A Disruption-Tolerant RESTful Support for the Web of Things

Nicolas Le Sommer, Lionel Touseau, Yves Mahéo, Maël Auzias, Frédéric Raimbault

A Disruption-Tolerant RESTful Support for the Web of Things

Nicolas Le Sommer, Lionel Touseau, Yves Mahéo, Maël Auzias, Frédéric Raimbault
IRISA, Université de Bretagne Sud, France
Email: Nicolas.Le-Sommer, Lionel.Touseau, Yves.Maheo, Mael.Auzias, Frederic.Raimbault@univ-ubs.fr

***Abstract*—The Web of Things (WoT) extends the Internet of Things (IoT) considering that each physical object can be accessed and controlled using Web-based languages and protocols. However, due to the mobility of physical objects and to the short radio range of the wireless interfaces they are equipped with, frequent and unpredictable connectivity disruptions may occur between the physical objects and the Web clients used to control and access these objects.**

This paper presents a disruption-tolerant RESTful support for the WoT, in which resources offered by physical objects are identified by URIs and accessed through stateless services. Service requests and responses are forwarded using the store-carry-and-forward principle. A complete service invocation model is provided, allowing to perform unicast, anycast, multicast and broadcast service invocations either using HTTP or CoAP, which makes it particularly suited for the WoT. This disruption-tolerant support is illustrated by a case study in the context of agricultural robotics.

1. Introduction

Everyday objects are getting more and more connected to the Internet, thus contributing to the emergence of the "Internet of Things". The next step, as mentioned by Guinard et al. in [7], is to use the World Wide Web and its associated standards and technologies as a platform to interconnect and to control smart objects (i.e., sensors, actuators, electronic appliances). The "Web of Things" (WoT) aims at doing for physical objects what the Web did for information resources, namely an identification of resources using Uniform Resource Identifiers (URIs), an access of resources through standard protocols, and the inclusion in resources of links to other resources in order to enable a scalable discovery of highly distributed resources. Physical objects are expected to embed a Web server (which can now run with a small memory footprint), or to be connected to a device that hosts one, in order to expose their functionalities as services. In this architecture, services are identified by URIs, and HTTP is not only considered as a communication protocol but as an application protocol, where services can be invoked using HTTP methods (GET, POST, etc.). In the WoT, the RESTful architecture is privileged for simplicity and scalability reasons. REST (REpresentational State Transfer) is an architectural client-server style for distributed hypermedia systems [6] that is mainly characterized by stateless servers with uniform interfaces and caching of responses.

Following this philosophy, project ASAWoO proposes a WoT architecture which introduces a new kind of software artifact called "avatar" to provide a virtual extension of physical objects in the Web [15]. Avatars are implemented using a distributed software platform that can be fully deployed on powerful objects, or distributed over resource-constrained objects and a cloud infrastructure. They 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. Avatars implement a set of features allowing them to perform dynamic context adaptations, to select, to deploy and to compose REST Web services based on semantic descriptions, to apply security and privacy policies, and to behave autonomously. They also provide a user-friendly representation of objects using WoT applications (WoTApps). This makes it possible to interact with them via Web browsers and thus explore the world of smart objects with its many relationships (via links to other related objects). Since embedded Web servers in an Internet of Things generally have fewer resources than Web clients such as browsers running on PCs or mobile phones, WoTApps are designed to execute partially within the client.

This paper focuses on one of the key parts of the ASAWoO platform, namely the module supporting the provision of REST services offered by the avatars. This module is named ADTRS, for ASAWoO Disruption-Tolerant RESTful Support, in the remainder of this paper. Avatar services are accessible not only via the traditional HTTP protocol but also via the recently standardized Constrained Application Protocol (CoAP [21]). As HTTP, CoAP is a Web transfer protocol based on a REST architecture, but it introduces a number of design choices that make it particularly suited for the constrained nodes and constrained networks found in the Internet of Things.

