Specification of the disruption-tolerant protocol stack for the Web of Things

Deliverable 2.4 (version 1)

Université de Bretagne Sud - IRISA

December 23rd, 2015

Project ASAWoO

Adaptive Supervision of Avatar / Object Links for the Web of Objects

Grant Agreement: ANR-13-INFR-0012-04

Abstract

This document presents the specification of a disruption tolerant network protocol stack dedicated to the Web of Things (WoT). This protocol stack allows to discover REST services provided by physical objects, and to invoke these services either using CoAP or HTTP protocols. CoAP and HTTP messages are routed towards their destination using the ”store, carry and forward” principle.

Contents

1 Introduction

The “Web of Things” (WoT) extends the Internet of Things so that physical objects can be accessed and controlled using Web standards. In project ASAWoO, the concept of avatar provides a logical representation of physical objects in the Web. Avatars expose the functionalities of objects as REST Web services. An avatar is composed of the ASAWoO middleware platform and of a set of REST Web services that allow to interact with the physical object it represents. These Web services can be invoked by end-users through a Web application (WoTApp), and by other avatars.
Actually, two main needs for communication are present in ASAWoO

- Application components (ASWAoO Functionalities and WoTApps) are in general deployed on distinct devices. A WoTApp should be able to discover and invoke services provided remotely by Functionalities (these Functionalites are then marked as exposable). As Functionalities may be composite, Functionalities may discover and invoke other Functionalities remotely.
- The ASAWoO middleware platform may be itself distributed across several devices. Some of the middleware modules must then be able to access other remote modules.

The choice is made in ASAWoO to expose both Functionalities and middleware modules as RESTful services. Therefore a unique middleware support will be designed that allows the discovery and the invocation of remote RESTful services. These RESTful Web services should be accessible via the traditional HTTP (over TCP/IP) protocol but also via CoAP (over UDP/IP).
The connected physical objects considered in project ASAWoO can be mobile. They can communicate using their wireless interfaces, typically using their Bluetooth or their Wi-Fi interfaces. Due to their mobility and the short radio range of these interfaces, some disconnections can occur while devices are moving. To cope with these issues, we propose to implement in project ASAWoO a disruption-tolerant communication module that allows disruption prone objects to communicate together using Web standard protocols and technologies. This communication module will indeed offer a HTTP or COAP interface but will be able to rely on a DTN communication software to make the requests and responses of these protocols tolerant to disruptions.
Note that ASAWoO applications may not be deployed in a DTN environment but target a traditional connected environment (Internet). In this case, the ASAWoO communication module should be able to avoid relying on a DTN platform and rather use a more efficient ”legacy” HTTP or CoAP implementation. Moreover, it should be considered that some ASAWoO applications are aware that they are deployed in a DTN environment, and can explicitely indicate it in the requests and responses they generate, whereas some other applications may have been designed independently of the type of networks they will be deployed in.
Figure 1 illustrates the traversals of the protocol stacks implemented in the ASAWoO communication modules. An external WoTApp (running for example in a CoAP-enabled browser) invokes an ASAWoO functionality exposed as a RESTful Web service by robot C. The delay-tolerant network is formed by the three robots A, B and C. The CoAP request first reaches robot A in Wi-Fi (assuming that this one was close to the laptop that hosts the WoTApp when the request was issued). The request traverses the DTN, through robot B, eventually reaching robot C, thanks to the successive encounters (with Zigbee transmissions) between robots A and B and then between robots B and C.
The remainder of this document is organized as follows. Section 2 presents a scenario involving physical mobile objects that collaborate to perform a task and that exchange data all along this task. Based on this

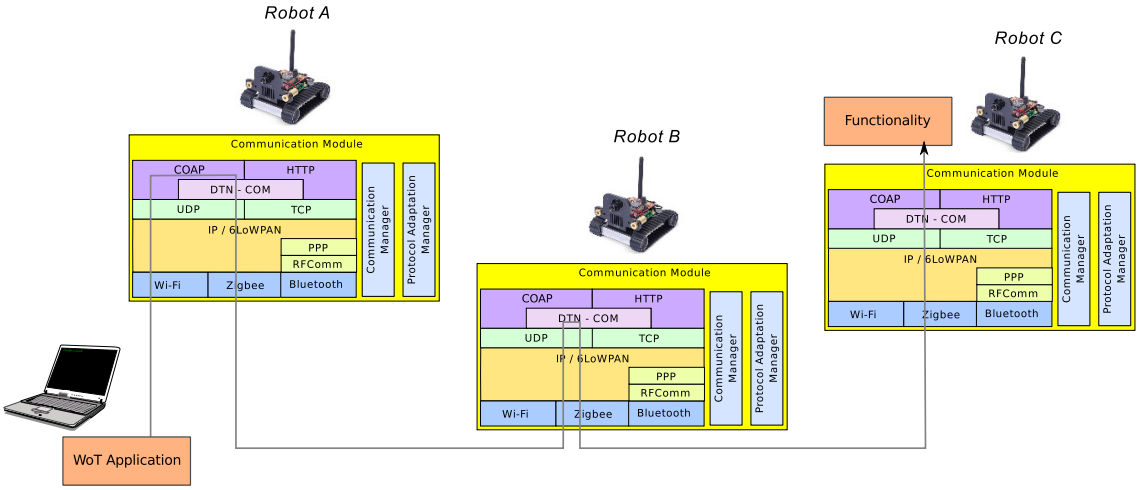

Figure 1: Example of traversals of the ASAWoO DTN stacks involved in the call from a WotApp to a Functionality

scenario, Section 3 justifies the use of distribution-tolerant or opportunistic computing techniques in the Web of things, especially for mobile objects equipped with short radio range wireless interfaces. This section also explains why a RESTful approach is preferable for a Web of things relying on distribution-tolerant or opportunistic computing techniques. Section 4 presents the core specification of the REST disruption-tolerant protocol stack we propose. It defines the spatial and temporal non-functional properties that can be exhibited by service providers and required by clients for the service provision process. These properties are inherent in the distribution-tolerant or opportunistic computing. Section 5 presents an extension of the core specification dedicated to geographic-aware services. Section 6 explains how service discovery can be achieved and how spatial and temporal non-functional properties are taken into account. Section 7 concludes this document by showing how service invocation and service discovery can be implemented using disruption-tolerant and/or opportunistic communication middleware platforms.

2 Scenarios

Communication modes (disruption-tolerant communication mode *vs* Internet-legacy communication mode) and usable protocols mainly depend on connectivity assumptions. These communication modes can vary according to the mobility of the devices, to the radio range of the wireless technology used to communicate, to the existence of an infrastructure, etc. So we have considered three test-beds, from the simple one to the most challenging one:

1. In the first scenario, the devices are motionless, or move in a limited geographic zone, and are always accessible through a network infrastructure. The resulting network is therefore considered as stable and fully-connected. This scenario is exemplified by an energy saving scenario as described in deliverable 2.2(https://liris.cnrs.fr/asawoo/doku.php?id=deliverable-2-2).

2. In the second scenario, most of devices are motionless, but unlike in the first scenario devices are distributed in a large geographic area. The resulting network is thus partitioned in several communication islands due to the limited radio range of wireless interfaces of devices. In some circumstances,

certain devices can be isolated. In order to provide communications between distinct communication islands, some mobile devices are used as data mules in our scenario. This scenario fits our vision of the Internet of Things in a smart city context.

3. The third scenario is a more challenging one. In this scenario, most of devices are mobile and autonomous. They move inside an area that is partially covered by a network infrastructure. In this scenario, communications between devices are not programmed and not predictable. Data exchanges are made opportunistically when devices are in the communication range of one another.

In the remainder of this section, we introduce scenarios 2 and 3.

