Bringing Opportunistic Networking to Smartphones: a Pragmatic Approach

Frédéric Guidec, Yves Mahéo, Pascale Launay, Lionel Touseau, Camille Noûs

Bringing Opportunistic Networking to Smartphones: a Pragmatic Approach

Frédéric Guidec[1], Yves Mahéo[1], Pascale Launay[1], Lionel Touseau[1] and Camille Noûs[2]
[1] IRISA, Université Bretagne Sud, Vannes France. Email: name.surname@univ-ubs.fr
[2] Laboratoire Cogitamus. Email: name.surname@cogitamus.fr

***Abstract*—Opportunistic networking allows mobile devices to communicate without any fixed infrastructure, using the store-carry-and-forward principle, based on D2D (device to device) transmissions. Although smartphones may appear as perfect candidates to implement opportunistic networking protocols and algorithms, it turns out that their ability to support D2D transmissions is quite constrained, which hinders the deployment of opportunistic applications at a large scale. Acknowledging this fact we present *Ligo*, a device that is meant to behave as a peripheral device of a smartphone, providing this smartphone with the opportunistic networking services it cannot implement natively. The hardware components of *Ligo* are detailed in this paper, as well as the different software elements that enable the communication in the opportunistic network and the interaction with the smartphone. Experimental results are finally presented in order to validate our approach.**

***Index Terms*—Opportunistic networking, mobile networking, wireless networking**

:

I. Introduction

Opportunistic networks are a kind of challenged networks in which the mobility of devices and their sparse or unpredictive distribution hinder end to end connectivity. In an opportunistic network, mobile devices communicate directly with one another through short-distance radio links. Network-wide communication is enabled by applying the store-carry-and-forward principle borrowed to Delay-Tolerant Neworks [1]: a mobile device maintains a cache of messages, so that it can store messages while moving, in order to be able to forward them when an opportunity occurs, that is, when it enters in contact with some other device.

In the last two decades, research has been active in opportunistic networking. Focus has been mainly put on forwarding protocols that try to minimize the delivery delay while limiting the number of copies of messages and the volume of data exchanged during a possibly short contact [2], [3]. This has led to the definition of many protocols, each more or less adapted to the type of devices considered (hand-held devices, vehicles, sensors...), or their mobility, in particular when this mobility depends on social behaviors. Yet, most of the works on forwarding protocols have been validated only through simulation, and few effective opportunistic networking systems have been proposed and experimented in real conditions, especially at medium or large scale.

A number of application domains have been identified for opportunistic networking, though. Opportunistic networking is obviously a good option when the fixed infrastructure that supports traditional communication is not available, for instance in disaster relief scenarios or wildlife sensing. But other use cases are also envisaged, such as vehicular communication, or social networking in regions where the Internet is censored. One could expect that such use cases would have drawn the development and the deployment of opportunistic networking software. In addition, as smartphones are now pervasive, one could even think that mobile opportunistic networking applications would have been distributed at a large scale. But the reality is far from that. Even at the research level, only a few works reach the level of a proof of concepts, deployed on a bunch of devices, and the publicly available operational applications are almost nonexistent.

As stated in [4], the human factor is one of the main reasons for the lack of real experimentation: developing a fully operational protocol stack or a communication middleware is not an easy task, and conducting field experiments remains time-consuming. There is a need for some opportunistic middleware exposing a clear and simple API and some effective deployment tools. The second reason mentioned in [4] is the *technology hindrance,* and this reason is probably the most pregnant one: to date there is no radio technology suitable for opportunistic networking that is easily usable on off-the-shelf mobile devices. Indeed, opportunistic networking requires device discovery and, in the general case, should allow direct transmissions between a priori unknown devices without human intervention. Even though Wi-Fi and Bluetooth chipsets are common on handheld mobile devices, operating systems (essentially, Android and iOS) do not offer proper support to meet opportunistic networking requirements. Wi-Fi in ad hoc mode is not possible on non-rooted devices, Wi-Fi Direct and Bluetooth solicit the user for configuration and pairing acceptance. Other promising technologies such as 4G or 5G D2D are simply not available yet, and it is unclear if device-to-device communication will be widely enabled in operator-managed networks in the near future.

This paper presents a prototype architecture that is intented to ease the development and the deployment of opportunistic applications on handheld mobile devices such as smartphones. Having aknowledged the inadequation of such devices to opportunistic communication, we propose to assign this function to a separate device, built out of more controlable materials. This device, that we call *Ligo*, is small enough to be carried in a pocket. In this paper, we present a prototype of *Ligo* based

