Vu, Quang Do, and Klara Nahrstedt. 3r: Fine-grained encounter-based routing in delay tolerant networks. In *World of Wireless, Mobile and Multimedia Networks (WoWMoM), 2011 IEEE International Symposium on a*, pages 1–6. IEEE, 2011.

[36] Ali Makke, Nicolas Le Sommer, and Yves Mahéo. TAO: A Time-Aware Opportunistic Routing Protocol for Service Invocation in Intermittently Connected Networks. In *8th International Conference on Wireless and Mobile Communications (ICWMC 2012)*, pages 118–123, Venice, Italy, June 2012. Xpert Publishing Services.

[37] Eiko Yoneki, Pan Hui, Shu-Yan Chan, and Jon Crowcroft. A Socio-Aware Overlay for Publish/Subscribe Communication in Delay Tolerant Networks. In *10th ACM/IEEE International Symposium on Modeling, Analysis and Simulation of Wireless and Mobile Systems (MSWiM)*, pages 225–234, Crete Island, Greece, October 2007. ACM.

[38] Elizabeth M. Daly and Mads Haahr. Social Network Analysis for Routing in Disconnected Delay-tolerant MANETs. In *Proceedings of the 8th ACM International Symposium on Mobile Ad Hoc Networking and Computing (Mobihoc'07)*, pages 32–40, Montreal, Quebec, Canada, 2007. ACM.

[39] Pan Hui, Jon Crowcroft, and Eiko Yoneki. Bubble rap: Social-based forwarding in delay-tolerant networks. *IEEE Transactions on Mobile Computing*, 10(11):1576–1589, 2011.

[40] David J Goodman, Joan Borras, Narayan B Mandayam, and Roy D Yates. Infostations: A new system model for data and messaging services. In *Vehicular Technology Conference, 1997, IEEE 47th*, volume 2, pages 969–973. IEEE, 1997.

[41] Sushant Jain, Rahul Shah, Waylon Brunette, Gaetano Borriello, and Sumit Roy. Exploiting Mobility for Energy Efficient Data Collection in Wireless Sensor Networks. *MONET*, 11(3):327–339, 2006.

[42] Wenrui Zhao, Mostafa Ammar, and Ellen Zegura. A Message Ferrying Approach for Data Delivery in Sparse Mobile Ad Hoc Networks. In *Proceedings of the Fifth International Symposium on Mobile Ad Hoc Networking and Computing (Mobihoc 2004)*, Tokyo, Japan, May 2004. ACM.

[43] James Scott, Pan Hui, Jon Crowcroft, and Christophe Diot. Haggle: a Networking Architecture Designed Around Mobile Users. In *IFIP Conference on Wireless on Demand Network Systems and Services (WONS 2006)*, Les Ménuires, France, January 2006.

[44] Nicolas Le Sommer, Pascale Launay, and Yves Mahéo. A Framework for Opportunistic Networking in Spontaneous and Ephemeral Social Networks. In *10th Workshop on Challenged Networks (CHANTS'2015)*, Paris, France, September 2015. ACM.

[45] Frédéric Guidec, Nicolas Le Sommer, and Yves Mahéo. Opportunistic Software Deployment in Disconnected Mobile Ad Hoc Networks. *International Journal of Handheld Computing Research*, 1(1):24–42, 2010.

[46] Ólafur Helgason, Sylvia T Kouyoumdjieva, Ljubica Pajević, Emre A Yavuz, and Gunnar Karlsson. A middleware for opportunistic content distribution. *Computer Networks*, 107:178–193, 2016.

[47] Valerio Arnaboldi, Marco Conti, and Franca Delmastro. CAMEO: A Novel Context-Aware Middleware for Opportunistic Mobile Social Networks. *Pervasive and Mobile Computing*, 2013.

[48] Marco Conti, Franca Delmastro, and Andrea Passarella. Mobile service platforms based on opportunistic computing: The scampi project. *Mobile Computing*, page 25, 2013.

[49] Soma Bandyopadhyay, Munmun Sengupta, Souvik Maiti, and Subhajit Dutta. Role of middleware for internet of things: A study. *International Journal of Computer Science & Engineering Survey (IJCSES)*, 2(3):94–105, 2011.

[50] Mohammad Abdur Razzaque, Marija Milojevic-Jevric, Andrei Palade, and Siobhán Clarke. Middleware for internet of things: a survey. *IEEE Internet of Things Journal*, 3(1):70–95, 2016.

[51] Carlos Albuquerque, Aércio Cavalcanti, Felipe S Ferraz, and Ana Paula Furtado. A Study on Middleware for IoT: A comparison between relevant articles. In *Proceedings on the International Conference on Internet Computing (ICOMP)*, page 32. The Steering Committee of The World Congress in Computer Science, Computer Engineering and Applied Computing (WorldComp), 2016.

[52] Anne HH Ngu, Mario Gutierrez, Vangelis Metsis, Surya Nepal, and Michael Z Sheng. Iot middleware: A survey on issues and enabling technologies. *IEEE Internet of Things Journal*, 2016.

[53] Dominique Guinard, Vlad Trifa, and Erik Wilde. A resource oriented architecture for the web of things. In *Internet of Things (IOT), 2010*, pages 1–8. IEEE, 2010.

[54] Kashif Dar, Amir Taherkordi, Harun Baraki, Frank Eliassen, and Kurt Geihs. A resource oriented integration architecture for the internet of things: A business process perspective. *Pervasive and Mobile Computing*, 20:145–159, 2015.

[55] Thiago Teixeira, Sara Hachem, Valérie Issarny, and Nikolaos Georgantas. Service oriented middleware for the internet of things: a perspective. In *Towards a Service-Based Internet*, pages 220–229. Springer, 2011.

[56] Valérie Issarny, Nikolaos Georgantas, Sara Hachem, Apostolos Zarras, Panos Vassiliadist, Marco Autili, Marco Aurelio Gerosa, and Amira Ben Hamida. Service-oriented middleware for the future internet: state of the art and research directions. *Journal of Internet Services and Applications*, 2(1):23–45, 2011.

[57] Suparna De, Payam Barnaghi, Martin Bauer, and Stefan Meissner. Service modelling for the internet of things. In *Computer Science and Information Systems (FedCSIS), 2011 Federated Conference on*, pages 949–955. IEEE, 2011.

[58] Michael Mrissa, Lionel Médini, and Jean-Paul Jamont. Semantic discovery and invocation of functionalities for the web of things. In *WETICE Conference (WETICE), 2014 IEEE 23rd International*, pages 281–286. IEEE, 2014.