2.1 Smart city

The smart city scenario can be seen as a geographic extension of home energy saving scenario. The goal is to monitor, manage and optimize all the urban infrastructure through a multitude of sensors (traffic detectors, instruments for measuring the air pollution, the noise level, the temperature, the brightness, the water quality...) and several actuators (coupled with gates, valves, shutters, signboards...). The resources are monitored, their data are collected and forwarded to some central structure, e.g. the city hall. These data then need to be aggregated and analyzed to choose which action should be performed in return. Such smart city already exists through large scale experiments , for example in the city of Santander. In the ASAWoO context, sensors and actuators are extended into the Web by avatars. We do not assume that the network of sensors and actuators is reliable and fully meshed ; many physical and financial impediments arise when we try to connect all sensors and actuators to a stable network infrastructure. In practice, short-range radio technologies (Wi-Fi, Bluetooth, NFC) are best suited to meet the energy, size and cost constraints of smart devices, as well as the network scaling. So a full mesh ensuring the flow of information between the network of sensors, the actuators and the central servers over an entire city at all times is unrealistic due to network partitioning. Data and commands exchanges between the avatar and its corresponding device may have to be entrusted to mobile carriers, as bus, taxis, municipality agents... The figure 2 resumes our vision of the architecture of a smart city. It describes how smart devices may respond to a query initiated by an end-user in the city hall. A bus passing nearby sensor gateways, or smart sensors, collects their data and carries them through the town. When a taxi intersects the bus, it stores and carries the data, forwarding them until they finally reach the city hall.

2.2 Precision viticulture

In the previous scenario, the communication paths and the communication delays to transfer data from sensors to gathering points, or from operators to actuators, are predictable since the mobile devices used as data mules follow regular mobility patterns. The scenario we consider in this sub-section encompasses the previous scenario, and adds opportunistic communication capabilities between some autonomous agricultural robots (or agribots).

The agricultural robotic is still in its beginning. It aims to provide to the agricultural industry the same productivity gains that robots give in manufacturing industries (e.g., automobile manufacturing). Agribots are part of the technology applied in agriculture that would maximize crop yields while preserving natural resources, financial and energy. Indeed the current trend is to shift from using big machines to smaller, lighter and more energy efficient unmanned machines that work as a team. These changing agricultural

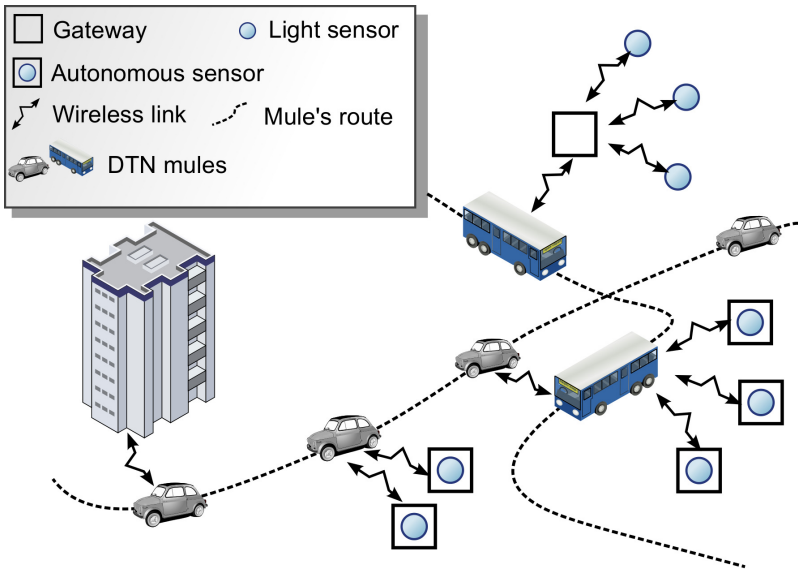

Figure 2: Smart city in a Delay Tolerant Network environment.

practices, known as precision farming term, have the same objectives of sustainable developments that smart cities. Several precision agricultural experiments involving agribots and sensors are already underway, around the world, particularly in the viticulture field (e.g. European projects FP7 VINBOT and Vinebot).
The scenario we propose takes place in a vineyard of several dozen hectares, far from any electrical infrastructure (see Figure 3). We also assume that mobile network technologies such 3G/4G, or satellite solutions, are too expensive and not suited to be used by all devices in large areas during long periods of times; only a few number of devices are connected to the Internet in 3G/4G, or occasionally connected to a local network using Wi-Fi.
The actors involved in the scenario are the manager, manual workers, agricultural machinery (tractors, etc.), agribots and sensors. The manager may interact with sensors and agribots thanks to a Web application running in a cloud infrastructure. He can access this application from his PC when he is in his office, or using his smartphone when he is in the vineyards. In this last situation, he may access to this Web application thanks to a 3G/4G connection, or to a Wi-Fi connection with a local network. He can also interact directly with these agricultural devices thanks to a specific Web application when he is close to them. Sensors monitor the essential agronomic parameters (soil composition, temperature, moisture, plant health,...) and transmit their data to the cloud infrastructure thanks to data mules that pass nearby (i.e., agribots, tractors equipped with a DTN communication device). These data are dated and georeferenced by the data mules. Data are delivered in the cloud infrastructure when a robot or an agricultural machine become connected to the Internet or to the local network.
Agribots work individually and independently, once informed of a new task to be done. For example, some agribots are in charge of weeding spaces nearby vineyards feet. At the same time other agribots have to visually estimate infected leaf area. Each agribot moves in a row, at its own speed. When an agribot has finished a row, he passes in front of a detector (e.g. a RFID tag) that allows him to inform another robot who would be tempted to redo the same row. Two agribots that pass within radio range of each other (crossing or adjacent row) can exchange information or instructions directly. When an agribot has finished its task, he goes back to its recharging base, and may collect and carry data in order to deliver these data to the cloud

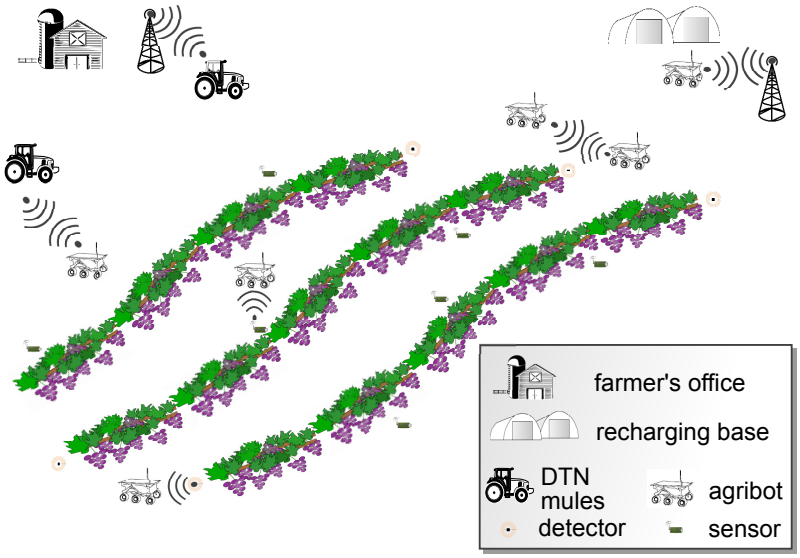

Figure 3: Precision viticulture in a Delay Tolerant Network environment

infrastructure. To avoid the interference between manual workers and robots, a site manager may prohibit the passage in some rows by programming the detectors placed at end of the rows.
This scenario of precision viticulture highlights the interaction needs between the human actors, the various sensors and detector scattered throughout the vineyard and the fleet of agribots. In such context, we argue that it is important to provide the human operators with a lazy supervision of the fleet of robots, so that they can achieve other tasks freely. The operators do not need to know the exact position of the robots or sensors at any time.