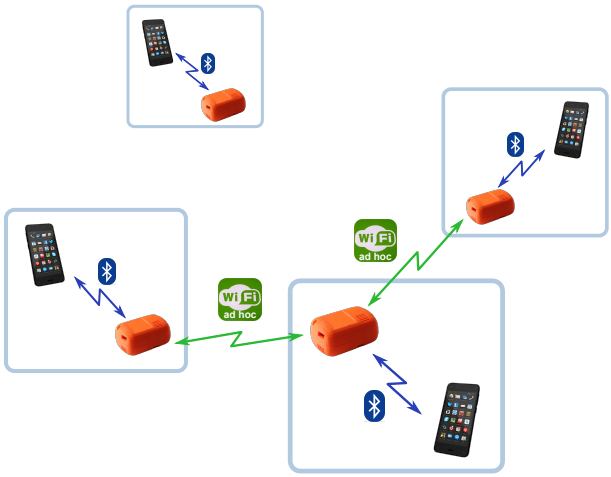

Figure 1. Global communication architecture

on a Raspberry Pi Zero, whose Wi-Fi interface is used in ad hoc mode so that several *Ligo* units can form an opportunistic network. Each *Ligo* unit is associated with a handheld device (for the sake of simplicity, we will assume in the rest of this paper that this device is a smartphone) via Bluetooth in order to interact with the user application. Figure 1 depicts the resulting global communication architecture, that involves smartphones and *Ligo* units.

Apart from the hardware design of *Ligo*, the main contributions detailed in this paper comprise a Bluetooth gateway application for connecting a smartphone to a *Ligo* unit, and a protocol added to the opportunistic communication middleware to enable exchanges with the external application hosted on the smartphone, which can be a native application or executed in a Web browser. Additionally, a solution for the deployment of user applications is presented, taking into account that the smartphones involved in the opportunistic network are not permanently connected to the Internet. The different hardware and software propositions are complemented by the description of a series of experiments that validate our approach.

II. Related Work

Deploying and testing opportunistic networks in real conditions and at a sufficiently large scale has been reckoned as a future direction to follow in opportunistic networking research, in order to go beyond the definition of forwarding protocols and their validation through simulation [5], [6]. Yet, it is admittedly a delicate task. One of the main problems is that although smartphones are often considered as the obvious deployment target for opportunistic networking protocols and applications, they can hardly support the kind of radio transmission required in an opportunistic network. Indeed, opportunistic networking requires spontaneous direct device-to-device (D2D) communication between network nodes. Yet the wireless interfaces available in smartphones are mostly adapted for infrastructure-based communication (i.e., via cellphone networks or Wi-Fi access points). Wi-Fi ad hoc communication would constitute a perfect means to achieve direct device to device transmissions among smartphones, but this mode of operation is disabled in standard releases of Android and iOS. This constraint can of course be removed by *rooting* (a.k.a. *jail-breaking*) the operating system of the smartphone as it has been done to test some opportunistic middleware [7], but in that case only a few people are willing to serve as testers for the proposed solution. In order to provide an alternative to real ad hoc communication among smartphones, many solutions have been proposed that rely on Bluetooth or Wi-Fi Direct [8], [9]. With both technologies, though, spontaneous interaction between smartphones is not possible, because both Android and iOS require user confirmation before two devices can connect together for the first time. This constraint prevents smartphones from interacting spontaneously with one another, and it constitutes an obstacle to large scale experimentation of opportunistic networking.

Since real ad hoc communication between smartphones cannot be obtained directly with the builtin hardware and OS, the approach presented in this paper consists in externalizing the missing features. Indeed, the idea of bringing additional transmission technologies to smartphones has already been considered in many research projects, and it has even materialized in a few commercial products.

goTenna proposes two kinds of devices (one for professional users, one for standard consumers) that can be paired with smartphones via BLE (Bluetooth Low Energy), bringing these smartphones the possibility to exchange short messages or GPS locations without relying on any fixed infrastructure (e.g., cellular or Wi-Fi) [10]. Transmissions are performed using GFSK modulation in either the UHF band (for both kinds of products) or the VHF band (for professional products), and mesh networking based on proprietary routing protocols allows messages to propagate over up to six hops [11], [12], [13]. goTenna devices do not rely on the store, carry and forward principle, though, so a device cannot store a message while its owner is moving, and forward this message later to other neighbor devices.

bearTooth is another device that works on a quite similar principle as goTenna [14]. It does not implement mesh networking, but PTT voice is supported in addition to text messaging and location sharing. Transmissions are achieved using FSK modulation (for PTT voice) and LoRa modulation (for short messages) in the ISM 915 band (902-928 MHz), so this device cannot currently be used out of ITU Region 2 (that is, mostly, the American continent).

Satellite communication can also be an option to provide communication services in unpopulated areas. A device such as Zoleo can be paired via Bluetooth with a smartphone, and then provide this smartphone with text messaging and location sharing services via the Iridium satellite constellation [15]. This approach requires paying a subscription fee, and it does not really qualify as an infrastructure-less approach.

The above-mentioned products all rely on the same basic idea: bringing new communication facilities to a smartphone by extending this smartphone with a peripheral device with

which it can interact via a Bluetooth link, and which will basically serve as a new external transmission interface for the smartphone. It is worth noting that none of these devices can support opportunistic networking, though, for none of them relies on the store, carry and forward principle. In the remainder of this paper we present *Ligo*, a device we designed specifically to provide smartphones with opportunistic networking services.

III. Ligo's hardware

A. Rationale

The *Ligo* device we based our approach on is built out of hardware components intented to support opportunistic communication and interaction with a smartphone. Before giving details on its hardware features, we summarize in the following the requirements we tried to fullfill when designing *Ligo*.