It is quite common that the physical objects involved in the WoT cannot be connected permanently, firstly for energy saving reasons but also because of mobility. In this paper, we consider mobile objects that communicate through short-range radio interfaces (Wi-Fi, Buetooth, ZigBee, etc.) with other objects and with potential Internet gateways. Hence, they are subjects to disconnections while moving. To cope with this issue, ADTRS makes the requests and responses of

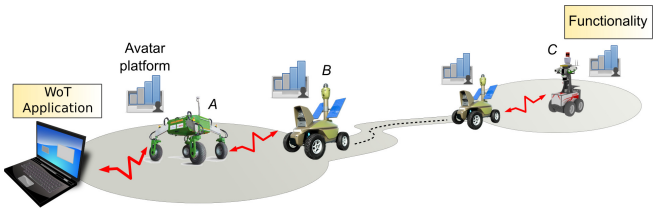

Figure 1. Example of traversal of a DTN involved in the call from a WoTApp to a Functionality

HTTP or CoAP tolerant to disruptions by transporting them using a DTN (Delay or Disruption-Tolerant Networking) protocol [5]. Indeed, this kind of protocol does not assume that there always exists an end-to-end path between two devices in the network. Mobile devices can store messages, carry these messages while moving and deliver them to other devices when possible. Figure 1 illustrates this principle: an external WoTApp (running for example in a CoAP-enabled browser) invokes an ASAWoO functionality exposed as a RESTful Web service by robot *C*. With only direct radio transmissions (that is to say device to device transmission, e.g. with Bluetooth), a delay-tolerant network is formed by the three robots *A*, *B* and *C*. The CoAP request first reaches robot *A* (assuming that this one was close to the PC that hosts the WoTApp when the request was issued). The request traverses the network, through robot *B*, eventually reaching robot *C*, thanks to the radio contact between robots A and B, the movement of robot B (acting as a data mule), and finally the encounter between robots B and C.

The remainder of this paper is organized as follows. Section 2 presents research works dealing with service provision in disruption-prone environments. Section 3 describes a case study we consider in project ASAWoO, and that we plan to use as a testbed to evaluate ADTRS. Section 4 presents ADTRS and illustrates its usage on the case study of Section 3. Section 4 also defines the spatial and temporal non-functional properties that can be exhibited by service providers and required by clients for the service provision process. Section 5 shows how ADTRS is implemented over existing disruption-tolerant communication middleware platforms. Section 6 concludes this paper by summarizing our contribution and by giving some perspectives.

2. Related Work

The provision of REST services in disruption-prone environments has been addressed in a few number of works, most of them dealing with opportunistic computing. Opportunistic computing is an extension of the concept of opportunistic networking [2], in which resources are exposed as application services. Services are discovered and invoked using opportunistic networking techniques. Like disruption-tolerant networking techniques, opportunistic networking techniques rely on the "store, carry and forward" principle. Opportunistic networking can be considered as a sub-category of disruption-tolerant networking, since opportunistic contacts between mobile devices are exploited to transfer data, and not only scheduled ones. Service invocation and service composition in opportunistic networks have been studied from a theoretical point of view in [3], [16]. In these works, the authors propose analytical models to determine the optimal number of parallel executions required to minimize the service time, without saturating the computational resources of the service providers, as well as to select the best service composition among alternative compositions based on the local knowledge of the network collected by a node through its opportunistic contacts with other nodes. In [14], the simultaneous invocation of services has also been investigated, but with a publish/subscribe and content-based approach. Service providers subscribe to service invocation requests containing the name or the description of the services they provide, process the requests, and return the responses to the clients. This approach allows to exploit the presence of several providers in the network, to reduce the service delivery delay, and thus to provide a better response time to end-users. The service invocation requests and service responses are transmitted using opportunistic communication techniques. To reduce the response time, intermediate nodes can be used as service proxies has shown in [12]. Intermediate nodes are expected to respond on behalf of service providers, as long as that these intermediate nodes have in their local cache the responses for the requests they receive and the services are stateless-services. Intermediate nodes do not forward the requests towards the providers, but send back the responses to the clients directly. This solution allows to reduce drastically the network load and the response time. Finally, discovery and invocation of location-aware services in opportunistic networks have been addressed in [11]. Based on their own location and on the location of service providers, clients can choose the closest provider, assumed to be the quickest provider to respond. The location information is also used to limit the propagation of messages in the network, and to forward the message towards providers and clients. All these research works address the service provision in general, but do not consider the constraints imposed by the delivery of REST services according to the RESTful architecture style. The purpose of the work presented in this paper is to propose a disruption-tolerant RESTful support to provide REST services in IoT environments composed of mobile devices.

3. Smart Vineyard Case Study

Agricultural robotics is an example of a challenging case study we consider in project ASAWoO. Agricultural robotics aims at providing to the agricultural industry the same productivity gains that robots give in manufacturing industries. The current trend is to shift from using big machines to smaller, lighter and more energy efficient unmanned machines, called agribots, that work as a team [17]. These changing agricultural practices, referred to as precision farming, have the same objectives of sustainable development that smart cities. Several precision agricultural

projects involving agribots and sensors are already underway[1], particularly in the viticulture field [18].