3 Disruption-tolerant Computing based on a RESTful approach

Even in disruption-prone environments, remote avatar functionalities as well as remote ASAWoO middleware distributed modules that are published as services should remain reachable. An Avatar publishes its functionalities through RESTful services, and some ASAWoO platform core modules can also be published remotely as RESTful services over HTTP. The avatar concept indeed revolves around the notion of resource where devices are represented as resourceful objects that exhibit their capabilities as functionalities. Therefore, communication between avatars is resource-oriented and that should not be altered by the DTN communication module.
As a consequence, the disruption-tolerant communication module should be accessed through web standard protocols like HTTP or CoAP. HTTP and CoAP can handle RESTful communications, which are required for homogeneity purposes with ASAWoO services.

3.1 Disruption-tolerant Computing

In order to deal with the frequent and unpredictable connectivity disruptions occurring between mobile objects equipped with short radio range interfaces in a WoT environment such that depicts in the previous section, project ASAWoO investigates the use of disruption-tolerant and opportunistic computing tech-

niques in WoT environments. Disruption-tolerant and opportunistic computing [3, 2] are close computing paradigms, and therefore rely on the same communication principle, namely the ”store, carry and forward” principle. Many delay/disruption-tolerant and opportunistic protocols have been proposed over the 10 last years [18, 14], but only few of them specifically address issues posed by the service provision in the absence of end-to-end connectivity in a network composed of mobile devices [17, 12, 16, 11].
The basic idea of the ”store, carry and forward” principle is to take advantage of radio contacts between devices to exchange messages, while exploiting the mobility of these devices to carry messages between different parts of the network. Two devices can thus communicate even if there never exists any temporaneous end-to-end path between them. Recent experiments conducted in real conditions have shown that applications such as voice-messaging, e-mail, or data sharing can indeed perform quite satisfactorily in networks that rely on the ”store, carry and forward” principle [15, 7, 10, 20].
Based on this principle, messages are, without specific limitations, disseminated in the whole network. Moreover, this principle induce additional delay in the message delivery process. This delay is directly related to the mobility of devices, to the contact opportunities and to the network density. Indeed, messages will be forwarded faster in a highly dense network composed of devices that move slowly, than in a sparse network formed by highly mobile devices.

3.2 RESTful Web Services

REST is an architectural style introduced by Roy Fielding in his PhD dissertation in 2000 [8] for distributed hypermedia systems. This architectural style imposed 5 main constraints and an optional one on these systems, notably for performance, scalability and simplicity purposes. This architecture style relies on client/server model (constraint 1). Clients and servers are decoupled and can interact through a uniform interface (constraint 2). Clients and servers can thus evolve independently, or can be replaced, as long as the interface between them is not altered. Clients and servers communicate through a stateless protocol (constraint 3). A session state can however be maintained on the client side. Clients must provide all the information necessary to serve a request. The session state can eventually be transferred by the server to another service, for instance to maintain a persistent state for a period (e.g., a database) and allow authentication. Responses can be cached by clients and intermediate hosts (constraints 4). Responses must therefore, implicitly or explicitly, define themselves as cacheable, or not, to prevent clients from reusing stale or inappropriate data contained in responses to next requests. Such a caching process improves the scalability and the performance of the systems since it partially or completely eliminates some client/server exchanges. According to the last strict constraint (constraint 5), distributed hypermedia systems must be designed as hierarchical-layered systems to reduce the overall complexity of the systems and to improve their scalability, as intermediate layers can be used to perform load balancing and to store data in caches. The optional constraint pertains on the provision of pieces of code on demand by servers to clients to extend or to customize the functionality of a client (e.g., JavaScript scripts).
Web service APIs that comply with the above architectural constraints are called RESTful APIs. HTTP-based RESTful APIs are defined by a base URI (e.g., http://example.com/resources/), an Internet media type for the data (e.g., JSON, XML), standard HTTP methods (e.g., GET, PUT, POST, or DELETE), hypertext links to reference a state and hypertext links to reference-related resources.
The Constrained Application Protocol (CoAP)[19], is a specialized web transfer protocol based on request/response for use with constrained nodes and constrained networks in the Internet of Things. The

protocol is designed for machine-to-machine (M2M) applications. Like HTTP, CoAP is based on a REST architecture: servers make resources available under a URL (e.g., coap://example.com/resources/), and clients access these resources using methods such as GET, PUT, POST, and DELETE.

3.3 Benefits of a RESTful approach to enable disruption-tolerance in ASAWoO

The REST architectural style is well adapted to the disruption/delay-tolerant and opportunistic computing. Indeed as shown in [16], by decoupling the client and the server parts of a distributed application (constraint 1), by specifying their interactions with a well defined interface (constraint 2), and by resorting to stateless service (constraint 3), one can substitute a service provider by another one, and we can take advantage of the redundancy of providers in the network to improve the overall performances of the application. Moreover, it can be difficult for a server to maintain a session with a client, because it is not necessarily connected directly to this one, but through intermediate hosts. Due to the mobility of these hosts, the communication path between the server and the client can be [broken irremediably / subject to disruptions]. So, it is preferable to maintain a state on the client side. By implementing the ”store, carry and forward” principle, disruption/delay-tolerant and opportunistic communication platforms allow to store in the cache of clients and of intermediate nodes the responses, but also the requests that have been sent in the network (constraint 4). By implementing a proxy-based approach such as that presented in [13], intermediate hosts can respond on the behalf of a server if they have the response in their local cache, when this one is still valid. Such an approach allows to improve the performance and the scalability of the system, because it naturally performs load balancing and by storing data in cache of intermediate hosts as referred in constraint number 5.

3.4 Requirements for disruption-tolerant RESTful Web Services

In order to cope with devices mobility and service disrutpion issues described in the introduction, previous section showed the benefits of making ASAWoO RESTful web services tolerant to disruptions. This section lists the requirements to enable disruption-tolerant RESTful web services in ASAWoO platform. The proposed specification should therefore fulfill the following requirements, ranging from the use of standard protocols to the the seamless integration in ASAWoO Communication Manager module.

3.4.1 Enable REST communications through standard web protocols : HTTP and CoAP

The Web of Things as defined in ASAWoO allows to interconnect devices and make their functionalities accessible through standard web protocols. HTTP has been widely adopted as a de-facto transport protocol when implementing REST architectures, since HTTP protocol matches REST requirements (GET, PUT, POST and DELETE methods are CRUD operations).
The web transfer protocol called Constrained Application Protocol (CoAP), which has been designed to target constrained nodes in the Internet of Things, is based on REST. As such it is resource-oriented, and like HTTP, resources can be accessed via the same methods (GET, PUT, POST and DELETE).
For these reasons as well as for homogeneity purpose with the transfer protocol ensuring communications between avatars in ASAWoO Web of Things, the protocol stack specificied in this document should use HTTP or CoAP as the web transport protocol.

3.4.2 Leverage disruption-tolerant computing using non-functional properties

In order to bound the propagation of messages in time and space, the opportunistic protocols traditionally define for each message a lifetime and a maximum number of hops (i.e., the maximum number of times it can be retransmitted). Some protocols also allow to circumscribe the propagation of messages in a given geographical area [11, 12]. Time and space being the two major factors of service disruptions, when such temporal and spatial boundaries exist, it may be worth expressing similar constraints at the application level in a cross-layered approach. These constraints could thus be used by both the disruption-tolerant middleware to manage messages propagation, and the application layer when discovering or publishing services to narrow service provider/consumer matching.
Service providers can indeed include, for each service they offer, such non-functional properties in their service advertisements and service responses in order to define in which area the services are available, and until when advertisements and responses must be considered as valid. Similarly, clients can include these non-functional properties in their service discovery requests and service invocation requests in order to define to which area they are looking for a service, and to precise the maximum delay they expect for a response. This delay must be less than the lifetime of advertisements and invocation requests. These properties can then be used by the communication module underlying layers that are responsible for carrying HTTP or CoAP queries in a disruption-tolerand way to forward the messages towards their destination.