Radio capabilities: *Ligo* units must be able to form an opportunistic network by communicating directly with each other. Thus, the device must be equipped with radio technology allowing discovery and direct transmissions between devices, even never encountered before, without human intervention. A Wi-Fi interface operating in ad hoc mode is a perfect way to achieve device-to-device communications.

Moreover, *Ligo* and the smartphone must be associated and communicate via a short range radio link. Thus, *Ligo* must embed a Bluetooth interface, a technology widely available on smartphones.

Battery life: a smartphone is usually expected to run for at least 12 hours between two battery charges. Any device bringing opportunistic networking functionalities to such devices should have a similar battery life. In a smartphone or a tablet, though, the operating system implements aggressive strategies in order to preserve the battery. The builtin radios (i.e., Wi-Fi, Bluetooth, and cellular radios) are disabled most of the time, and the task scheduler itself freezes whenever possible, so the power consumption of the CPU is kept at a minimum. In contrast, the *Ligo* device must operate continuously in order to be able to detect neighbor devices, and be detected by them.

Storage capacity: *Ligo* must implement the store, carry and forward principle in order to serve as an effective carrier of messages. *Ligo* must therefore be able to store a large number of messages in a cache. The storage capacity is a major characteristic of a *Ligo* device. Besides it is preferable that messages be stored in a persistent storage space, so *Ligo* does not lose all the messages it was carrying every time it is shut down.

Data processing: *Ligo* must be able to interact with several neighbor *Ligo* units simultaneously, which requires multitasking.

Autonomous behavior: *Ligo* must be able to interact with a smartphone in order to receive commands from this smartphone, but it must also be able to run autonomously (interacting continuously with other *Ligo* units) when the smarpthone is in sleep mode or is shut down.

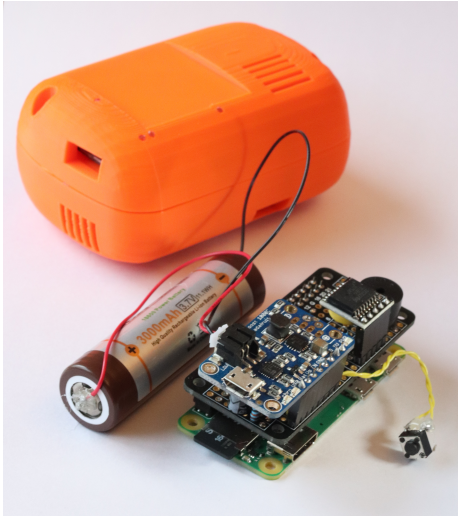

Figure 2. *Ligo*'s components and its casing

User interactions: Finally, *Ligo* must allow some basic interactions with its user, such as providing information on its operating status and battery level through visual or audible signals, and allow the user to turn it on or off.

B. Ligo's components

The *Ligo* prototype combines a Raspberry Pi Zero W with several additional components: a battery cell, a power controller, a real-time clock module, a buzzer, and a few passive components that make it possible to switch the device on and off. These components are assembled together thanks to a ProtoZero prototyping board (see Figure 2).

The Pi Zero W is a small single-board computer developed by the Raspberry Pi Foundation. It is based on Broadcom's single-core BCM2835 SoC, running at 1 GHz CPU clock speed. It features 256 MB onboard memory (RAM), 2 USB ports (one of which can only be used to power the device), and wireless capabilities (Wi-Fi and Bluetooth). A MicroSDHC card is used to store the operating system, and serve for persistent data storage. The BCM2835 also includes a GPU videocore, but this feature is not used in *Ligo* as this is a headless device.

The antenna used for Bluetooth and Wi-Fi communication on the Raspberry Pi Zero W is a resonant cavity antenna that is directly printed on the circuit board. There is no provision for connecting an external antenna, although this would certainly improve the performance of wireless transmissions. While preparing the first release of the *Ligo* prototype, though, we decided to keep it simple, and to do with the builtin antenna. Experiments conducted with the *Ligo* prototype confirm that, although the builtin antenna of the Pi Zero W is located close to the Li-Ion battery in *Ligo*'s casing, this does not seem to hinder wireless transmissions dramatically (see Section VII-B for details about these experiments). In any case, adding an external high-gain antenna to the *Ligo* prototype may be an

option for future releases, if only to improve significantly the radio range of this device when it is used outdoors.

The battery cell of the *Ligo* prototype is a 18650 rechargeable Li-Ion battery, with 3000 mAh nominal capacity. Other kinds of batteries could of course be used, in order to get a higher capacity (e.g., 26650 cells with 5000 mAh capacity), or a different form factor (e.g., Li-Po batteries).