In the case study we consider, agribots are autonomous and modeled by avatars. These agribots are supervised and controlled through these logical artifacts, that are endowed with coordinations capacities, and that can collaborate to perform specific tasks. The mobility of these agribots induces frequent communication disruptions. The case study described thereafter is reused in the rest of the paper to illustrate the usage of ADTRS.

3.1. Precision Viticulture Scenario

The proposed scenario, depicted in Figure 2, takes place in a vineyard of several dozen hectares, far from any electrical infrastructure. It is assumed that mobile network technologies such as 3G/4G, or satellite solutions or other communication infrastructures, are not suited to be used by all devices in large areas during long periods of times; only a few number of devices should be connected to the Internet with 3G/4G, or occasionally with a Wi-Fi access point. In the studied scenario, such connected places are limited to the farmer's office and to the charging bases of agribots (places numbered 1 and 2 in Figure 2).

The farm size and its remoteness from urban areas justifies the use of DTN solutions: the systematic wiring is unrealistic and the lack of sources of stable electric power advocates energy-efficient solutions based on short-range communication.

3.2. Actors and Communication Paths

The actors involved in our scenario are the manager, manual workers, farm machinery (tractors, etc.), agribots, sensors and active RFID tags (referred to as class IV or V devices).

The manager can control all devices thanks to a Web client, from his PC when he stays in his office, or from his smartphone while moving, for example when he works in the vineyards. In this last case, he may access from time to time the application through a 3G/4G connection. The key point is to let the fleet of agribots under a lazy supervision, so that the manager can freely go about other tasks without having to monitor the agribots in real-time.

Agribots should work autonomously, once assigned a new task. For example, some agribots are in charge of weeding spaces nearby vine feet. At the same time other agribots perform background jobs, such as visually detecting infected leaf areas. Each agribot moves in a row, at its own speed, according to the conditions it encounters. When an agribot has finished a row, it passes in front of an active RFID tag (place 3 in Figure 2) that stores a piece of information coding the passage of the agribot, in order to notify any other robot that would try to redo the same row. Such an active RFID tag, serving as a row crossing, is positioned at one end of each row that the agribots must take to enter and to exit. Two agribots that pass within radio range of each other (crossing or progressing in nearby rows) can exchange information directly (place 4 in Figure 2), thus participating in the transmission of service requests and responses. When a robot finishes its work or when it lacks energy, it comes back to its charging base (place 2 in Figure 2), and thereby may also carry measurements provided by sensors in order to deliver them to the cloud infrastructure. Other data mules are farm vehicles (e.g., tractors) equipped with storage and short-range communication devices (place 5 in Figure 2).

Sensors monitor the essential agronomic parameters (soil composition, temperature, moisture, plant health, etc.) and transmit their measurements to data mules that pass nearby (place 6 in Figure 2). Finally data are delivered in the cloud infrastructure when a robot or a farm vehicle is connected to the Internet (places 1 and 2 in Figure 2). Note that sensors may be moved at will by manual workers; they are detected in an opportunistic manner when an agribot passes nearby, without the need to inform the agribots of their new locations.

3.3. Smart Vineyard Avatars

As mentioned in Section 1, an avatar provides a computational view of a physical object (e.g., sensor, RFID tag, agribot) or abstracts a set of avatars that collectively provide some services (e.g., the smart vineyard). Interactions with avatars are defined by the interface of the REST services they provide. An avatar can additionally provide users with a WoT application, to control a physical object or a set of avatars via a graphical interface. In our scenario, all the features needed by the vineyard manager are included in such a WoTApp, that represents the smart vineyard. This main WoTApp runs in the cloud. It translates the application-level operations (monitoring of vine's health parameters, weeding, phytosanitary treatment, irrigation, pruning, etc.) expressed by the manager using its smartphone or PC, in terms of requests to the avatars that model the other actors of the vineyard.