3.4.3 RESTful disruption-tolerant service discovery and invocation

Even in disruption-prone environments, remote avatar functionalities as well as remote ASAWoO middleware distributed modules that are published as services should remain reachable. An avatar publishes its functionnalities through RESTful services, and some of ASAWoO platform core modules can also be published as RESTful services to be used remotely by resource-constrained devices. The avatar concept indeed revolves around the notion of resource where devices are represented as resourceful objects that offer their capabilities as functionalities. Therefore, communication between avatars consists in resource-oriented service invocations, and this approach should not be altered by the communication module that would make service invocations disruption-tolerant.
Service discovery will be regarded as a separate concern. Each avatar can indeed manage its own service registry which can be itself provided as a RESTful service. Therefore, discovery services published this way can be invoked through the disruption-tolerant communication module like any other services.

3.4.4 Avatar identification and addressing

Physical objects can use different wireless technologies, so avatar identification should not depend on the wireless technology. Furthermore, as different disruption/delay-tolerant and opportunistic communication middleware platforms are likely to be used to perform service provision in a WoT environment composed of mobile devices, it is suitable to identify mobile devices with a unique identifier that is also independent of a specific platform.
Since several opportunistic networking techniques rely on broadcasting messages on multicast or anycast channels, beyond supporting end-to-end addressing, the disruption-tolerant communication module should also consider broadcast, multicast and anycast addresses. Some works have investigated anycast communication in opportunistic networks [5, 21].

Moreover, even at service level these kinds of addressing should be enabled. When spatial constraints are defined, service advertisements and service discovery requests must be disseminated, received and treated respectively by all service clients and service providers located in the area specified in the messages. Thus, a multicast addressing scheme would help to implement such a forwarding process. Service invocation traditionally relies on a point-to-point communication model and on a unicast addressing scheme, where a client sends a request to a given provider, and where the provider returns a response back to the client. Sometimes, for performance reasons, it could also be relevant to send a same service invocation request to several service providers at the same time. Such an operation requires an anycast addressing scheme.
Similarly to avatar identification, anycast and multicast addressing scheme must be as generic as possible.

3.4.5 Seamless DTN support

Developers should not necessarily know beforehand whether their applications will or will not run in a disruption-prone environment. Nonetheless, developers should still be able to produce disruption-aware code if they are aware that the services used or provided by their applications might not be available.
Hence the followlwing two requirements:

- Developers should be able to leverage disruption-tolerant routing by defining specific service-level constraints on service providers and consumers. These constraints can be expressed by the non-functional properties previously mentionned in section 3.4.2.
- Communication module should be usable in a transparent way. If no DTN-related constraint is defined, an implicit default mode should be used to handle communications when running in a disruption-prone environment.

The decision of switching from a disruption-tolerant communication mode to default mode is left to ASAWoO context manager. Depending on the context, ASAWoO context manager should enable or disable [use or ignore ?] the disruption-tolerancy features of the communication module. This kind of switch mechanism would allow to spare network resources as well as computing resources since affording superfluous DTN support overhead would be a waste when running in a static fully-connected environment.

4 Core specification

This section presents the specification of the core of the disruption-tolerant communication module of the ASAWoO middleware platform. It gives an overview of the architecture of the protocol stack, how avatars are identified, the format of the URLs used to access REST Web services, and the temporal and spatial constraints associated with REST Web services that must be delivered by a disruption-tolerant communication system.

4.1 Overview of the protocol stack architecture

As mentioned in Section 1, project ASAWoO aims at controlling and at interconnecting physical objects using standard Web protocols and technologies. In project ASAWoO, REST Web services deployed on the ASAWoO middleware platform can be accessed either using the application-level protocols HTTP and CoAP (see Figure 4). The application-level messages (i.e., HTTP and CoAP messages) can be encapsulated

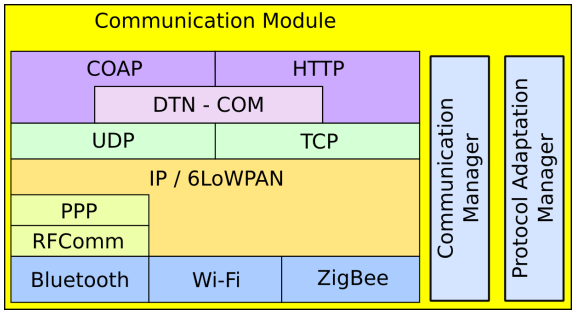

Figure 4: Architecture of the protocol stack.

in UDP datagrams, in TCP segments or in messages of a given disruption-tolerant communication system in order to be transmitted to their destination.
It must be noticed that service discovery will be considered as a separate concern. Each avatar can indeed manage its own service registry. In our approach, this registry is exposed as a REST Web service for discovery purposes. This service can therefore be invoked like any other services *via* the application-level protocols HTTP and CoAP, and using Internet-legacy protocols or disruption-tolerant protocols. This service discovery process is further specified in Section 6.

4.2 Identification of avatars

In project ASAWoO, an avatar can be deployed on the physical object it represents, can be distributed both on the physical object and on a remote host (or a cloud infrastructure), or can be installed only on a remote host (or a cloud infrastructure). A host can thus accommodate several avatars at the same time. Avatars are identified by a unique ID, thus allowing to distinguish the different avatars running on a same host. This ID is a short string (of 8 characters) encoded in Base62. A user-friendly name (i.e., an alias) can be associated with this ID. An avatar can be accessed through its ID and the address (or the name) of the host on which it runs. The address of the host depends if the transmission of the application-level messages is achieved using Internet-legacy protocols (i.e., TCP/IP, UDP/IP) or a disruption-tolerant communication system. In this last case, the address depends on the communication system. For instance, in the C3PO opportunistic communication middleware platform, hosts are identified by a string (of 6 characters) encoded in Base62. It must be noticed that, if a host must be accessed using both Internet-legacy protocols and a given disruption-tolerant communication system, it is advisable to assign the same alias to the IP address of the host and to its address provided by the disruption-tolerant communication system, thus allowing to access this host with the same alias whatever the communication mode.
The syntax that must be used to access an avatar is the following:

```
<avatar URL> ::= <host name>[":"<port>]"/"<avatar name>

<host name> ::= (<host alias> | <host address>)["."<domain>]

<avatar name> ::= <avatar alias> | <avatar id>
```

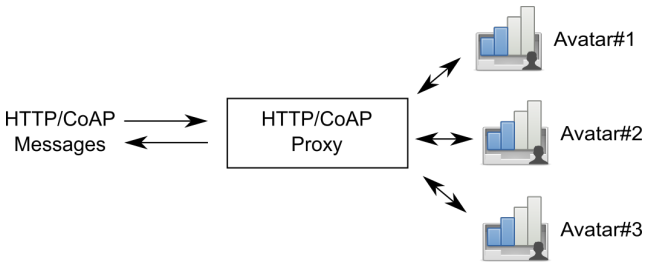

Figure 5: Host running a HTTP/CoAP reverse proxy and several avatars at the same time.

When a host accommodates several avatars simultaneously, the alias or the ID of the avatar that must be accessed must be specified. This host must run a (HTTP/CoAP) proxy in order to forward the application messages it receives to the right avatar (as illustrated in Figure 5). On the contrary, when a host accommodates only one avatar, the alias or the ID of the avatar is optional. When an avatar is distributed on both the physical object it represents and a remote host, the avatar can indifferently be accessed on the two devices.