The power controller is Adafruit's PowerBoost 1000C. This charger circuit is based on a DC/DC boost converter module that can be powered by any 3.7 V Li-Ion or Li-Po battery. It can provide the Pi Zero with a steady 5 V power input, while recharging the battery if necessary. *Ligo* can thus run equally on its battery or on an external power supply (provided via a USB plug), and it can switch from one power mode to the other without being interrupted. The power controller features four LEDs that indicate its current state. These LEDs are visible to the user of *Ligo*, and therefore indicate if *Ligo* is currently running, if the battery is being charged or is fully charged, and if the battery level is low. The power controller also features two control pins EN (enable) and LBO (low-battery output). In *Ligo*, pin EN is used to enable the power output (thus allowing the Pi Zero to boot), and pin LBO is connected to an input pin of the Pi Zero's GPIO port, so the Pi Zero can be notified when the battery level is getting low.

The real-time clock module is a small PCB (printed circuit board) based on a DS3231 RTC clock. This PCB includes its own cell-button battery, so the clock keeps running even when the other components of *Ligo* are off. The timing accuracy of the DS3231 clock is around ± 5ppm (± 0.432 sec / day). This clock communicates via I^2C, so its pins are connected to the pins devoted to I^2C communication on the Pi Zero's GPIO port. With the appropriate driver and configuration on the Pi Zero, the current time can be read from the DS3231 during the Pi Zero's boot sequence.

The buzzer is connected to a digital output pin of the GPIO port. It can thus be controlled by appropriate software (typically, a simple Python script). Currently, *Ligo* produces three short beeps after booting, and six short beeps before shutting down. Additional audio signals may of course be designed in order to notify the user of various other events.

Switching *Ligo* on and off is obtained by pressing an on/off button. A few passive components (a few resistors, a capacitor, and a diode) have been used to connect this button with the power controller and with the Pi Zero. As a result, when the button is pressed while *Ligo* is not running, this event triggers the EN pin of the power controller, so the Pi Zero is powered on and starts booting. When the button is pressed while *Ligo* is running, this event is detected by a Python script running on the Pi Zero, and this script triggers a shutdown of the Pi Zero. The same script also monitors continuously the state of the LBO (low battery) pin of the power controller, so a shutdown is triggered automatically when the battery gets too low.

Since the *Ligo* device is a prototype, a 3D printable casing has been designed for this device. With this casing, the USB port of the power controller is accessible so the battery can be charged when necessary. One USB port of the Pi Zero is accessible as well, so the Pi Zero can be connected to another computer, and get controlled (via an SSH session) from this computer. This feature is mostly used to update the software of the Pi Zero. In order to prevent any unintentional activation of the on/off button, this button is kept hidden inside *Ligo*'s casing, and it can only be accessed through a small hole. The user must thus use the tip of a pen, or a similar sharp object, to switch the device on or off. Light guides are embedded in the casing, so the four LEDs that indicate the status of the power controller are clearly visible from the outside.

IV. Bluetooth link between a smartphone and Ligo

Communication between a smartphone and *Ligo* relies on Bluetooth transmissions. An application we designed must first be installed on the smartphone, so it can communicate with *Ligo*. This application is named *Ligo Gateway*. To date it is only available for Android smartphones.

When *Ligo* boots, the *discoverable* and *pairable* modes are enabled for a limited time (by default: 2 minutes) on its Bluetooth interface. During that time, a user can use the *Ligo Gateway* app to pair her smartphone with *Ligo.* There is no need to repeat this procedure once the two devices have been paired, since they will both remember that they can trust each other.

Once a smartphone has been paired with *Ligo*, the *Ligo Gateway* app is meant to run in the background, as its main goal is to open and maintain an RFCOMM link between the smartphone and *Ligo*.

Ligo and the *Ligo Gateway* app support two methods for managing RFCOMM links. In both cases the Bluetooth gateway running on *Ligo* listens to a predefined RFCOMM channel C_0, waiting for a smartphone to connect to that channel. By default *Ligo* is meant to serve only one smartphone at a time, so when a smartphone connects to channel C_0 it is served directly on that channel. If *Ligo* must serve several smartphones simultaneously, then when a smartphone connects to channel C_0, *Ligo* selects another available RFCOMM channel C_i, and it notifies the smartphone that it should connect to C_i rather than monopolize C_0. Channel C_0 is thus used as a signalling channel, while actual data transfers are performed on other channels.

Note that the Service Discovery Protocol (SDP) of the Bluetooth standard is not used in the current version of *Ligo*, because it would not bring any real advantage in that case: since *Ligo* only provides a single kind of service over Bluetooth, there is no need for smartphones to enquire about the kinds of services it provides. Besides the *Ligo Gateway* app must be used to establish an RFCOMM link between both devices.

Once this RFCOMM link is established, it is used as a tunnel in which multiple TCP sessions can be multiplexed. Application programs running on both sides of the tunnel can then communicate together via TCP sessions, although all traffic is actually multiplexed on a single RFCOMM link (see Figure 3). If the RFCOMM link —and thus the tunnel it

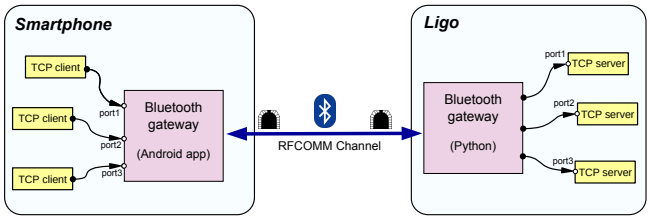

Figure 3. Bluetooth link between a smartphone and *Ligo*

supports— gets disrupted, all TCP sessions are automatically closed, and the *Ligo Gateway* app starts looking for *Ligo* again.