[59] Cesare Pautasso, Olaf Zimmermann, and Frank Leymann. Restful web services vs. big'web services: making the right architectural decision. In *17th international conference on World Wide Web*, pages 805–814. ACM, 2008.

[60] Mark Nottingham and Eran Hammer-Lahav. Defining well-known uniform resource identifiers (uris). IETF Draft, May 2016.

[61] Klaus Hartke. Observing Resources in the Constrained Application Protocol (CoAP). IETF RFC, September 2015.

[62] Bilhanan Silverajan and Teemu Savolainen. CoAP Communication with Alternative Transports. IETF Internet Draft, July 2014.

[63] Nenad Gligoric, Tomislav Dimcic, Dejan Drajic, Srdjan Krco, Igor Dejanovic, Nhon Chu, and Aleksandar Obradovic. CoAP over SMS: Performance evaluation for machine to machine communication. In *20th Telecommunications Forum (TELFOR 2012)*, pages 1–4, Belgrade, Serbia, November 2012. IEEE CS.

[64] Guido Moritz. SOAP-over-CoAP Binding. IETF Internet Draft, June 2011.

[65] Guido Moritz, Frank Golatowski, and Dirk Timmermann. A lightweight soap over coap transport binding for resource constraint networks. In *8th IEEE International Conference on Mobile Adhoc and Sensor Systems (MASS)*, pages 861–866. IEEE CS, 2011.

[66] Koojana Kuladinithi, Olaf Bergmann, Thomas Pötsch, Markus Becker, and Carmelita Görg. Implementation of coap and its application in transport logistics. In *Proceedings of the Workshop on Extending the Internet to Low Power and Lossy Networks (IP+SN 2011)*, Chicago, IL, USA, April 2011.

[67] Jouni Mäenpää, Jaime Jiménez Bolonio, and Salvatore Loreto. Using reload and coap for wide area sensor and actuator networking. *EURASIP Journal on Wireless Communications and Networking*, 2012:1–22, 2012.

[68] Mark Nottingham. Web Linking, October 2010.

[69] Markus Eisenhauer, Peter Rosengren, and Pablo Antolin. Hydra: A development platform for integrating wireless devices and sensors into ambient intelligence systems. In *The Internet of Things*, pages 367–373. Springer, 2010.

[70] Mauro Caporuscio, Pierre-Guillaume Raverdy, and Valerie Issarny. ubisoap: A service-oriented middleware for ubiquitous networking. *IEEE Transactions on Services Computing*, 5(1):86–98, 2012.

[71] Romain Rouvoy, Paolo Barone, Yun Ding, Frank Eliassen, Svein Hallsteinsen, Jorge Lorenzo, Alessandro Mamelli, and Ulrich Scholz. Music: Middleware support for self-adaptation in ubiquitous and service-oriented environments. In *Software engineering for self-adaptive systems*, pages 164–182. Springer, 2009.

[72] Dominique Guinard, Vlad Trifa, Stamatis Karnouskos, Patrik Spiess, and Domnic Savio. Interacting with the soa-based internet of things: Discovery, query, selection, and on-demand provisioning of web services. *IEEE transactions on Services Computing*, 3(3):223–235, 2010.

[73] Iván Corredor, José F Martínez, Miguel S Familiar, and Lourdes López. Knowledge-aware and service-oriented middleware for deploying pervasive services. *Journal of Network and Computer Applications*, 35(2):562–576, 2012.

[74] Edgardo Avilés-López and J Antonio García-Macías. Tinysoa: a service-oriented architecture for wireless sensor networks. *Service Oriented Computing and Applications*, 3(2):99–108, 2009.

[75] Gaetano F Anastasi, Enrico Bini, Antonio Romano, and Giuseppe Lipari. A service-oriented architecture for qos configuration and management of wireless sensor networks. In *Emerging Technologies and Factory Automation (ETFA), 2010 IEEE Conference on*, pages 1–8. IEEE, 2010.

[76] I LogMeIn. Xively, 2015.

[77] Carriots.

[78] Echelon Corporation.

[79] Amira Ben Hamida, Fabio Kon, Gustavo Ansaldi Oliva, Carlos Dos Santos, Jean-Pierre Lorré, Marco Autili, Guglielmo De Angelis, Apostolos Zarras, Nikolaos Georgantas, Valérie Issarny, et al. An integrated development and runtime environment for the future internet. *The Future Internet*, pages 81–92, 2012.

[80] Mauro Caporuscio, Marco Funaro, Carlo Ghezzi, and Valérie Issarny. ubirest: A restful service-oriented middleware for ubiquitous networking. In *Advanced Web Services*, pages 475–500. Springer, 2014.

[81] Mauro Caporuscio, Marco Funaro, and Carlo Ghezzi. RESTful service architectures for pervasive networking environments. In *REST: from Research to Practice*, pages 401–422.

[82] Charith Perera, Prem Prakash Jayaraman, Arkady Zaslavsky, Peter Christen, and Dimitrios Georgakopoulos. Mosden: An internet of things middleware for resource constrained mobile devices. In *47th Hawaii International Conference on System Sciences (HICSS)*, pages 1053–1062. IEEE, 2014.

[83] Karl Aberer and Manfred Hauswirth. Middleware support for the "internet of things". *5th Workshop of Digital Enterprise Research Institute*, 2006.

[84] Chien-Liang Fok, Gruia-Catalin Roman, and Chenyang Lu. Servilla: a flexible service provisioning middleware for heterogeneous sensor networks. *Science of Computer Programming*, 77(6):663–684, 2012.

[85] Eclipse. Kura.

[86] Microej.

[87] AllSeen Alliance. Iotivity, 2015.

[88] Massimo Villari, Antonio Celesti, Maria Fazio, and Antonio Puliafito. Alljoyn lambda: An architecture for the management of smart environments in iot. In *Smart Computing Workshops (SMARTCOMP Workshops), 2014 International Conference on*, pages 9–14. IEEE, 2014.

[89] Yan Rodriguez and Ben Garcia. Programmable multi-function z-wave adapter for z-wave wireless networks, 2012. US Patent 8,117,362.

[90] ASHRAE Standard. Standard 135-2005, bacnet-a data communication protocol for building automation and control networks. *American Society of Heating, Refrigerating and Air-Conditioning Engineers, Atlanta, GA*, 2005.

[91] Linyi Tian. Lightweight m2m (oma lwm2m). *OMA device management working group (OMA DM WG), Open Mobile Alliance (OMA)*, 2012.

[92] Antonio Puliafito, Angelo Cucinotta, Antonino Longo Minnolo, and Angelo Zaia. Making the internet of things a reality: The wherex solution. In *The Internet of Things*, pages 99–108. Springer, 2010.