4.3 URL format

REST Web services are traditionally accessed from a URL, through a point-to-point client/server communication model. In the Web of Things (WoT), it seems interesting to be not limited to this simple communication model, and to adopt new communication models, that are not necessarily relevant for traditional Web applications, but that have a sense in the WoT, such as anycast, multicast and broadcast communication models. Indeed in some scenarios, such as those presented in Section 2, it can be useful to send a service request to several objects (e.g., sensors) simultaneously without naming them explicitly. Similarly in local area networks, avatars should be able to announce their presence to other avatars, or to advertise the services they offer, using a broadcast communication model. These communication models have the advantage of reducing the network load, while improving the response time in comparison of a sequential unicast invocation of several services. The disruption-tolerant protocol stack implemented in the communication module of the ASAWoO middleware platform integrates these different communication models. In order to remain consistent with the RESTful approach, these different communication models are specified in the scheme of the URLs used to access REST Web services. The general format of a URL is the following:

```
<service URL> ::= <scheme>"://"<destination>"/"<avatar name>"/"<resource>["?"<parameters>]
<scheme> ::= ("http"["s"]|"coap")["+dtn"]["+acast"]["+mcast"]["+bcast"]
<destination> ::= (<host name> | <anycast name> |
                   <multicast name> | <broadcast name>)[":"<port>]
<anycast name> ::= <anycast alias> | <anycast address>
<multicast name> ::= <multicast alias> | <multicast address>
<broadcast name> ::= <broadcast address> | "*"
```

The first part of the scheme indicates the application-level protocol used (i.e., HTTP, HTTPS, or CoAP) to communicate with a remote service. ”+dtn” must additionally be specified in the scheme, if CoAP and HTTP messages must be forwarded by a disruption-tolerant communication system. By default, messages

are transmitted using a unicast communication model. If another model must be employed for the transmission, it must be specified in the scheme. "`+acast`", "`+mcast`" and "`+bcast`" specify respectively that an anycast, a multicast and a broadcast communication model must be used in the forwarding process. It must be noticed that the transmission of HTTP messages using a multicast communication model has been proposed in the past as an Internet draft [9].
Depending on the communication model, the "destination" part of the URL can designate the name or the address of a host, of a multicast group or an anycast group. It can also be a broadcast address or the wildcard "*".
A domain name can be added to access a remote host using Internet-legacy protocols. A port number can also be specified if the remote host(s) do(es) not use the default port numbers associated with protocols HTTP, HTTPS and CoAP.
The "`avatar`" part of the URL designates the name or the ID of the avatar. This part must be specified if the remote host(s) accommodate(s) several avatars simultaneously, otherwise it is optional.
The "`resource`" part identifies the resource that must be created, read, updated or deleted (CRUD operations).
Additional parameters can be also specified in order to define non-functional properties for disruption-tolerant and opportunistic computing as presented in the next sub-section.

4.4 Non-functional properties for delay-tolerant and opportunistic computing

In delay-tolerant and opportunistic networks, service messages (i.e., service discovery requests, service advertisements, service invocation requests, service responses) are forwarded following the "store, carry and forward" principle. In some circumstances, the propagation delay and dissemination area of messages must be bounded. In the remainder of this section, we list a set of non-functional properties defining temporal and spatial constraints that can be expressed by service clients and service providers regarding the service delivery conditions. These non-functional properties can be exploited by disruption-tolerant or opportunistic communication systems in the message routing process.

4.4.1 Caching parameter

Service clients can specify in their requests if the latter can be cached by intermediate nodes. If so, intermediate nodes will store in their cache both the request and the response associated with this request until they expire. Thus, they can reply later to a similar request sent by any node on the behalf of the service provider (i.e., by returning immediately the cached response, instead of forwarding the request towards the service provider). This parameter is named *dtn_cacheable.* Then, intermediate nodes can remove from their local cache a request when they receive a response to this request, thus implementing a network healing mechanism. Although they can keep this request in their local cache, they must not process them in order to reply to a service client. Such a principle has been presented in [13].

4.4.2 Time parameters

Temporal constraints can be expressed in URLs as query strings or in the payload of application-level messages. These constraints are defined as a relative time from the message creation time. They are identified respectively by *dtn_ctime* (creation time) and *dtn_etime* (expiration time).

dtn_ctime is expressed as a long integer that represents the difference, measured in milliseconds, between the current time and midnight, January 1, 1970 UTC.
dtn_etime is expressed as an integer that represents the difference, measured in milliseconds, between the creation time and the expiration time.
The expiration time, can also be specified as a symbolic social time using the parameter *dtn_stime*. The values of this parameter are strings defined in an ontology (e.g., afternoon, evening, tomorrow, ...). The values are prefixed by the "@" symbol as shown in the following example:

```
Example:  dtn_stime=@midnight
```

These constraints have a different signification depending on the type of messages. In a service invocation request, these constraints express the fact that the client wants to get a response before the expiration time specified in the request message. The request can be disseminated in the network, and a provider of the service can answer to this request until it expires.
When specifying time constraints for a response, these constraints express the lifetime of the response. This lifetime may be:

- equal to the lifetime of the request (i.e., the response must be returned to the client before the deadline specified in the request),
- inferior to the lifetime of the request, if information contained in the response becomes invalid before the lifetime of the request expires,
- or greater than the lifetime of the request, if information contained in the response is still valid after the lifetime of the request expires. In this case, the response can be cached until it expires too.

In service discovery requests (e.g., GET method on a service registry), temporal constraints have the same signification than in the service invocation requests. In service advertisements (e.g., POST method on clients), these temporal constraints specify how long the service advertisement can be considered valid, (i.e., how long the service should be expected to be provided). Service advertisements can indeed be cached until they expire.
These temporal constraints will be used by the disruption-tolerant communication systems to determine how long a message can be stored in a cache or forwarded through the network.

4.4.3 Number of hops

A number of hops can additionally be specified in application-level messages in order to circumscribe at a coarse grain the area in which the messages can be disseminated, and to avoid that a message eternally roams in the network.

5 Extension specification

This section presents the extension part of the specification. Extended specification is about non-functional geographical properties and callback functions for asynchronous communications. These properties can be specified by service clients and service providers in application-level messages, but are not necessarily supported and taken into consideration by all disruption-tolerant communication platforms.

5.1 Location parameters

The dissemination of messages can approximatively be circumscribed to an area using a limited number of hops. Nevertheless , this limitation is not exact, and does not guarantee that a message cannot be forwarded outside an expected area. Indeed, a host can forward a message to another host outside this area if the maximum number of hops of the message has not been reached. In order to limit the propagation of messages in the physical environment, geographical properties can be defined. Geographical areas may be defined as squares or circles relatively to a given GPS position. The GPS point is identified by *dtn_src_pos*. This point is defined by 3 floating values (longitude, latitude, altitude) as shown below:

```
Example:  dtn_src_pos=47.234,-2.123,100
```

The location of service providers can also be specified by clients in their service invocation requests if they know this location. This property is identified by *dtn_dst_pos*. It is defined in the same manner as *dtn_src_pos*. This property can be used in the message forwarding process as shown in [11].
A circle is defined by a GPS point and a radius expressed as a floating value. Metric units must be used. In the following example, the altitude is not specified.

```
Example:  dtn_area=47.234,-2.123,,100
```

A square is defined by a GPS point and 2 distances expressed in meters.

```
Example:  dtn_area=47.234,-2.123,,100,100
```

The location property can also be defined by a symbolic name (e.g., home, office). This property is identified by *dtn_slocation*:
The location property can also be defined as an address. This property is identified by *dtn_address*. A location service (OpenStreetMap, GoogleMaps, ...) is required to obtain the GPS coordinates for this address.