In practice, both the *Ligo Gateway* app and its Python counterpart on *Ligo* can be configured so as to listen to a list of specific TCP sockets. Let us assume that the *Ligo Gateway* app has been configured to listen to TCP port #8000 on the smartphone. If a local client program attempts to connect to this particular socket, then a notification is sent to the opposite gateway (running on *Ligo*), which in turn attempts to open a TCP session to socket #8000 on *Ligo*. If this attempt succeeds, then all traffic originating from the client program on the smartphone will be forwarded via the Bluetooth tunnel to the server program running on *Ligo*. Both gateways can be configured to listen to any number of TCP sockets.

V. Opportunistic communication

A. Device-to-device transmissions

Two major charateristics required from a mobile device in an opportunistic network are the ability to spontaneously discover its neighbors, and to engage in communication (i.e., exchange messages) with these neighbors. In both cases, direct device-to-device (D2D for short) communication on a wireless channel is required between the mobile devices. In *Ligo*, we chose to use the Wi-Fi interface for this purpose. The Wi-Fi standard can support D2D communication thanks to the so-called *ad hoc* mode of operation. Contrarily to the operating systems running on smartphones, the Linux Debian OS installed on *Ligo* imposes no restriction on the configuration of the Wi-Fi interface, and hence allows us to enable the ad hoc mode. An IPv6 address is forged and assigned to the Wi-Fi interface on each *Ligo* unit, using stateless autoconfiguration (SLAAC), so the opportunistic network is inherently scalable. Neighbor discovery relies on announcements sent periodically to a multicast group (with link local scope, so only one-hop neighbors can receive these announcements), and data exchanges between neighbor nodes rely on standard unicast transmissions.

B. Opportunistic middleware

Ensuring the forwarding of messages from node to node in an opportunistic network requires appropriate protocols. Although many message forwarding protocols for opportunistic networks have already been proposed in the literature [16], [5], [17], [2], [18], [19], most of these protocols have only been implemented as pseudo-code, and run with simulators. They can thus hardly be deployed on real platforms, and used in real conditions. Notable exceptions are IBR-DTN [20] and aDTN [21], which both support destination-based message forwarding based on the Bundle Protocol (BP) [22]. Another option is DoDWAN [23], which does not rely on BP as it is devoted to content-driven message dissemination, while BP is devoted to address-driven message forwarding.

The first release of *Ligo* includes the DoDWAN middleware. IBR-DTN and aDTN may be considered for inclusion in future releases.

When a radio contact is established between two instances of DoDWAN, they strive to exchange messages that match their respective interest profiles. Each message received by a DoDWAN node is stored in a local cache, so this message can be forwarded later to other nodes. This mode of operation is inspired from the Autonomous Gossiping (A/G) algorithm [24], which itself defines a selective version of the epidemic routing model proposed in [25].

DoDWAN is implemented in Java, and it is meant to support the content-driven dissemination of messages in an opportunistic network. In content-driven dissemination, the flow of messages is directed towards interested receivers rather than towards specifically set destinations. This mode of communication typically calls for a publish/subscribe (pub/sub) API. An application service that needs to send a message in the opportunistic network uses this API to publish the message, and all application services that have subscribed to receive this kind of message receice a copy of the message. Since DoDWAN supports content-driven dissemination, each message must be published with a descriptor that characterizes its content. Conversely, when an application service subscribes to receive some content, this subscription is characterized by a pattern that can be matched to the descriptor of any message. The patterns defined by all the application services running on a DoDWAN node define the interest profile of this node. The descriptor associated with a message must include an indication of this message's deadline. When this deadline is reached, the message is automatically expunged from the cache of any DoDWAN node. Assigning each message that propagates in an opportunistic network a set deadline is a common way of ensuring that this message will not remain forever in the network.

DoDWAN can support different kinds of wireless technologies and protocol stacks. In *Ligo* it relies on IPv6-based transmissions over a Wi-Fi channel. Neighbor discovery is therefore based on UDP annoucements sent to a multicast group, and two neighbor nodes can interact with each other via TCP sessions.

Each DoDWAN node periodically broadcasts an announcement (in a UDP datagram) in order to inform its neighbors about its presence on the wireless channel. By receiving similar announcements a host discovers its neighbors. Every time a DoDWAN node detects the presence of a new neighbor, a connection is established (via a TCP session) with this neighbor, and both nodes first exchange their interest profiles. Based on this information, each node can examine the descriptors

of the messages available in its cache, selecting among these messages those that could be of interest to the peer node. An offer is then sent to this peer, which can then request any of the proposed messages. This process goes on until the radio link between the two peer nodes is disrupted. Each DoDWAN node maintains soft-state information about the other nodes it has encountered recently, and about the messages it has already exchanged with each of theses nodes. Thus, when two nodes meet again, they can avoid offering each other messages they have already exchanged. In any case, a message is never sent to a node that has not requested it explicitly, so the wireless channel is not obstructed by redundant, and possibly useless message transfers. Of course a DoDWAN node may be connected to several other nodes simultaneously, so each node may actually serve as a conduit between two other nodes that cannot exchange messages directly.