[93] Michal Nagy, Artem Katasonov, Michal Szydlowski, Oleksiy Khriyenko, Sergiy Nikitin, and Vagan Terziyan. *Challenges of middleware for the internet of things*. INTECH Open Access Publisher, 2009.

[94] Marco Conti, Silvia Giordano, Martin May, and Andrea Passarella. From opportunistic networks to opportunistic computing. *IEEE Communications Magazine*, 48(9):126–139, September 2010.

[95] Marco Conti and Mohan Kumar. Opportunities in Opportunistic Computing. *Computer*, 43:42–50, 2010.

[96] Jörg Ott and Dirk Kutscher. Applying dtn to mobile internet access: An experiment with HTTP. *Universität Bremen," TRTZI-050701*, 2005.

[97] Philo Juang, Hidekazu Oki, Yong Wang, Margaret Martonosi, Li Shiuan Peh, and Daniel Rubenstein. Energy-efficient Computing for Wildlife Tracking: Design Tradeoffs and Early Experiences with ZebraNet. In *Proceedings of the 10th International Conference on Architectural Support for Programming Languages and Operating Systems (APLOS X)*, pages 96–107, San Jose, CA, USA, 2002. ACM.

[98] Tara Small and Zygmunt J Haas. The shared wireless infostation model: a new ad hoc networking paradigm (or where there is a whale, there is a way). In *Proceedings of the 4th ACM international symposium on Mobile ad hoc networking & computing*, pages 233–244. ACM, 2003.

[99] Kamini Kamini and Rakesh Kumar. Vanet parameters and applications: A review. *Global Journal of Computer Science and Technology*, 10(7), 2010.

[100] Aruna Balasubramanian, Yun Zhou, W Bruce Croft, Brian Neil Levine, and Aruna Venkataramani. Web search from a bus. In *Proceedings of the second ACM workshop on Challenged networks*, pages 59–66. ACM, 2007.

[101] N4c project.

[102] Anders Lindgren. Social networking in a disconnected network: fbDTN: facebook over DTN. In *Proceedings of the 6th ACM workshop on Challenged networks*, pages 69–70. ACM, 2011.

[103] Mikko Pitkänen, Teemu Kärkkäinen, and Jörg Ott. Opportunistic web access via wlan hotspots. In *Pervasive Computing and Communications (PerCom), 2010 IEEE International Conference on*, pages 20–30. IEEE, 2010.

[104] Yves Mahéo, Nicolas Le Sommer, Pascale Launay, Frédéric Guidec, and Mario Dragone. Beyond Opportunistic Networking Protocols: a Disruption-Tolerant Application Suite for Disconnected MANETs. In *4th Extreme Conference on Communication (ExtremeCom'12)*, pages 1–6, Zürich, Switzerland, March 2012. ACM.

[105] Jörg Ott and Dirk Kutscher. Bundling the Web: HTTP over DTN. In *Workshop on Networking in Public transport (WNEPT 2006)*, Waterloo, Ontario, Canada, August 2006.

[106] Lloyd Wood, Peter Holliday, Daniel Floreani, and Ioannis Psaras. Moving data in DTNs with HTTP and MIME. In *ICUMT*, pages 1–4, 2009.

[107] Andrea Passarella, Mohan Kumar, Marco Conti, and Eleonora Borgia. Minimum-Delay Service Provisioning in Opportunistic Networks. *IEEE Transactions on Parallel and Distributed Systems*, 22(8):1267–1275, 2010.

[108] Abdulkader Benchi, Pascale Launay, and Frédéric Guidec. A P2P Tuple Space Implementation for Disconnected MANETs. *Peer-to-Peer Networking and Applications*, 8(1):1–16, January 2015.

[109] Gian Pietro Picco, Amy L. Murphy, and Gruia-Catalin Roman. LIME: Linda Meets Mobility. In *Proceedings of International Conference on Software Engineering (ICSE 99)*, pages 368–377, 1999.

[110] P. Pantazopoulos, I. Stavrakakis, A. Passarella, and M. Conti. Efficient social-aware content placement in opportunistic networks. In *Wireless On-demand Network Systems and Services (WONS), 2010 Seventh International Conference on*, pages 17–24, February 2010.

[111] Chiara Boldrini, Marco Conti, and Andrea Passarella. Contentplace: Social-aware data dissemination in opportunistic networks. In *Proceedings of the 11th International Symposium on Modeling, Analysis and Simulation of Wireless and Mobile Systems*, MSWiM '08, pages 203–210, New York, NY, USA, 2008. ACM.

[112] Frédéric Guidec, Pascale Launay, Nicolas Le Sommer, and Yves Mahéo. Experimentation with a DoDWAN-based Application Suite for Opportunistic Computing (demo proposal). In *4th Extreme Conference on Communication (ExtremeCom'12)*, pages 1–2, Zürich, Switzerland, March 2012. ACM.

[113] Yves Mahéo and Romeo Said. Service Invocation over Content-Based Communication in Disconnected Mobile Ad Hoc Networks. In *24th International Conference on Advanced Information Networking and Applications (AINA'10)*, pages 503–510, Perth, Australia, April 2010. IEEE CS.

[114] Fadhlallah Baklouti, Nicolas Le Sommer, and Yves Mahéo. Opportunistic Service Composition in Pervasive Networks. In *IFIP Wireless Days Conference*, Porto, Portugal, March 2017. IEEE.

[115] Michaël Mrissa, Lionel Médini, Jean-Paul Jamont, Nicolas Le Sommer, and Jérôme Laplace. An Avatar Architecture for the Web of Things. *IEEE Internet Computing*, 19(2):30–38, March 2015.

[116] Nicolas Le Sommer, Lionel Touseau, Yves Mahéo, Maël Auzias, and Frédéric Raimbault. A Disruption-Tolerant RESTful Support for the Web of Things. In *International Conference on Future Internet of Things and Cloud (icfiotc), Proceedings of the 4th*, pages 17–24, Vienna, Austria, August 2016. IEEE.

[117] Stephen Farrell, Vinny Cahill, Dermot Geraghty, Ivor Humphreys, and Paul McDonald. When TCP breaks: Delay-and disruption-tolerant networking. *IEEE Internet Computing*, 10(4):72–78, 2006.

[118] Carsten Bormann and Zach Shelby. Block-Wise Transfers in the Constrained Application Protocol (CoAP), August 2016.