5.2 Asynchronous communication

Service clients can add in the query string part of the URL or as a path-parameter, a parameter "*callback*" in order to define the URL that must be used by service providers to return the response. Thus, clients can process the responses they receive asynchronously without being blocked by the reception of service responses. As show in the following example, a callback references a URL.

```
Example:  callback=http://my_host/my_avatar/response_processor
```

6 Service management

The list of services offered by a node, as well as discovered and registered services, are maintained on a node's local registry. The node broadcasts its local services when it starts a mission ; he also registers the discovered broadcast services. Nevertheless, when a node joins an already started mission or encounters a new node, it can also ask the newly encountered node its services. For this purpose the service registry is published as a RESTful service that can be invoked like any other service.

6.1 Service discovery and registration

The service management module maintains two categories of services: local or remote.
The local services are advertised (POST service/announce/name) at the beginning of the mission. Then they are also advertised in response of a service discovery request (GET service/request/name or GET service/request/*). This service request may have been broadcast (in anycast or broadcast sending mode) or may have been addressed to a node (unicast sending mode).
A remote service is added to the service registry when the node receives a service advertisement (POST service/announce/name) .

6.2 Service description

The service description contains five elements. Two are mandatory, namely the identification and the type, the three others (i.e., geographical and schedule availability details and the semantic description document in case a service should be deployed after the initialization of the mission) are optional. As explained in the previous deliverable, services are described by their functional and non-functional properties using JSON-LD-based semantic notations.
Once a mission has started and as soon as the nodes are deployed, they do not need to discover the whole semantic documents as they are known beforehand.

6.2.1 Identification

Services must be identified. A service cannot be invoked if it is not identified, requests must be targeted. This identification can be likened to request destinations that are, both in HTTP, CoAP and the BP, URI.
That is why the service URI must be included in the service description.

6.2.2 Type

While several types of services can be deployed, different devices may offer the same type of services, and thus the same interface. For example, a given sensor may end up being deployed twice or more; hence offering the same service at multiple times and positions. Instead of including the whole description of functional and non-functional properties, and as the different service interfaces are already known by the nodes, the service description only needs to declare which service interface is implemented by the service provider.
All interfaces must then uniquely match a type. This type can take the form of a hash (especially if the interfaces are stored in a table), a number, an enumeration, or any format that can be used to uniquely identify the interfaces.

6.2.3 Time restrictions

Devices may restrict their service availability depending on time constraints. This restriction can be useful when a device relies on some scheduling such as working hours, sleep time for battery issue or any other reason. Services can thus provide a schedule that nodes should respect to spare network resources as the provider may not answer outside the described schedule.
In order to inform clients of their schedule availability, services must include in their description their frequency of wakening and duration.

```
{"schedule":[
    {"start": crontab_format, "duration": duration_in_seconds},
    {"start": crontab_format, "end": crontab_format}
]}
```

6.2.4 Geographical restrictions

In addition to time restrictions, some service providers may restrict their advertisements and discovery areas. These restrictions allow to bound the dissemination of the local services descriptions, and thus avoid to be requested from too far. This way, opportunistic networks are then scattered offering a better managing policy and ensuring a better bandwidth usage.
To restrict their availability area, services must include the area definition using positions as defined in 5.1. Circles can be defined by a GPS position and a radius while rectangles can be defined by a GPS position and 2 distances expressed in meters. A list of areas can be provided to define more complex geometric figures.
The format to describe the area is:

```
{"area":[
    {"shape: "rectangle",
     "start": {"lat": lat, "long": long},
     "stop": {"lat": lat, "long": long}},
    {"shape: "rectangle",
     "start": {"lat": lat, "long": long},
     "size": {"width": lat, "height": long}},
    {"shape: "circle",
     "center": {"lat": lat, "long": long},
     "radius": radius}
]}
```

6.2.5 Semantic description

If a whole new service is deployed dynamically in an already-running environment, clients need the semantic description of this service to be able to use it. This last optional field, semantic description, should not be useful in most cases as the nodes are supposed to know beforehand all deployed services, yet it allows new services to be deployed.
To do so, the service description must include the document describing their functional and non-functional properties using JSON-LD-based semantic notations.

6.2.6 Overview

As explained below and pictured in 6, the service description contains five different fields:

- Two mandatory:

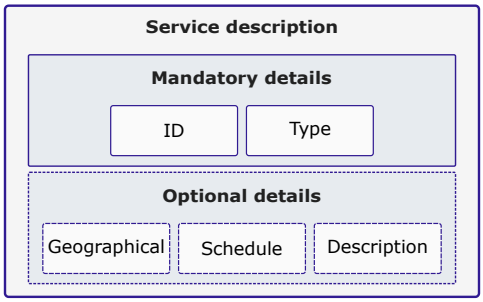

Figure 6: Service description overview

  - ID: identification of the service in its URI form.
  - Type: the identification of the interface exposed by the service. The different interfaces are known beforehand, so the whole interface does not need to be included entirely.

- Three optional:
  - Schedule details: to constrain the usage of the service according to a specific schedule (availability of the device).
  - Geographical details: to constrain the usage of the service in a specific area.
  - Description: to include a new service non-functional and functional semantic document dynamically deployed.

7 Bindings with disruption-tolerant communication platforms

An implementation of the specification of the ASAWoO delay-tolerant REST services provision presented in the previous sections relies on the ability to transfer messages in a delay-tolerant network. Some platforms offering disruption-tolerant (often abbreviated as DTN) communication are available. However, they may differ in interfaces and capabilities. Binding ASAWoO with a given DTN platform may raise issues regarding the supported protocol, addressing, deployment, or other practical issues. As ASAWoO provides HTTP or CoAP interfaces for service provision and discovery, an adapter is required to accomodate the underlying DTN platform. This adapter includes a HTTP/CoAP proxy server in charge of catching HTTP/CoAP requests and responses, and a DTN adapter that will include these requests and reponses in adequate DTN messages, making use of a dedicated DTN library. Figure 7 depicts the general architecture of a binding in the case of a HTTP proxy.
In this section we discuss three different bindings with available DTN platforms, namely Bundle Protocol, C3PO and DoDWAN.

Session management

As the DTN ASAWoO requests and responses should include non-functional properties such as their lifetime, this information can be used to close a HTTP or CoAP session at the proxy-level in order to avoid maintaining unecessary connections between clients and servers.

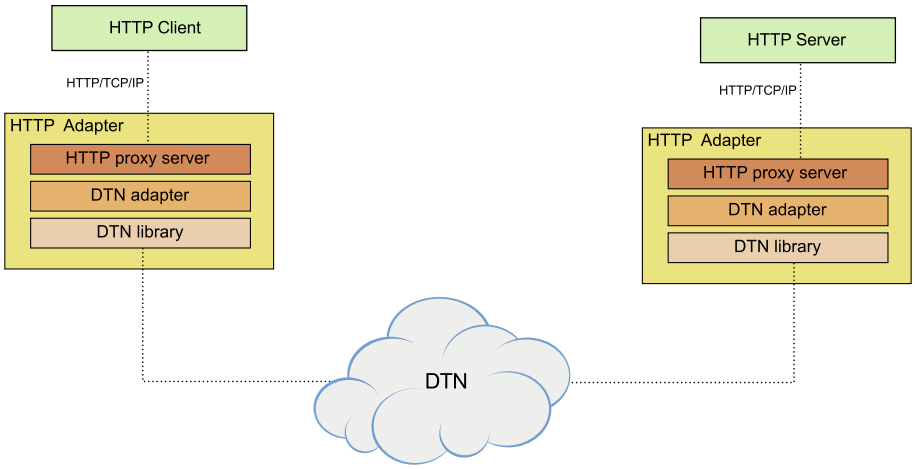

Figure 7: Architecture of a ASAWoO binding

7.1 Bundle Protocol (BP) [1]

7.1.1 Protocol

The BP is the de facto standard for the bundle-layer of the DTN architecture [6]. 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 subnet-specific protocols called Convergence Layers (CL) 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 to store bundles. These bundles are either processed by an application (if the destination is on the node), or forwarded to other nodes toward the bundle destination. A bundle destination (or bundle endpoint) is identified by an Endpoint IDentification (EID) that takes the form of a URI. As there is no central authority to manage IP address space, BP relies on a specific independent identification mechanism based on URI with the *dtn* scheme. These URIs are used to identify BP endpoints. An EID can either be a singleton (i.e. dtn://sensors0a1e/temperature) or a set of BP endpoints that register themselves with a common EID (i.e. dtn://all/sensors/temperature), thus allowing multicast-like operations to be performed.