Each of the physical devices that needs to be managed or requested by the main WoTApp has to be represented by an avatar. To take into account the various computing and storage capabilities of the devices, we have to distinguish locally hosted avatars from remotely hosted avatars. We assume that agribots are able to host their own avatar, i.e., they can notably run a Web server that receives, treats and returns HTTP messages. Avatars deployed on agribots are also in charge of collecting data generated by the sensors distributed in the vineyard, and of interacting with the avatars of these sensors. The avatars of the sensors are not deployed on the sensors themselves, but in the cloud, because these ones have limited resources. Each of these avatars implements a service that returns the value collected by the remote sensor it represents. These services are expected to be invoked by the main WoTApp. As the sensors are not online, the returned value is the last known value collected by a robot and transported by a mule, and delivered by this one when it

1. See for example, European FP7 projects VINEBOT (http://vinbot.eu/) and VINEROBOT (http://www.vinerobot.eu).

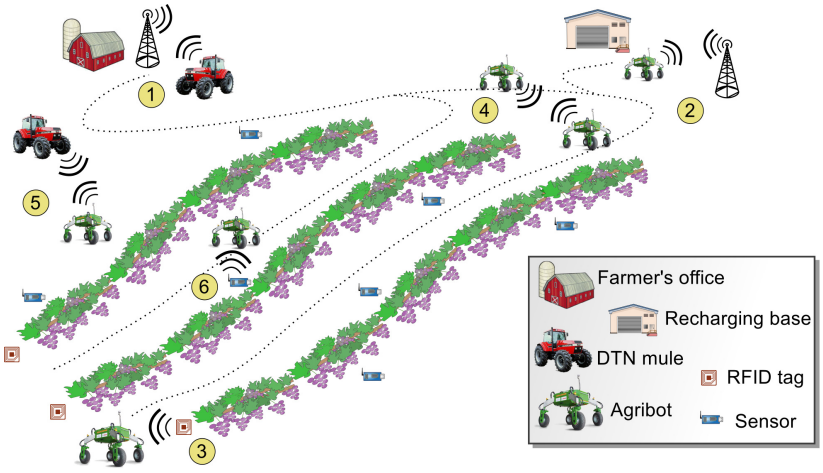

Figure 2. Precision viticulture in a Delay Tolerant Network environment.

is connected to the Internet. Thus, the gathering of sensors values is not modeled by an interaction between avatars, but it is exposed as an agribot's functionality.

Moreover, for simplicity reasons, it has been chosen not to associate avatars to the active RFID tags lying at the beginning of each vine row. An agribot can read or change the status of an RFID tag (which may be also changed by a worker to prevent robots from accessing the row as mentioned before). But the interaction is not managed by avatars because there is no direct application interest. Tags are only means to ensure exclusive access to the rows. Similarly, agricultural machinery is not modeled by avatars as it is involved only as data mules.

4. A Disruption-Tolerant Support for REST Services

In order to be compliant with the six constraints –the sixth one is optional– defined by the REST architecture style for performance, scalability and simplicity purposes, a disruption-tolerant communication support must implement the specific features that are described hereafter. ADTRS implements all of these ones. First, REST is based on a loosely-coupled *client/server* architecture where servers are accessed through *uniform interfaces* and do not maintain any session state (servers must be *stateless*). ADTRS meet these three constraints since avatars are designed as a set of distributed clients and stateless services, that interact through well defined interfaces (i.e., avatars functionalities). Since the communication path between a service provider and its clients may be subject to disruptions, maintaining a session with a client can become difficult for a provider. It is consequently preferable to maintain a state on the client side.

REST promotes a *layered system* as well as *cacheable* responses in order to improve overall scalability. Layers consist in intermediate servers in charge of non-functional concerns such as security, load balancing or shared caches provision. By implementing the "store, carry and forward" principle, ADTRS makes it possible to store service responses in the cache of clients and of intermediate nodes, but also the service requests that have been sent in the network. Following a proxy-based approach, intermediate hosts can respond on 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 data caching on intermediate hosts, and thus fulfills the two above-mentioned requirements.

In addition, WoTApps can be partially or fully developed in Javascript, thus allowing to execute on the client side a part of the application and to reduce the computation load on the physical objects, which complies with REST optional *code-on-demand* constraint.

The remainder of this section presents the communication models and the non-functional properties offered by ADTRS, and shows how REST services provided by avatars can be invoked using this RESTful support.

4.1. Communication models

Besides providing the traditional point-to-point client/server communication model used in the Web, ADTRS proposes alternative communication models, that are not necessarily relevant for traditional Web applications, but that are suited in the WoT, such as anycast, multicast and broadcast transmission models. Disruption-tolerant communication systems implementing a multiple copy forwarding strategy can take advantage of these message transmission models to increase the message delivery probability and to reduce the response time. Indeed, if it exists several providers offering the same service in the environment, these providers can indifferently be invoked with the same request. If an anycast communication model is used, only the first response received by the communication system will be returned to the client. Similarly, several sensors can simultaneously be invoked

using a multicast transmission model without naming them explicitly. All the responses returned by these sensors will be transmitted to the client.