[119] Matthias Kovatsch, Martin Lanter, and Zach Shelby. Californium: Scalable Cloud Services for the Internet of Things through CoAP. In *Proceedings of the International Conference on the Internet of Things (IoT 2014)*, Cambridge, MA, USA, October 2014. ACM Press.

[120] Markus Becker, Kepeng Li, Koojana Kuladinithi, and Thomas Poetsch. Transport of CoAP over SMS. IETF Internet Draft, August 2014.

[121] Peter Van Der Stock and Anuj Sehgal. Patch Method for Constrained Application Protocol (CoAP). IETF ID, March 2015.

[122] Carlos Borrego, Sergi Robles, Angela Fabregues, and Adrián Sánchez-Carmona. A mobile code bundle extension for application-defined routing in delay and disruption tolerant networking. volume 87, pages 59–77. Elsevier, 2015.

[123] Haifeng Li, Huachun Zhou, Hongke Zhang, Bohao Feng, and Wenfeng Shi. EmuStack: An OpenStack-Based DTN Network Emulation Platform. *Mobile Information Systems*, 2016(6540207):15, October 2016.

[124] Jeff Ahrenholz. Comparison of core network emulation platforms. In *IEEE Military Communications Conference (MILCOM)*, pages 864–869, 2010.

[125] Injong Rhee, Minsu Shin, Seongik Hong, Kyunghan Lee, Seong Joon Kim, and Song Chong. On the levy-walk nature of human mobility. *IEEE/ACM transactions on networking (TON)*, 19(3):630–643, 2011.

[126] Mongnam Han, Youngseok Lee, Sue B. Moon, Keon Jang, and Dooyoung Lee. CRAWDAD dataset kaist/wibro (v. 2008-06-04). Downloaded from http://crawdad.org/kaist/wibro/20080604, June 2008.

[127] Yu-Kang Lee, Chun-Tuan Chang, You Lin, and Zhao-Hong Cheng. The dark side of smartphone usage: Psychological traits, compulsive behavior and technostress. *Computers in Human Behavior*, 31:373–383, 2014.

[128] Stuart Kurkowski, Tracy Camp, and Michael Colagrosso. MANET Simulation Studies: the Incredibles. *ACM SIGMOBILE Mobile Computing and Communications Review*, 9:50–61, 2005.

[129] James Scott, Richard Gass, Jon Crowcroft, Pan Hui, Christophe Diot, and Augustin Chaintreau. Crawdad dataset cambridge/haggle (v. 2006-09-15). *CRAWDAD wireless network data archive*, 2006.

[130] José M Hernández-Muñoz, Jesús Bernat Vercher, Luis Muñoz, José A Galache, Mirko Presser, Luis A Hernández Gómez, and Jan Pettersson. Smart cities at the forefront of the future internet. In *The Future Internet Assembly*, pages 447–462. Springer, 2011.

[131] Sara Hachem, Thiago Teixeira, and Valérie Issarny. Ontologies for the internet of things. In *Proceedings of the 8th Middleware Doctoral Symposium*, page 3. ACM, 2011.

[132] Robert G Raskin and Michael J Pan. Knowledge representation in the semantic web for earth and environmental terminology (sweet). *Computers & geosciences*, 31(9):1119–1125, 2005.

[133] Zhexuan Song, Alvaro A Cárdenas, and Ryusuke Masuoka. Semantic middleware for the internet of things. *IEEE Internet of Things 2010*, pages 1–8, 2010.

[134] Amarnath Palavalli, Durgaprasad Karri, and Swarnalatha Pasupuleti. Semantic internet of things. In *Semantic Computing (ICSC), 2016 IEEE Tenth International Conference on*, pages 91–95. IEEE, 2016.

[135] Nicolas Le Sommer and Yves Mahéo. Location-Aware Routing for Service-Oriented Opportunistic Computing. *International Journal on Advances in Networks and Services*, (3):225–235, 2012.

[136] Ólafur R Helgason, Emre A Yavuz, Sylvia T Kouyoumdjieva, Ljubica Pajevic, and Gunnar Karlsson. A mobile peer-to-peer system for opportunistic content-centric networking. In *Proceedings of the second ACM SIGCOMM workshop on Networking, systems, and applications on mobile handhelds*. ACM, 2010.

[137] Mauro Caporuscio, Antonio Carzaniga, and Alexander L Wolf. Design and evaluation of a support service for mobile, wireless publish/subscribe applications. *Software Engineering, Technical report IEEE Transactions on*, 29(12):1059–1071, 2003.

[138] Michael Koster, Ari Keranen, and Jaime Jimenez. Publish-Subscribe Broker for the Constrained Application Protocol (CoAP), March 2015.

[139] Lloyd Wood, Peter Holliday, Daniel Floreani, and Wesley M. Eddy. Sharing the Dream: the Consensual Hallucination Offered by the Bundle Protocol. In *Internation Congress on Ultra Modern Telecommunication (ICUMT'09)*, pages 1–2. IEEE, 2009.

[140] Lloyd Wood, Wesley M. Eddy, and Peter Holliday. A bundle of problems. In *Aerospace conference, 2009 IEEE*, pages 1–17. IEEE, 2009.

[141] Andrew Jenkins, Sebastian Kuzminsky, Kevin K Gifford, Robert L Pitts, and Kelvin Nichols. Delay/disruption-tolerant networking: flight test results from the international space station. In *Aerospace Conference, 2010 IEEE*, pages 1–8. IEEE, 2010.

[142] Romeo Said. *Middleware for Service Provision in Disconnected Mobile Ad Hoc Networks*. PhD thesis, Université de Bretagne-Sud / Université Européenne de Bretagne, February 2011.

[143] Jonah P Tower and Thomas DC Little. A Proposed Scheme for Epidemic Routing With Active Curing for Opportunistic Networks. In *2008 22nd International Workshops on Advanced Information Networking and Applications (AINA Workshops)*, pages 1696–1701, Gino-wan, Okinawa, Japan, 2008. IEEE.

[144] Khaled A. Harras, Kevin C. Almeroth, and Elisabeth M. Belding-Royer. Delay Tolerant Mobile Networks (DTMNs): Controlled Flooding in Sparse Mobile Networks. In *IFIP Networking Conference, Waterloo, Ontario, CANADA*, May 2005.

[145] Ling jyh Chen, Chen hung Yu, Cheng long Tseng, Hao hua Chu, and Cheng fu Chou. A Content-Centric Framework for Effective Data Dissemination in Opportunistic Networks. *IEEE Journal of Selected Areas in Communications*, 2008.

[146] Arnaud Casteigts, Paola Flocchini, Walter Quattrociocchi, and Nicola Santoro. Time-Varying Graphs and Dynamic Networks. *International Journal of Parallel, Emergent and Distributed Systems (IJPEDS)*, 27(5):387–408, 2012.