The BP bundles have to be routed from node to node. The BP specification does not specify a routing method in particular. Many routing algorithms intended to be adapted to a networking context (e.g., the mobility of the nodes) or to a type of application can be used. 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).

Bundles are constituted of one primary block (header) 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.

7.1.2 Implementation

The BP implemention chosen for the development of ASAWoO is IBR-DTN[4]. IBR-DTN runs as a daemon that listens on a local TCP socket. Command line IBR-DTN tools communicate with the daemon

through the local TCP socket using the API offered by the daemon. An IBR-DTN Java interface has been developed to ease the communication between BP Java applications and the IBR-DTN daemon.
In our case, a CoAP client sends a local request to the HTTP/CoAP proxy by embeding the URI of the destination as a CGI parameter, see 4.3. The HTTP/CoAP Proxy, acting as a BP application, will then create a bundle if the requested transport protocol is the BP. This bundle will be destinated to the node matching with the URI passed as CGI parameter. This bundle, once sent, will be forwarded and carried through a set of relay nodes running IBR-DTN. These relay nodes do not need neither the IBR-DTN Java interface nor the Proxy to disseminate the bundle. When the bundle reaches its destination, the IBR-DTN daemon will pass the bundle to the local BP endpoint matching the URI, that is the local HTTP/CoAP Proxy. The proxy will then send locally the request to the CoAP server, wait for the response, encapsulate this response in a bundle and send it back to the client. Note that the response may be forwarded through a whole different set of BP relay nodes.
The Figure 8 presents the architecture stack of CoAP deployed with the BP as the transport protocol.

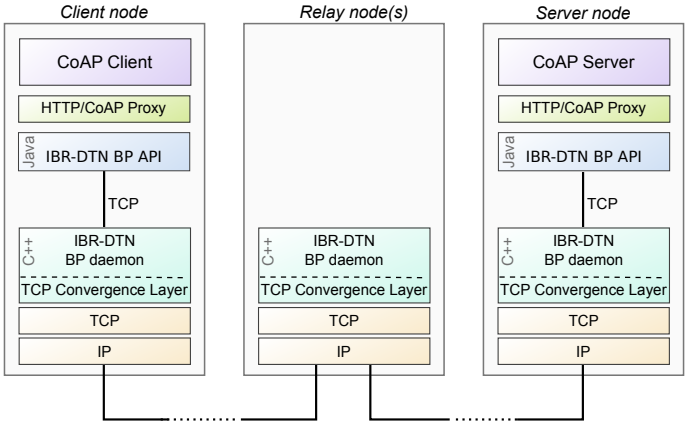

Figure 8: BP architecture stack

7.2 C3PO

The C3PO platform provides different types of messages for service discovery and service invocation. These types are identified respectively by the values `SERVICE_REQUEST`, `SERVICE_RESPONSE`, `SERVICE_DISCOVERY_REQUEST`, `SERVICE_ADVERTISEMENT` of the header `MESSAGE_TYPE`. Service discovery requests and service advertisements can be disseminated in the network using a predefined topic, such as ”SERVICE_MANAGEMENT”. All the devices connected to the network and running the ASAWoO platform (and that have subscribed to this topic) will receive these messages. The service invocation traditionally relies on a point-to-point communication model, and therefore could be implemented with the channel-based communication paradigm proposed by the C3PO platform.
The C3PO platform defines additional message headers that can be used to take into account the disruption-tolerant properties specified by the application services in the message forwarding process. These additional message headers are:

- EMISSION_DATE: date of message emission,
- LIFETIME: lifetime of the message (relative time expressed from the emission date),

- NB_HOPS: the maximum number of hops the message can make,
- SOURCE_GPS_LOCATION: the GPS location of the emitter,
- DESTINATION_GPS_LOCATION: the GPS location of the destination,
- GEOGRAPHIC_AREA_RESTRICTION: the area in which the message can be disseminated,

In the remainder of this sub-section, we describe both the topic-based publish/subscribe paradigm and the channel-based send/receive paradigm.

Topic-based publish/subscribe paradigm The topic-based publish/subscribe communication model makes it possible to develop applications that can publish multimedia contents on specific topics, and subscribe to given topics in order to receive related content. The model implemented relies on a purely peer-to-peer decentralized approach. Topic names are assumed to be already known by the applications, a given topic is generally pre-defined or handled by a single application module that performs both production and consumption. The subscription and the publication are local to each device. Thanks to the store-carry-and-forward principle, contents published in a topic by publishers (*P* in Figure 9) are disseminated opportunistically in the communication network by mobile devices, being either devices hosting subscribers for this topic or ordinary intermediate devices (resp. *S* and *I* in Figure 9), and are thus delivered to the topic subscribers.

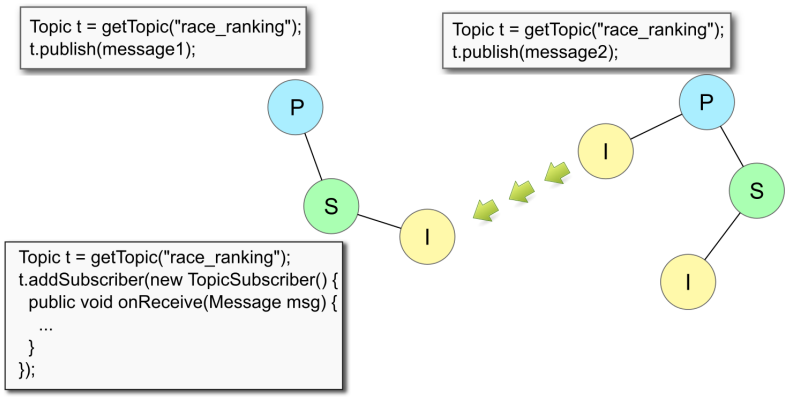

Figure 9: Communication between mobile devices using the publish/subscribe paradigm

Channel-based send/receive paradigm The point-to-point communication paradigm using the concept of channel is intended for applications that allow users to communicate with each others by sending messages addressed to specific recipients. In the framework, a channel between two devices is identified by the addresses of the devices and a channel ID. Messages sent through a channel are opportunistically forwarded by intermediate devices (*I* in Figure 10) towards their destination according to one of the message forwarding strategies implemented in the framework. Thus, two devices can exchange data even if they are not within mutual radio range. The API we have developed is illustrated in Figure 10. Device D_1 obtains a reference to a new channel object by providing the address of the remote device D_2 and a channel ID. Using this channel object, D_1 can send messages to D_2. These messages will be stored, carried and forwarded by intermediate devices until being delivered to D_2. Device D_2 waits for a channel establishment through

the `accept()` method. As soon as it receives a request from D_1, the channel is established at either sides, and D_2 can start receiving messages from D_1. Messages can be received either by calling the `receive()` method or by registering a message listener to this channel.