In order to remain consistent with the RESTful approach, these different transmission models are specified in the scheme of the URIs used to access REST Web services. The general format of a URI is the following:

```
<service URI> ::= <scheme>"://"<destination>"/"<avatar name>"/"
                  <resource>["?"<parameters>]
```

The first part of the scheme indicates the application-level protocol (i.e., HTTP, HTTPS, or CoAP) used 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. +acast, +mcast and +bcast specify respectively that an anycast, a multicast and a broadcast transmission model must be used in the forwarding process.

Depending on the communication model, the destination part of the URI 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 *. 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. When an avatar is hosted by the physical object it represents, it is advisable to define as host alias the name of the avatar. The resource part identifies the resource that must be created, read, updated or deleted (CRUD operations). Additional parameters can also be specified in order to define non-functional properties for disruption-tolerant and opportunistic computing as presented in the sub-section below.

Table 1 gives examples of URIs that could be used in the smart vineyard scenario presented in Section 3. URI number 1 could be sent in unicast to order robot whose ID is agribot7 to go back to its recharging base after its current mission. Concretely, such an order is given by submitting a new job back_to_charging_station to the job manager of the avatar. This job manager is exposed as a Web service and is invoked via a POST CoAP method. Similarly, URI number 2 could be used to ask robots belonging to the agribots anycast group to weed around vine feet from rows 1 to 3 (additional parameters specified in the body of the POST request). Finally, URI number 3 allows to get the soil moisture from avatars of sensors monitoring the soil. As avatars of sensors are hosted in a cloud infrastructure, they do not need to be reached in a disruption-tolerant way.

4.2. Non-functional parameters for delay-tolerant computing

In delay-tolerant networks, service messages are forwarded following the "store, carry and forward" principle. The propagation delay and transmission area of messages is likely to be bounded not only to fulfill application requirements (e.g., a 2nd floor printer that can only be used by 2nd floor users), but also to reduce the network load. In the remainder of this section, we list a set of non-functional properties that can be expressed by service clients and service providers regarding the service delivery conditions on the one hand, and that can be exploited by disruption-tolerant communication systems in the message routing process on the other hand.

4.2.1. Caching parameter. Service clients can specify in their requests if these requests can be cached by intermediate nodes or not. 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 behalf of the service provider (i.e., by returning immediately the cached response, instead of forwarding the request towards the service provider).

For instance, environmental data such as air temperature at the vine feet, which is not likely to change frequently, can be retrieved by sending a GET request at the following URI:

```
coap+dtn+acast://agribots/tempsensors/
                    temperature?dtn_cacheable=true
```

4.2.2. Time parameters. Temporal constraints can be expressed in URIs as query strings or in the payload of application-level messages. Two temporal constraints are considered: the message creation time, and the message expiration time, expressed relatively to the creation time. The expiration time can also be specified as a symbolic value defined from a temporal social ontology (e.g., afternoon, evening, tomorrow).

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 forwarded 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, and the validity duration of the data it contains. These temporal constraints are also used by the disruption-tolerant communication systems to determine how long a message can be stored in a cache or forwarded in the network.

In the robotic viticulture case study, robots can be ordered to start weeding the vine feet of the first row by sending a POST request to the following URI:

```
coap+dtn+acast://agribots/JobManager/weed?row=1;
            dtn_ctime=1453913100000;dtn_etime=900000
```

This request expires after 15 minutes: considering the time required for weeding a row, passed this deadline it will be too late to start.

4.2.3. Spatial parameters. 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

	URI	Method	Parameters
1	coap+dtn://agribot7/JobManager/back_to_charging_station	POST	
2	coap+dtn+acast://agribots/JobManager/weed	POST	rows=1-3
3	http+mcast://soilsensors/moisture	GET	

Table 1. EXAMPLES OF URIS SUPPORTED BY ADTRS.

messages can be forwarded, and to avoid that a message eternally roams in the network. Nevertheless, this limitation is not exact, and does not guarantee that a message cannot be transfered outside an expected area. In order to limit more precisely the propagation of messages in the physical environment, geographical properties can be specified. Geographical areas are defined as squares or circles relatively to a given GPS position or identified by a symbolic name referencing a known area (e.g., home, office, myVineyard). The location of service providers can also be specified by clients in their service invocation requests as GPS coordinates or as an address whose GPS coordinates can be provided by a location service (e.g., OpenStreetMap, GoogleMaps). This location information can be used in the message forwarding process as shown in [9].