[147] Petter Holme and Jari Saramäki. Temporal networks. *Physics reports*, 519(3):97–125, 2012.

[148] Brad Williams and Tracy Camp. Comparison of broadcasting techniques for mobile ad hoc networks. In *Proceedings of the 3rd ACM international symposium on Mobile ad hoc networking & computing (MobiHoc'02)*, pages 194–205, Lausanne, Switzerland, 2002. ACM.

[149] Vincent Lenders, Martin May, and Bernhard Plattner. Service Discovery in Mobile Ad Hoc Networks: A Field Theoretic Approach. In *Proceedings of the IEEE 6th International Symposium on a World of Wireless, Mobile, and Multimedia Networks (WoWMoM 2005)*, pages 120–130, Taormina, Italy, June 2005. IEEE press.

[150] Zhenguo Gao, Ling Wang, Mei Yang, and Xiaozong Yang. CNPGSDP: An efficient group-based service discovery protocol for MANETs. *Computer Networks*, 50(16):3165–3182, November 2006.

[151] Christian Glacet, Marco Fiore, and Marco Gramaglia. Temporal connectivity of vehicular networks: The power of store-carry-and-forward. In *IEEE Vehicular Networking Conference (VNC 2015)*, pages 52–59, Kyoto, Japan, December 2015. IEEE.

[152] Einar W Vollset and Paul D Ezhilchelvan. Design and Performance-Study of Crash-Tolerant Protocols for Broadcasting and Reaching Consensus in MANETs. In *Proceedings of the 24th IEEE Symposium on Reliable Distributed Systems*, volume 0, pages 166–178, Washington, DC, USA, 2005. IEEE Computer Society.

[153] Qun Li and Daniela Rus. Sending Messages to Mobile Users in Disconnected Ad-hoc Wireless Networks. In *Proceedings of the Sixth Annual International Conference on Mobile Computing and Networking*, pages 44–55, Boston, August 2000. ACM Press.

[154] Gabriel Sandulescu and Simin Nadjm-Tehrani. Opportunistic dtn routing with window-aware adaptive replication. In *Proceedings of the 4th Asian Conference on Internet Engineering*, pages 103–112. ACM, November 2008.

[155] Khaled A. Harras and Kevin C. Almeroth. Transport Layer Issues in Delay Tolerant Mobile Networks. In *Proceedings of IFIP Networking*, Coimbra, Portugal, May 2006.

[156] Ritesh Shah and Norman C. Hutchinson. Delivering Messages in Disconnected Mobile Ad-Hoc Networks. In *Proceedings of ADHOC-NOW 2003*, Montreal, October 2003.

[157] Ling-Jyh Chen, Chen-Hung Yu, Tony Sun, Yung-Chih Chen, and Hao hua Chu. A Hybrid Routing Approach for Opportunistic Networks. 2006.

[158] Ilias Leontiadis, Paolo Costa, and Cecilia Mascolo. Persistent Content-based Information Dissemination in Hybrid Vehicular Networks. In *Proc. of the 7th IEEE International Conference on Pervasive Computing and Communications (Percom09)*, Galveston, TX, USA, March 2009.

[159] Sébastien Baehni, Chirdeep Singh Chhabra, and Rachid Guerraoui. Selective Information Dissemination in a Mobile Environment. In *Proceedings of the 6th Middleware Conference*, December 2005. Extended version.

[160] Mikko Pitkänen, Teemu Kaärkkäinen, and Jörg Ott. Mobility and Service Discovery in Opportunistic Networks. In *International Workshop on the Impact of Human Mobility in Pervasive Systems and Applications (PerMoby 2012)*, pages 204–210, Lugano, Switzerland, March 2012. IEEE CS.

[161] Balazs E Pataki and Levente Kovacs. Sensor data collection experiments with chaoster in the fed4fire federated testbeds. In *Wireless and Mobile Computing, Networking and Communications (WiMob), 2014 IEEE 10th International Conference on*, pages 306–313. IEEE, 2014.

[162] Hu Zhou and Suresh Singh. Content based multicast (CBM) in ad hoc networks. In *Proceedings of the 1st ACM International Symposium on Mobile Ad Hoc Networking and Computing (Mobihoc'00)*, pages 51–60, Boston, Massachusetts, 2000. IEEE Press.

[163] Afonso Ferreira, Alfredo Goldman, and Julian Monteiro. Performance Evaluation of Routing Protocols for MANETs with Known Connectivity Patterns using Evolving Graphs. *Wireless Networks*, 16(3):627–640, 2010.

[164] John Whitbeck, Yoann Lopez, Jérémie Leguay, Vania Conan, and Marcelo Dias de Amorim. Push-and-track: Saving infrastructure bandwidth through opportunistic forwarding. *Pervasive and Mobile Computing*, 8(5):682–697, 2012.

[165] Andronikos Nedos, Kulpreet Singh, and Siobhan Clarke. Service*: Distributed Service Advertisement for Multi-Service, Multi-Hop MANET Environments. In *Proceedings of 7th IFIP International Conference on Mobile and Wireless Communication Networks (MWCN'05)*, Marrakech, Morocco, September 2005.

[166] Dipanjan Chakraborty, Anupam Joshi, Yelena Yesha, and Tim Finin. Toward distributed service discovery in pervasive computing environments. *IEEE Transactions on Mobile Computing*, 5(2):97– 112, February 2006.

[167] D.E. Cooper, P. Ezhilchelvan, and I. Mitrani. Encounter-based message propagation in mobile ad-hoc networks. *Ad Hoc Networks*, 7(7):1271–1284, 2009.

[168] Françoise Sailhan and Valérie Issarny. Scalable Service Discovery for MANET. In *Proceedings of the 3rd IEEE International Conference on Pervasive Computing and Communications (PerCom'2005)*, Hawaii, USA, March 2005. IEEE Press.

[169] Nedos Andronikos, Kulpreet Singh, Raymond Cunningham, and Siobhan Clarke. Probabilistic discovery of semantically diverse content in manets. *IEEE Transactions on Mobile Computing*, 2009.

[170] Marios Logothetis, Kostas Tsagkaris, and Panagiotis Demestichas. Application and mobility aware integration of opportunistic networks with wireless infrastructures. *Computers & Electrical Engineering*, 39(6):1609–1624, August 2013. Special Issue on Wireless Systems: Modeling, Monitoring, Transmission, Performance Evaluation and Optimization.