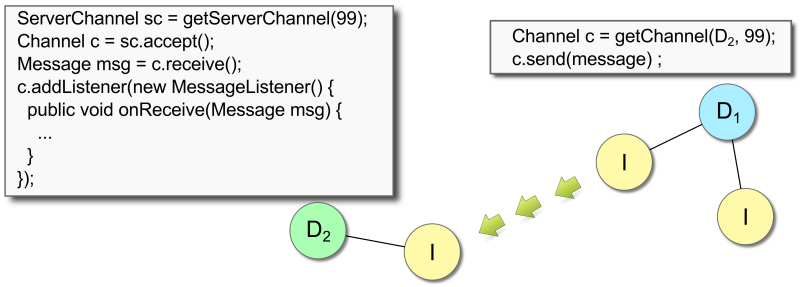

Figure 10: Communication between mobile devices using the point-to-point send/receive paradigm

7.3 DoDWAN

DoDWAN (Document Dissemination in mobile Wireless Ad hoc Networks) is a Java-based middleware platform developed by CASA/IRISA in order to support communication in opportunistic networks.
DoDWAN implements a selective version of the epidemic routing model proposed in [ER]. It provides applications with a publish/subscribe API. When a message is published on a device, it is simply put in the local cache maintained on this device. Afterwards, each contact with another device is an opportunity for the DoDWAN system to transfer a copy of the message to that device. In order to receive messages an application must subscribe to DoDWAN and provide a selection pattern that characterizes the kind of messages it would like to receive. The selection patterns specified by all local applications running on the same device define this device's interest profile. DoDWAN uses this profile to determine which messages should be exchanged whenever a contact is established between two devices. For the time being, DoDWAN implementation relies on UDP or TCP (over IPv4 or IPv6) in order to exchange messages between neigbouring devices. No assumption is made about the underlying MAC and Physical layers.
In some practical opportunistic networks, DoDWAN can be associated with an external piece of software that plays the role of a connectivity manager. This connectivity manager dynamically creates and destroys so-called communication channels according to the ability of the device to exploit a communication interface (through which other neighbour devices are potentially reachable). At the moment, UDP channels (unicast or multicast) and TCP channels (unicast) can be used. In simple scenarios (for example when a device accesses a Wi-Fi ad-hoc network), no connectivity manager is required and only one initial channel (a multicast-enabled UDP channel in the case of a Wi-Fi ad-hoc network) is specified in the DoDWAN configuration and created automatically.

7.3.1 DoDWAN API

With DoDWAN, devices are identified with a name uniquely defined in the network. There is no constraint on the naming: any string of characters can be used.
DoDWAN supports the dissemination of so-called "messages". A message contains a descriptor and a payload. A descriptor consists in a list of attributes in the form of pairs of strings (key, value). Two mandatory attributes must be present in a descriptor (message ID and lifetime) and the application can add

```
Dodwan dodwan = new Dodwan();

dodwan.online();

Descriptor desc = new Descriptor();
desc.setUniqueDocumentId();
desc.setDeadline(new Date(System.currentTimeMillis()+60000));

desc.setAttribute("type", "command");

dodwan.pubsubService.publish(desc, new String("turn left").getBytes());
```

```
Dodwan dodwan = new Dodwan();

dodwan.online();

Descriptor pattern = new Descriptor();
pattern.setAttribute(("type", "command|config");

dodwan.pubsubService.addSubscriber("sub1", pattern, new MyProcessor());

...

public class MyProcessor implements Processor<Descriptor> {
   if (desc.getAttribute("type").equals("command")
        executeCommand(dodwan.pubsubService.getAsBytes(desc.getKey());
}
```

Figure 11: API DoDWAN

any other attributes. The payload of a DoDWAN message is considered as raw data (bytes). Figure 11 shows two examples of code desribing message publication, as well as subscription and message reception.

7.3.2 Point-to-point adressing with DoDWAN

In order to use the content-based dissemination mechanism provided by DoDWAN to implement the destination-based mechanism induced by the client/server communication scheme of HTTP or CoAP, one can simply include an attribute *destination* in the descriptor of the messages. This attribute will be set by the sender to the identity of the destination host (as found in the URL). The message can then be published as usual. The destination host will actually be the only one receiving the message as far as all the hosts subscribe to messages containing a *destination* attribute with a value equal to their own identity.

7.3.3 Message format

DoDWAN is considered as a transport protocol for HTTP or CoAP. The entire content of the HTTP or CoAP request or response will form the payload of the DoDWAN message. In addition, attributes may be added in the DoDWAN descriptor for storing HTTP headers or CoAP options in a more convenient way than in the payload, for logging purpose for example.

References

[1] Maël Auzias, Yves Mahéo, and Frédéric Raimbault. CoAP over BP for a Delay-Tolerant Internet of Things. In *International Conference on Future Internet of Things and Cloud (icfiotc), Proceedings of*

the 3rd. IEEE, 2015.

[2] Marco Conti, Silvia Giordano, Martin May, and Andrea Passarella. From Opportunistic Networks to Opportunistic Computing. *IEEE Communications Magazine*, 48(9):126–139, September 2010.

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

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

[5] Armel Esnault, Nicolas Le Sommer, and Frédéric Guidec. An Anycast Communication Model for Data Offloading in Intermittently-Connected Hybrid Networks. In *The 10th International Conference on Future Networks and Communications (FNC 2015) / The 12th International Conference on Mobile Systems and Pervasive Computing (MobiSPC 2015), ADDRESS = Belfort, France, PUBLISHER = Elsevier, SERIES = Procedia Computer Science, OPTVOLUME = , NUMBER = 56, PAGES = 59–66, YEAR = 2015, MONTH = Aug, DOI = 10.1016/j.procs.2015.07.184*.

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

[7] Stephen Farrell, Alex McMahon, Stefan Weber, Kerry Hartnett, Aidan Lynch, and Eoin Meehan. Report on DTN Applications During Arctic Summer 2010 Trial. In *1st International Workshop on Opportunistic and Delay/Disruption-Tolerant Networking*, October 2011.

[8] Roy Thomas Fielding. *Architectural Styles and the Design of Network-based Software Architectures*. PhD thesis, University of California, Irvine, 2000. AAI9980887.

[9] Yaron Y. Goland. Multicast and Unicast UDP HTTP Messages, Internet Draft, Internet Engineering Task Force, nov 1999.

[10] Md. Tarikul Islam, Anssi Turkulainen, Teemu Kärkkäinen, Mikko Pitkänen, and Jörg Ott. Practical Voice Communications in Challenged Networks. In *1st Extreme Workshop on Communications*, August 2009.

[11] Nicolas Le Sommer, Salma Ben Sassi, Frédéric Guidec, and Yves Mahéo. A Middleware Support for Location-Based Service Discovery and Invocation in Disconnected MANETs. *Studia Informatica Universalis*, 8(3):71–97, September 2010.

[12] 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.

[13] Nicolas Le Sommer, Romeo Said, and Yves Mahéo. A Proxy-based Model for Service Provision in Opportunistic Networks. In *6th International Workshop on Middleware for Pervasive and Ad-Hoc Computing - Middleware Conference*, pages 7–12, Louvain, Belgium, December 2008.

[14] Changling Liu and Jörg Kaiser. A Survey of Mobile Ad Hoc Network Routing Protocols. Technical report, University of Magdeburg, 2005.

[15] 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.

[16] 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.

[17] 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.

[18] Vinìcius F. S. Mota, Felipe D. Cunha, Daniel F. Macedo, José M. S. Nogueira, and Antonio A. F. Loureiro. Protocols, Mobility Models and Tools in Opportunistic Networks: A Survey. *Computer Communications*, March 2014.

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

[20] Costas Tziouvas, Lambros Lambrinos, and Chrysostomos Chrysostomou. A Delay Tolerant Platform for Voice Message Delivery. In *1st International Workshop on Opportunistic and Delay/Disruption-Tolerant Networking*, pages 1–5, 2011.

[21] 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.