The cache where DoDWAN stores messages is implemented in the host's filesystem, but a lookup table is maintained in memory so the descriptors of these messages can be parsed rapidly. Since the *Ligo* prototype uses a MicroSDHC card as a persistent filesystem, the number of messages that can be stored on *Ligo* is only limited by the capacity of this card. Although there is no limit to the size of the messages that can be exchanged between two DoDWAN nodes, using *Ligo* — and thus DoDWAN— to share video files may not be a wise choice. Exchanging text messages, images, and even small audio files is perfectly acceptable, though.

C. DoDWAN network API

The user application deployed on a smartphone must be able to interact with the DoDWAN instance running on the associated *Ligo* unit. For this purpose, DoDWAN has been endowed with a network API that proposes the main functions needed to develop an application that exploits opportunistic communication. This API covers the aspects related to neighbor discovery and to publication/subscription. The network API is presently implemented in two versions: a TCP version adapted to native applications, and a WebSocket version more useful for Web applications (that is, applications running in Web browsers).

The DoDWAN network API (NAPI for short) relies on a client-server asynchronous protocol. The DoDWAN instance hosted on *Ligo* runs a daemon to which clients can connect via one of the TCP connections multiplexed on the Bluetooth link. The protocol data units (PDU) exchanged between the client and the DoDWAN daemon are serialized. The serializisation method can be chosen among several ones (presently JSON or BSON). A PDU issued by a client is a command PDU (e.g., publish). It is expected that the daemon replies to a command with a response PDU. A token is included in the command and is copied in the corresponding response by the daemon to ensure command-response matching. A PDU issued by the DoDWAN daemon can be either a response to a command or a notification. There are two types of notifications: the notifications related to the changes of neighborhood (peers appearing and disappearing), and notifications related to the arrival of DoDWAN messages in the opportunisc network, resulting from subscriptions. PDUs are structured as maps, composed of key/value pairs that include a name and some parameters. Each value of a pair has a given type among the following seven types: String, 32 bit Integer, 64 bit Integer, Boolean, Binary Data, Map of strings, Array of strings. The following PDU (noted for clarity in pseudo-JSON) contains a command to publish a message, with a given descriptor and content.

```
{ "name":"publish",
  "tkn":"1234",
  "mid":"3f56fr67",
  "desc":{"topic":"general", "language":"en"},
  "data":...0x5 0x00...}
```

A client opens a NAPI connection to the daemon using the underlying transport protocol (WebSocket or TCP). The first NAPI PDU issued by the client should be the *hello* command in which the client passes its (unique) id to the daemon. The reception by the daemon of a *hello* command marks the beginning of a session. During a session, the notifications generated by DoDWAN are transmitted to the client that initiated the session, on the current underlying connection, if the session is active. A session is made active once the client has issued a *start* command. It is made inactive when the client has issued a *stop* command. Notifications generated by DoDWAN, when no connection is opened or when an opened connection is non active, are lost. However the client may later retrieve missed subscription notifications with the *get_matching* command. A connection may be closed for several reasons, unexpectedly or not. When closed, the connection is automatically considered inactive. The client is expected to retry to open the connection within the session, and if successful to first issue a hello command with the continuation parameter set. A session is ended explicitely by the client when this client issues a *bye* command.

On the daemon's side, the sessions are not persistent. This means that no session is considered open by the daemon when the daemon restarts after having been stopped (or after a crash). The client can be aware that the daemon has restarted when the daemon responds to a *hello* command (with the continuation parameter set) with an error.

Table I lists the main commands (for each command, a corresponding response, not shown in the table, is issued by the daemon) and notifications handled by the NAPI protocol.

VI. Application deployment

One of the main motivations for using *Ligo* for opportunistic computing is the ease of development and deployment of opportunistic applications.

The DoDWAN Network API is a simple API that allows the rapid development of pub/sub client applications to be deployed on smartphones. The programmer can chose to develop a native client (iOS or Android) or she can develop a client running in a Browser, that will connect to the DoDWAN

Commands		
name	***parameters***	***role***
hello	tkn, client, [cont]	starts a NAPI session
bye	tkn	ends a NAPI session
start	tkn	activates a NAPI session
stop	tkn	deactivate a NAPI session
publish	tkn, mid, desc, data | file, [expire]	publishes a message
add_sub	tkn, key, desc, [dir]	adds a subscription
remove_sub	tkn, subs	removes subscriptions
get_desc	tkn, msg_id	retrieves the descriptor of a message
get_payload	tkn , msg_id, [file]	retrieves the payload of a message
get_matching	tkn , subs	gets the ids of the messages matching some subscriptions
get_peers_id	tkn	gets the ids of the neighbor DoDWAN peers
get_my_id	tkn	gets the id of the local DoDWAN peer
Notifications		
on_desc_recvd	tkn, mid, key, desc, [ferror]	a message has been received (its descriptor is provided)
on_peer_added	pid	a new DoDWAN peer has come in contact
on_peer_removed	pid	a DoWDAN peer has lost contact

Table I
MAIN COMMANDS AND NOTIFICATIONS OF THE DODWAN NAPI PROTOCOL

daemon via a WebSocket. In both cases, the Bluetooth gateway will maintain the link between the smartphone and *Ligo*.