[171] Dipanjan Chakraborty, Anupam Joshi, and Yelena Yesha. Integrating service discovery with routing and session management for ad-hoc networks. *Ad Hoc Networks*, 4(2):204–224, 2006.

[172] Marco Conti, Emanuel Marzini, Davide Mascitti, Andrea Passarella, and Laura Ricci. Service Selection and Composition in Opportunistic Networks. In *9th International Wireless Communications and Mobile Computing Conference (IWCMC 2013)*, pages 1565–1572, Cagliari, Italy, July 2013.

[173] Chiara Boldrini, Marco Conti, and Andrea Passarella. Autonomic Behaviour of Opportunistic Network Routing. *Inderscience International Journal of Autonomous and Adaptive Communications Systems*, 1(1):122–147, 2008.

[174] Chiara Boldrini, Marco Conti, Andrea Passarella, et al. Users mobility models for opportunistic networks: the role of physical locations. *Proc. of IEEE WRECOM*, page 23, 2007.

[175] Isam Ishaq, Jeroen Hoebeke, Ingrid Moerman, and Piet Demeester. Observing CoAP groups efficiently. *Ad Hoc Networks*, 37, Part 2:368–388, February 2016.

[176] Keith Scott, Tamer Refaei, Nirav Trivedi, Jenny Trinh, and Joseph P Macker. Robust communications for disconnected, intermittent, low-bandwidth (dil) environments. In *2011-MILCOM 2011 Military Communications Conference*, pages 1009–1014. IEEE, 2011.

[177] Gunnar Karlsson, Vincent Lenders, and Martin May. Delay-Tolerant Broadcasting. pages 369–381, February 2007.

[178] Vincent Lenders, Gunnar Karlsson, and Martin May. Wireless Ad Hoc Podcasting. In *SECON '07. 4th Annual IEEE Communications Society Conference on Sensor, Mesh and Ad Hoc Communications and Networks*, pages 273–283. IEEE CNF, June 2007.

[179] Yongtao Wei, Jinkuan Wang, Junwei Wang, and Zhenqi Wang. A double-slot based delay/disruption tolerant routing algorithm for iot. *IEEE 14th International Conference on Communication Technology*, pages 1027–1031, 2012.

[180] Marco Bonola, Lorenzo Bracciale, Pierpaolo Loreti, Raul Amici, Antonello Rabuffi, and Giuseppe Bianchi. Opportunistic communication in smart city: Experimental insight with small-scale taxi fleets as data carriers. *Ad Hoc Networks*, 43:43–55, June 2016.

[181] Thrasyvoulos Spyropoulos, Thierry Turletti, and Katia Obraczka. Routing in Delay-Tolerant Networks Comprising Heterogeneous Node Populations. *IEEE Transactions on Mobile Computing*, 8(8):1132–1147, August 2009.

[182] Thrasyvoulos Spyropoulos, Konstantinos Psounis, and Cauligi S Raghavendra. Performance analysis of mobility-assisted routing. In *Proceedings of the 7th ACM international symposium on Mobile ad hoc networking and computing*, pages 49–60. ACM, 2006.

[183] Rubén Martínez-Vidal. *Architectures for Aeronautical Opportunistic Networking*. PhD thesis, Universitat Autónoma de Barcelona, November 2015.

[184] Uday Mohan, Kevin C. Almeroth, and Elizabeth M. Belding-Royer. Scalable Service Discovery in Mobile Ad hoc Networks. In *Proceedings of 3rd International IFIP-TC6 Networking Conference*, volume 3042 of *LNCS*, Athens, Greece, May 2004. Springer.

[185] Paal E. Engelstad, Yan Zheng, Rajeev Koodli, and Charles E. Perkins. Service Discovery Architectures for On-Demand Ad Hoc Networks. *Ad Hoc and Sensor Wireless Networks*, 1, 2006.

[186] O.V. Drugan, T. Plagemann, and E. Munthe-Kaas. Building resource aware middleware services over manet for rescue and emergency applications. In *IEEE 16th International Symposium on Personal, Indoor and Mobile Radio Communications (PIMRC 2005)*, volume 2, pages 816–820, Germany, sep 2005. IEEE Press.

[187] Jerry Tyan and H. Mahmoud, Qusay. A network layer based architecture for service discovery in mobile ad hoc networks. In *17th Ann. IEEE Canadian Conf. Electrical and Computer Eng. (CCECE 04)*, volume 3, pages 1379– 1384, Niagara Falls, Canada, May 2004. IEEE Press.

[188] Roberto Baldoni, Roberto Beraldi, Leonardo Querzoni, Gianpaolo Cugola, and Matteo Migliavacca. Content-Based Routing in Highly Dynamic Mobile Ad Hoc Networks. *Journal of Pervasive Computing and Communication*, 1(4):277–288, dec 2005.

[189] Luc Hogie, Pascal Bouvry, Frédéric Guinand, Grégoire Danoy, and Enrique Alba. A Bandwidth-Efficient Broadcasting Protocol for Mobile Multi-hop Ad hoc Networks. In *Proceedings of the International Conference on Networking*, Mauritius, April 2006. IEEE CS.

[190] Joseph Kopena, Evan Sultanik, Gaurav Naik, Iris Howley, Maxim Peysakhov, Vincent Cicirello, Moshe Kam, and William Regli. Service-Based Computing on Manets: Enabling Dynamic Interoperability of First Responders. *IEEE Intelligent Systems*, 20(5):17–25, 2005.

[191] Evan A Sultanik, Maxim D Peysakhov, and William C Regli. Agent transport simulation for dynamic peer-to-peer networks. In *International Workshop on Multi-Agent Systems and Agent-Based Simulation*, pages 162–173. Springer, 2005.

[192] Amin Vahdat and David Becker. Epidemic routing for partially connected ad hoc networks, 2000.

[193] U. C. Kozat and L. Tassiulas. Network Layer Support for Service Discovery in Mobile Ad Hoc Networks. In *Proceedings of IEEE/INFOCOM-2003*, April 2003.

[194] R.S.D. Wahidabanu and G. Fathima. A New Queuing Policy for Delay Tolerant Networks. *International Journal of Computer Applications*, 1(20):56–60, February 2010.

[195] Milenko Petrovic, Vinod Muthusamy, and Hans-Arno Jacobsen. Content-Based Routing in Mobile Ad Hoc Networks. In *Proc. of the 2nd Annual International Conference on Mobile and Ubiquitous Systems: Networking and Services (MobiQuitous'05)*, San Diego, California, USA, July 2005. IEEE Press.

[196] Jorg Ott, Dirk Kutscher, and Christoph Dwertmann. Integrating DTN and MANET Routing. In *Proceedings of the 2006 SIGCOMM workshop on Challenged networks*, pages 221–228, 2006.