In the following example two vineyards are owned by the same wine grower. However the owner does not want the robots from one vineyard to interact with the robots operating on the other one. For instance, to restrict the communication to a circle defined by a GPS point matching the vineyard center and a 100 meter radius, one can configure the communication manager of the agribots –which is exposed as a Web service– with the dtn_area parameter.

```
coap+dtn+mcast://agribots/CommunicationManager/
                ?dtn_area=47.319, -0.535, 100
```

4.2.4. Asynchronous communication. Service clients can add in the query string part of the URI or as a path-parameter a parameter callback in order to define the URI 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.

In the vineyard case study, a sprinkler agribot that does not know the soil composition on the third row can ask other agribots to analyze it and send back the result. While waiting for the analysis, the agribot continues its current task (e.g., irrigating the first row). Once the soil composition analysis is received, the robot can process it and if the soil is dry, it will proceed to irrigate the row.

```
coap+dtn+mcast://agribots/JobManager/analyze_soil?row=3;
            callback=coap+dtn://agribot7/JobManager/sprinkle
```

5. Architecture and Implementation of the Disruption-Tolerant RESTful Support

5.1. Identification of Devices Hosting Avatars

Unlike Internet-legacy protocols, which rely on IP addresses, there is no standard regarding host identification and addressing in disruption-tolerant communication systems. Thus, we choose to identify devices hosting avatars by short string unique IDs in ADTRS. These IDs are mapped to the specific addresses of the underlying network layer (IP addresses or ad hoc addresses). A user-friendly name (i.e., an alias) can also be associated to these IDs. If a host can be accessed using both Internet-legacy protocols and a given disruption-tolerant communication system, the same alias should be assigned to the IP address of this 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 that is used.

For instance, an agricultural robot whose IP address is 192.168.1.7 aliased as cabernet (as defined in /etc/hosts file) would be identified by the name resolver of the DTN communication system as agribot7 (ID) also aliased as cabernet.

5.2. Overview of the architecture

ADTRS allows to invoke services using the HTTP and CoAP application-level protocols. The application-level messages (i.e., HTTP and CoAP messages) can be encapsulated in UDP datagrams, in TCP segments or in messages of a given disruption-tolerant communication system in order to be transmitted to their destination. Different wireless technologies (e.g., Bluetooth, Wi-Fi) can be employed to communicate with physical objects.

ADTRS is implemented by two main elements, namely an HTTP/CoAP proxy and a DTN adapter (see Figure 3). The HTTP/CoAP proxy makes it possible for standard HTTP and CoAP clients and servers to send and receive service requests and responses using disruption-tolerant communication techniques. To do so, clients specify in the URL of the proxy –that, in our current implementation, listens on a predefined communication port of the local host– a parameter dtn_uri containing a URI compliant with the syntax specified in the previous section. For instance, to simultaneously invoke from a standard HTTP Web browser several smart temperature sensors using the CoAP protocol, and to process the responses asynchronously, one can proceed as follows:

```
http://localhost:8080/?dtn_uri=coap+dtn+mcast://soilsensors/
  moisture?callback=coap+dtn://192.168.1.10/moisturecallback
```

Thanks to this proxy, programmers can develop HTTP and CoAP WoTApps using regular HTTP and CoAP, libraries. Moreover, standard HTTP and CoAP servers do not need to be modified. This proxy can also invoke remote REST services using Internet-legacy routing protocols (i.e., TCP/IP). The HTTP/CoAP proxy is a common element shared between all the implementations.

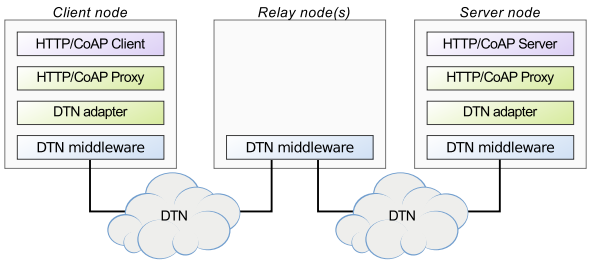

Figure 3. Overview of the implementation of ADTRS.