Developing a DoDWAN client as a Web application (a DoDWAN WebApp for short) is particulary interesting for portability reasons. Such an application is generally developed in HTML/CSS/JavaScript. A few simple Javascript functions can easily be built in order to use the DoDWAN Network API via a WebSocket. Of course, compared to native applications, Web applications may be restricted, in particular as far as the access to system functions is concerned. But for a number of applications, these restrictions are not a concern and can be simply circumvented. Morever, the deployment of Web applications is almost transparent for the user.

In the context of opportunistic networking, though, some constraints are to be considered as the smartphone may not be permanently connected to the Internet. In order to facilitate the deployment of DoDWAN WebApps, we have added to the *Ligo*'s software a Web server that plays the role of a WebApp repository (see Figure 4 for a global picture of the designed architecture). This repository is intended to store DoDWAN WebApps so that these applications can be used on demand, even when the smartphone is not connected to the Internet. The main entry point to this repository is a WebApp manager, accessible via HTTP on the smartphone (see Figure 5). The WebApp manager is used to download DoWDAN WebApps from the Internet when possible, to store them on *Ligo*'s SDcard for later use, and to present the list of available WebApps that can then be launched via a simple click. A REST API is also provided in order to access the main repository functions (list the WebApps ; add, remove, update a WebApp in the repository). A simple archive format (with filename extension .dwar) has been defined for packaging DoDWAN WebApps: the Web application tree is compressed, encoded in base64 and encapsulated in a JavaScript call, following the JSONP pattern[1]. Hence, despite same-origin restrictions, WebApps can be easily provided by any Web server on the Internet as .dwar files, without any further configuration. Finally, the HTTP server deployed on *Ligo* also provides WebApps with storage facilities via a REST API (as an equivalent to Cloud storage) to compensate for the lack of large storage capabilities for browser-embedded applications.

[1]JSONP has been chosen for its simplicity for our prototype. It should be replaced by CORS to increase security.

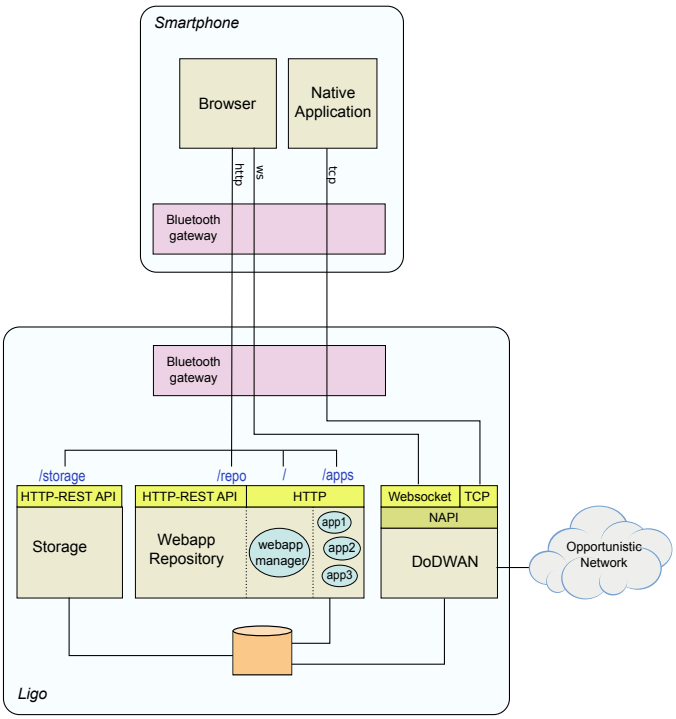

Figure 4. Deployment architecture on the smartphone and on *Ligo*

VII. EXPERIMENTAL RESULTS

A. Battery Life

We ran experiments in order to observe how long a *Ligo* unit can run with a 18650 battery cell. These experiments involved a fully functioning unit, meaning that both the

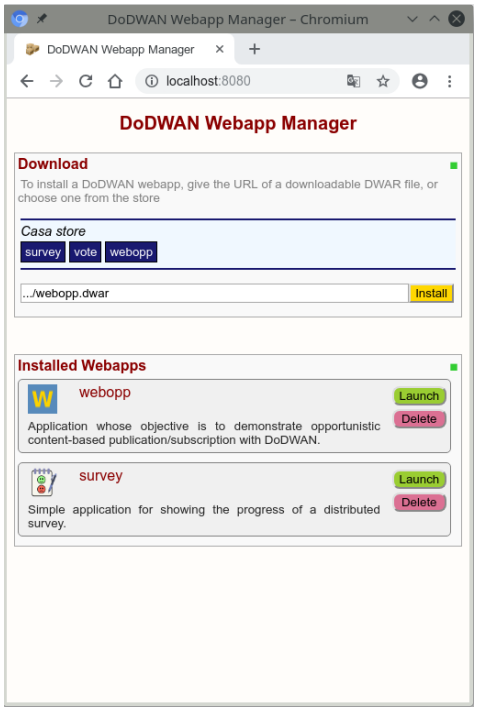

Figure 5. DoDWAN WebApp manager application

Wi-Fi and Bluetooth radios were enabled during the tests. Besides, the opportunistic networking middleware DoDWAN was also running during the tests, and the *Ligo* unit was paired and interacting actively with a smartphone. Each experiment started with a fully charged battery cell, and the *Ligo* unit was allowed to run until the low battery signal triggered an automatic shutdown of the unit. The uptime was later retrieved from the log files.