[197] Andronikos Nedos, Kulpreet Singh, Raymond Cunningham, and Siobhan Clarke. A Gossip Protocol to Support Service Discovery with Heterogeneous Ontologies in MANETs. In *Proceeedings of the 3rd IEEE International Conference on Wireless and Mobile Computing, Networking and Communications (WiMOB 2007)*, pages 53–53, White Plains, New York, USA, October 2007. IEEE Press.

[198] Chalermek Intanagonwiwat, Ramesh Govindan, Deborah Estrin, John Heidemann, and Fabio Silva. Directed diffusion for wireless sensor networking. *Networking, IEEE/ACM Transactions on*, 11(1):2–16, 2003.

[199] Zijian Wang, Eyuphan Bulut, and Boleslaw K. Szymanski. Service Discovery for Delay Tolerant Networks. In *Proceedings of the 5th Workshop on Heterogeneous, Multi-Hop, Wireless and Mobile Networks (HeterWMN 2010), in conjunction with Globecom 2010*, pages 136–141, Miami, FL, USA, December 2010. IEEE CS.

[200] Paolo Costa and Gian Pietro Picco. Semi-Probabilistic Content-Based Publish-Subscribe. In *25th International Conference on Distributed Computing Systems (ICDCS 2005)*, pages 575–585, Columbus, Ohio, USA, jun 2005. IEEE Computer Society.

[201] Mirco Musolesi and Cecilia Mascolo. CAR: Context-Aware Adaptive Routing for Delay Tolerant Mobile Networks. *IEEE Transactions on Mobile Computing*, 8(2):246–260, 2009.

[202] Guiseppe Sollazzo, Mirco Musolesi, and Cecilia Mascolo. TACO-DTN: A Time-Aware COntent-based dissemination system for Delay Tolerant Networks. In *Proceedings of the 1st international MobiSys workshop on Mobile opportunistic networking (MobiOpp '07)*, pages 83–90, New York, NY, USA, jun 2007. ACM Press.

[203] Alfredo Goldman, Paulo Floriano, and Afonso Ferreira. A Tool for Obtaining Information on DTN Traces. In *4th Extreme Conference on Communication (ExtremeCom'12)*, Zürich, Switzerland, March 2012. ACM.

[204] Ilias Leontiadis and Cecilia Mascolo. Opportunistic Spatio-Temporal Dissemination System for Vehicular Networks. jun 2007.

[205] André Marques Poersch, Daniel F. Macedo, and José Marcos S. Nogueira. Resource Location for Opportunistic Networks. In *Proceedings of the 5th International Conference on New Technologies, Mobility and Security (NTMS 2012)*, pages 1–5, Istanbul, Turkey, May 2012. IEEE CS.

[206] Samuel C Nelson, Albert F Harris III, and Robin Kravets. Event-driven, role-based mobility in disaster recovery networks. In *Proceedings of the second ACM workshop on Challenged networks*, pages 27–34. ACM, 2007.

[207] Marci Nagy, Teemu Kärkkäinen, and Jörg Ott. Enhancing Opportunistic Networks with Legacy Nodes. In *Proceedings of the 9th Workshop on Challenged Networks (CHANTS 2014)*, pages 1–6, Maui, Hawaii, USA, September 2014. ACM.

[208] Mikko Pitkänen, Teemu Kärkkäinen, and Jörg Ott. Opportunistic Web Access via WLAN Hotspots. In *Proceedings of the 8th International Conference on Pervasive Computing and Communications (PerCom 2010)*, pages 20–30. IEEE CS, March 2010.

[209] Christoph P. Mayer and Oliver P. Waldhorst. Routing in hybrid Delay Tolerant Networks. *Computer Communications*, pages 44–55, July 2014.

[210] Li Li and Louise Lamont. A Lightweight Service Discovery Mechanism for Mobile Ad hoc Pervasive Environment using Cross-Layer Design. In *Proc. of the 2nd Mobile Peer-to-Peer Computing Workshop (MP2P, PerCom'05)*, Hawaii, USA, March 2005.

[211] Sha Hua, Yang Guo, Yong Liu, Hang Liu, and Shivendra S Panwar. Scalable video multicast in hybrid 3g/ad-hoc networks. *Multimedia, IEEE Transactions on*, pages 402–413, April 2011.

[212] Alejandro Tovar, Travis Friesen, Ken. Ferens, and Bob McLeod. A DTN Wireless Sensor Network for Wildlife Habitat Monitoring. In *Proceedings of the 23rd Canadian Conference on Electrical and Computer Engineering (CCECE 2010)*, pages 1–5, May 2010.

[213] Sandor Dornbush and Anupam Joshi. StreetSmart Traffic: Discovering and Disseminating Automobile Congestion Using VANET's. In *65th Semi-Annual Vehicular Technology Conference (VTC2007 07-Spring)*, pages 11–15, Dublin, Ireland, April 2007. IEEE CS.

[214] Khaled Alekeish and Paul Ezhilchelvan. Consensus in Sparse, Mobile Ad Hoc Networks. *IEEE Transactions on Parallel and Distributed Systems*, 23(3):467–474, 2012.

[215] Fatemeh Borran, Ravi Prakash, and André Schiper. Extending Paxos/LastVoting with an Adequate Communication Layer for Wireless Ad Hoc Networks. In *2008 Symposium on Reliable Distributed Systems*, pages 227–236. IEEE, 2008.

[216] Thabotharan Kathiravelu, Thabotharan Kathiravelu, and Arnold Pears. What & When?: Distributing Content in Opportunistic Networks. In Arnold Pears,

editor, *Proc. International Conference on Wireless and Mobile Communications (ICWMC '06)*, pages 64–64, 2006.

[217] Yili Gong, Yongqiang Xiong, Qian Zhang, Zhensheng Zhang, Wenjie Wang, and Zhiwei Xu. Anycast Routing in Delay Tolerant Networks. In *Proceedings of the Global Telecommunications Conference (GLOBECOMM'06)*, San Francisco, CA, USA, November 2006.

[218] Mirco Musolesi, Cecilia Mascolo, and Stephen Hailes. EMMA: Epidemic Messaging Middleware for Ad Hoc Networks. volume 10, pages 28–36, August 2005.

[219] Mirco Musolesi, Stephen Hailes, and Cecilia Mascolo. An ad hoc mobility model founded on social network theory. In *Proceedings of the 7th ACM international symposium on Modeling, analysis and simulation of wireless and mobile systems*, pages 20–24. ACM, 2004.