As for the DTN adapter, it binds the proxy and the disruption-communication system in charge of forwarding messages in the network. Hence, the DTN adapter depends of the underlying communication system and is specifically developed for each different system. The Bundle Protocol (BP) [20], which is the standard message-based protocol over the DTN architecture [1], has been chosen as the default disruption-communication system in our current implementation. Another implementation has also been done using an opportunistic communication platform we have developed, and which is called C3PO [10]. This platform provides interesting features to implement the anycast, multicast and broadcast communication models efficiently. For instance, service responses are used to stop the propagation of service requests in the network, and messages can be restricted to a given geographical area. Adapters for other disruption-tolerant communication systems could be developed in the future, such as for Haggle [19], DoDWAN [13], or the one presented in [8].

When the proxy receives a request, it forwards this one directly to the server if the networking environment is well connected. Otherwise, if the networking environment is challenging and intermittently connected, the proxy uses the DTN adapter to emit the request via the DTN communication system.

5.3. Implementation over IBR-DTN

BP defines a store-carry-and-forward overlay specifying the format of messages –called bundles– and the logic part to process them. BP is not bound to any routing algorithms though many exist, most of them offering to disseminate multiple copies of a bundle in the network. Bundles do not have a hop-count TTL, but a lifetime expressed in seconds. When its lifetime expires, the bundle, and its copies, are deleted from the cache of the devices. The applications running over BP –which are called endpoints– are identified by URIs. These URIs can either reference a single endpoint, or a set of endpoints that register themselves with a common URI. BP lies on a Convergence Layer (CL) to transport bundles (e.g., TCP, UDP, LTP).

In our current implementation, the IBR-DTN [4] has been chosen as the BP implementation. IBR-DTN is started as a daemon. This daemon is accessible via a TCP socket. Applications (BP endpoints) that need to communicate can use the daemon API through this socket. This implementation design makes it possible for constrained nodes (i.e., nodes that are not able to run their own BP daemon) to use a remote daemon, that can be likened to a DTN gateway.

The DTN adapter, communicating with IBR-DTN, acts as a BP application. It sends bundles to and receive bundles from the IBR-DTN daemon through the TCP socket, on behalf of the HTTP and CoAP clients and servers. It extracts from the HTTP and CoAP messages it receives from the proxy the pieces of information required to build bundles (destination, lifetime, etc.) and embeds these messages as payloads in bundles. Reversely, it extracts from the bundles the payloads and builds HTTP and CoAP messages that it transfers to the proxy. The IBR-DTN daemon ensures the delivery of the bundles based on the routing algorithm it relies on.

6. Conclusion

In this paper, we have presented a disruption-tolerant RESTful support for the Web of Things. This support is a key element of a distributed middleware platform defined in project ASAWoO, which proposes to extend physical objects in the Web with software artifact called “avatar”. Avatars allow to access and control physical objects using standard Web technologies and protocols. Avatars can be deployed on powerful physical objects, in a cloud infrastructure, or can be distributed over physical objects and a cloud infrastructure. The disruption-tolerant RESTful support we detailed in the paper permits to invoke the services offered by the avatars using HTTP and CoAP, despite the connectivity disruptions induced notably by the power saving strategies and of the mobility of physical objects equipped with short-range wireless interfaces. We have also shown in a viticulture testbed scenario how this support can be used. In a near future, the implementations of this support relying on IBR-DTN and on the C3PO opportunistic communication platform will be compared and tested on the viticulture scenario, both in simulation and on robots.

Acknowledgment

This work is supported by the French ANR (Agence Nationale de la Recherche) grant number ANR-13-INFR-012.

References

[1] Cerf, V.G., Burleigh, S.C., Durst, R.C., Fall, K., Hooke, A.J., Scott, K.L., Torgerson, L., Weiss, H.S.: Delay-Tolerant Networking Architecture. IETF RFC 4838 (Nov 2007)

[2] Conti, M., Giordano, S., May, M., Passarella, A.: From Opportunistic Networks to Opportunistic Computing. IEEE Communications Magazine 48(9), 126–139 (2010)

[3] Conti, M., Marzini, E., Mascitti, D., Passarella, A., Ricci, L.: Service Selection and Composition in Opportunistic Networks. In: 9th International Wireless Communications and Mobile Computing Conference (IWCMC 2013) . pp. 1565–1572. Cagliari, Italy (July 2013)

[4] Doering, M., Lahde, S., Morgenroth, J., Wolf, L.: IBR-DTN: an efficient implementation for embedded systems. In: 3rd ACM Workshop on Challenged Networks (CHANTS 2008). pp. 117–120. ACM (Sep 2008)