[220] Pan Hui, Jeremie Leguay, Jon Crowcroft, James Scott, Timur Friedman, and Vania Conan. Osmosis in Pocket Switched Networks. In *First International Conference on Communications and Networking in China*, pages 1–6, Beijing, China, oct 2006. ACM.

[221] ólafur Helgason, Sylvia T. Kouyoumdjieva, Ljubica Pajević, Emre A. Yavuz, and Gunnar Karlsson. A Middleware for Opportunistic Content Distribution. *Computer Networks*, pages –, 2016.

[222] Sacha Trifunovic, Bernhard Distl, Dominik Schatzmann, and Franck Legendre. WiFi-Opp: Ad-hoc-less Opportunistic Networking. In *Proceedings of the 6th ACM workshop on Challenged networks (CHANTS'11)*, pages 37–42, Las Vegas, Nevada, USA, September 2011. ACM.

[223] Elena Pagani and Gian Paolo Rossi. Interest-driven forwarding for delay-tolerant mobile ad hoc networks. In *Proceedings of the 9th International Wireless Communications and Mobile Computing Conference (IWCMC 2013)*, pages 718–723. IEEE CS, 2013.

[224] Pan Hui, Anders Lindgren, and Jon Crowcroft. Empirical evaluation of hybrid opportunistic networks. In *1st International Conference on Communication Systems and Networks (COMSNETS 2009)*, pages 1–10, Bangalore, India, January 2009. IEEE CS.

[225] Yongping Xiong, Limin Sun, Wenbo He, and Jian Ma. Anycast Routing in Mobile Opportunistic Networks. In *IEEE Symposium on Computers and Communications (ISCC'10)*, pages 599–604, Riccione, Italy, June 2010. IEEE CS.

[226] Johannes Morgenroth, Tobias Pögel, and Lars Wolf. Live-streaming in delay tolerant networks. In *Proceedings of the 6th ACM workshop on Challenged networks*, pages 67–68. ACM, September 2011.

[227] Chiara Boldrini, Marco Conti, and Andrea Passarella. Context and Resource Awareness in Opportunistic Network Data Dissemination. In *The Second IEEE WoWMoM Workshop on Autonomic and Opportunistic Communications (AOC 2008)*, Newport Beach, CA, USA, June 2008.

[228] Jie Wu and Fei Dai. Broadcasting in Ad Hoc Networks Based on Self-Pruning. In *Proceedings of the Joint Conference of the IEEE Computer and Communications Societies (INFOCOMM 03)*, volume 3, pages 2240–2250, San Francisco, California, USA, 2003. IEEE CS.

[229] Yu Yang, Hossam Hassanein, and Afzal Mawji. A New Approach to Service Discovery in Wireless Mobile Ad Hoc Networks. In *Proceedings of the IEEE International Conference Communication (ICC 2006)*, Istanbul, Turkey, June 2006. IEEE Press.

Abstract

The Internet of Things (IoT) is usually presented as a set of *things* interconnected through a network that is, in practice, Internet. However, there exist many contexts in which the connectivity is intermittent due to short-range wireless communication means or energy constraints. The Delay Tolerant Networking (DTN) architecture and the Bundle Protocol (BP) are known to overcome this communication challenge as they provide communication means by relying on a store-carry-and-forward mechanism.

This thesis aims to provide programming supports adapted to both IoT and DTN contexts. For this, both DTN and IoT (DT-IoT) challenges are studied and several design principles are proposed. These principles aim to optimize reactivity and efficiency of applications targeting the DT-IoT context.

The first contribution is the definition of a resource-oriented programming support, named BoaP, to enable a DT-IoT. It provides a protocol based on request/response thanks to a transposition of CoAP (Contrained Application Protocol). This transposition consists of fundamental adjustments and enhancements to use BP as the underlying transport protocol.

BoaP has been implemented and tested in a small physical network. A method to evaluate middleware systems in DTNs is presented. A tool implementing this method has been developed. It relies on a virtualization platform that simulates network contacts and emulates network devices. This tool was used to run experimentations that assessed the validity of BoaP.

Finally, another programming support is investigated. It follows a service-oriented approach and respects REST (Representational State Transfer) constraints. It is built on top of BoaP with IoT in mind and is adapted to DTN environments. Its discovery/advertisement exploits a publish/subscribe interface. Service descriptors contain specific fields to inform on the availability of the service providers.

Résumé

L'internet des Objets (IoT) est habituellement présenté comme l'ensemble d'objets interconnectés à travers un réseau qui est, en pratique, Internet. Or, il existe beaucoup de cas où la connectivité est intermittente à cause des interfaces radio courte-portées et des contraintes d'économie d'énergie. L'architecture de réseautage tolérant les délais (DTN) ainsi que le *Bundle Protocole* (BP) sont considérés comme des solutions viables pour résoudre ce genre de challenges grâce au mécanisme *store-carry-and-forward.*

Cette thèse vise à fournir des supports de programmation adaptés autant à l'IoT qu'au contexte DTN. Dans ce but, les challenges relevant du DTN et de l'IoT(DT-IoT) sont étudiés et quelques principes de design logiciels sont proposés. Ces principes ont pour but d'optimiser la réactivité et l'efficacité des applications ayant pour cible un contexte DT-IoT.

La première contribution est la définition d'un support de programmation orienté ressources, nommé BoaP. Ce support fournit un protocole de requête/réponse grâce à une transposition de CoAP (Contrained Application Protocol). Cette transposition est composée d'ajustements fondamentaux et d'améliorations pour utiliser BP en tant que couche de transport.

BoaP a été implémenté et testée dans un petit réseau physique. Une méthode pour évaluer des intergiciels dans des réseaux DTNs est présentée. Un outil implémentant cette méthode a été développé. Il repose sur une plateforme de virtualisation qui simule les contacts réseaux tout en émulant les nœuds du réseau. Cet outil a été utilisé pour exécuter des expériences pour évaluer la validité de BoaP.

Enfin, un autre support de programmation est examiné. Celui-ci adopte une approche orientée service et respecte les contraintes REST (Representational State Transfer). Il se repose sur BoaP a été créé avec l'IoT en tête et est adapté à l'environnement DTN. La découverte exploite une interface de publications/souscriptions. Les descripteurs de services contiennent des champs spécifiques pour informer de la disponibilité de leur fournisseurs.

n d'ordre : 462

Université de Bretagne Sud
Centre d'Enseignement et de Recherche Y. Coppens - rue Yves Mainguy - 56000 VANNES
Tél : + 33(0)2 97 01 70 70 Fax : + 33(0)2 97 01 70 70