[5] Fall, K.: A Delay-Tolerant Network Architecture for Challenged Internets. In: Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications (SIGCOMM'03). pp. 27–34. ACM, Karlsruhe, Germany (Aug 2003)

[6] Fielding, R.T.: Architectural Styles and the Design of Network-based Software Architectures. Ph.D. thesis, University of California, Irvine (2000)

[7] Guinard, D., Trifa, V., Mattern, F., Wilde, E.: From the Internet of Things to the Web of Things: Resource-oriented Architecture and Best Practices. In: Architecting the Internet of Things, chap. 5, pp. 97–129. Springer (2011)

[8] Helgason, O., Yavuz, E., Kouyoumdjieva, S., Pajevic, L., Karlsson, G.: A Mobile Peer-to-Peer System for Opportunistic Content-Centric Networking. In: 2nd ACM SIGCOMM Workshop on Networking, Systems, and Applications on Mobile Handhelds (MobiHeld'10). pp. 21–26. ACM, New Delhi, India (Aug 2010)

[9] Le Sommer, N., Ben Sassi, S., Guidec, F., Mahéo, Y.: A Middleware Support for Location-Based Service Discovery and Invocation in Disconnected MANETs. Studia Informatica Universalis 8(3), 71–97 (Sep 2010)

[10] Le Sommer, N., Launay, P., Mahéo, Y.: A Framework for Opportunistic Networking in Spontaneous and Ephemeral Social Networks. In: 10th Workshop on Challenged Networks (CHANTS'2015). ACM, Paris, France (Sep 2015)

[11] Le Sommer, N., Mahéo, Y.: Location-Aware Routing for Service-Oriented Opportunistic Computing. International Journal on Advances in Networks and Services 5(3&4), 225–235 (2012)

[12] Le Sommer, N., Said, R., Mahéo, Y.: A Proxy-based Model for Service Provision in Opportunistic Networks. In: 6th International Workshop on Middleware for Pervasive and Ad-Hoc Computing (MPAC'08). ACM, Louvain, Belgium (Dec 2008)

[13] Mahéo, Y., Le Sommer, N., Launay, P., Guidec, F., Dragone, M.: Beyond Opportunistic Networking Protocols: a Disruption-Tolerant Application Suite for Disconnected MANETs. In: 4th Extreme Conference on Communication (ExtremeCom'12). pp. 1–6. ACM, Zürich, Switzerland (Mar 2012)

[14] Mahéo, Y., Said, R.: Service Invocation over Content-Based Communication in Disconnected Mobile Ad Hoc Networks. In: 24th International Conference on Advanced Information Networking and Applications (AINA'10). pp. 503–510. IEEE, Perth, Australia (Apr 2010)

[15] Mrissa, M., Médini, L., Jamont, J.P., Le Sommer, N., Laplace, J.: An Avatar Architecture for the Web of Things. IEEE Internet Computing 19(2), 30–38 (2015)

[16] Passarella, A., Kumar, M., Conti, M., Borgia, E.: Minimum-Delay Service Provisioning in Opportunistic Networks. IEEE Transactions on Parallel and Distributed Systems 22(8), 1267–1275 (2010)

[17] Redhead, F., Snow, S., Vyas, D., Bawden, O., Russell, R., Perez, T., Brereton, M.: Bringing the Farmer Perspective to Agricultural Robots. In: 33rd Annual ACM Conference on Human Factors in Computing Systems (CHI EA '15). pp. 1067–1072. ACM, Seoul, Republic of Korea (Apr 2015)

[18] Neves dos Santos, F., Sobreira, H., Campos, D., Morais, R., Moreira, A.P., Contente, O.: Towards a Reliable Monitoring Robot for Mountain Vineyards. In: International Conference on Autonomous Robot Systems and Competitions (ICARSC 2015). pp. 37–43. IEEE, Vila Real, Portugal (Apr 2015)

[19] Scott, J., Hui, P., Crowcroft, J., Diot, C.: Haggle: a Networking Architecture Designed Around Mobile Users. In: IFIP Conference on Wireless on Demand Network Systems and Services (WONS 2006). Les Ménuires, France (Jan 2006)

[20] Scott, K., Burleigh, S.: Bundle Protocol Specification. IETF RFC 5050 (Apr 2007)

[21] Shelby, Z., Hartke, K., Bormann, C.: Constrained Application Protocol (CoAP). IETF Internet Draft (Jun 2014)