We observed that with a 3000 mAh battery cell the *Ligo* unit can operate for about 14 hours. This autonomy can be doubled by using a 26650 battery cell instead of a 18650 cell, but in that case the *Ligo* unit is of course heavier, and its casing bulkier.

B. Transmission range and bitrates

1) Bluetooth link: Experiments have confirmed that a Bluetooth link can be established between a smartphone and a *Ligo* unit over several meters, and that this link remains stable even when the user carrying the smarpthone moves around the *Ligo* unit. There is thus no need for the user to carry continuously the *Ligo* unit that is paired with her smartphone. As long as the smartphone and the *Ligo* unit are in the same room (or even in adjacent rooms), the Bluetooth link remains quite stable.

The BCM2835 SoC of the Pi Zero W implements the BLE 4.1 standard, which is retro-compatible with Bluetooth 2.1 + EDR (Extended Data Rate). Although the physical bit rate of EDR is 3 Mbps, we never observed application-level bitrates over 950 kbps when testing the Bluetooth link between a smartphone and a *Ligo* unit. To the best of our knowledge, the reason for this rather poor performance is that on a Pi Zero W, the Bluetooth transceiver is interfaced with the CPU via a UART. This UART constitutes a bottleneck, as it cannot stream data fast enough when a Bluetooth channel is used intensively. We confirmed this hypothesis by plugging a Bluetooth USB dongle into the *Ligo* unit, using this additional Bluetooth interface instead of the builtin one. In that case we observed application-level bitrates around 2 Mbps. The Bluetooth dongle option has not been retained in our prototype, though, because it tends to monopolize the only available USB port of the *Ligo* unit.

2) WiFi channel: Experiments conducted with several *Ligo* units deployed either indoor or outdoor have shown that two units can usually easily communicate (using Wi-Fi ad hoc transmissions) over dozens of meters in a single building, and that this distance can often exceed 100 meters in open space.

While conducting these experiments we have also measured the transmission bitrates that can be observed over a Wi-Fi channel. Since *Ligo* units communicate together in ad hoc mode in the 2.4 GHz band, the transmission bitrates at physical level are typically those allowed by the IEEE 802.11bgn standard. We measured the transmission bitrate at application level, using a home-made application that can either send or receive streams of data on a TCP session (without any access to the filesystem). While running this application on two *Ligo* units, we also monitored the traffic on the radio channel with a laptop running Wireshark. With the two *Ligo* units lying about two meters apart, we observed that the data frames transferred from the sending unit to the receiving unit were transmitted at 72 Mbps on the radio channel. Yet, at application level the transfer rate did not exceed 14 Mbps. When we moved one unit a few rooms away, or even to the next floor, the parameters of the OFDM modulation changed on the radio channel, so the transmission rate of data frames decreased accordingly. Yet the transfer rate at application level remained around 14 Mbps, and it only started to decrease when the distance between the two *Ligo* units exceeded 30 meters (indoor).

Thus it seems that although the builtin Wi-Fi transceiver of the BCM2835 SoC makes a decent use of the radio channel, application programs running on the Pi Zero W can hardly observe data transfers over 14 Mbps. This is probably due to the architecture of the Pi Zero W, but we are unable to pinpoint what specific part of this architecture explains this rather poor performance. Surprisingly enough, the application-level bitrate is far better (about 32 Mbps) when a Pi Zero W receives a stream of data from a laptop, rather than from another Pi Zero W. When the Pi Zero W sends a stream of data to a laptop, though, the bitrate is again around 14 Mbps.

C. Field experiment

We conducted a small-scale field experiment to observe how *Ligo* units paired with smartphones can perform in real conditions. This experiment lasted about 2 hours and it involved 15 volunteers, each volunteer using a *Ligo* unit together with her own Android smartphone. Before starting the experiment, each volunteer was required to install the *Ligo Gateway* app on her smartphone, and use this app to pair the

Metrics	***Values*** $^{(*=min/max/avg/stdev\ values)}$
Duration of the experiment	1h58'
Number of nodes	15
Field area	420 m × 160 m
Number of active nodes	5.0 / 15.0 / 14.2 / 1.8 $^{(*)}$
Activity duration per node	1h46' / 1h55' / 1h52' / 2'12" $^{(*)}$
Average number of neighbors per node	0.3 / 4.0 / 1.8 / 0.7 $^{(*)}$
Number of contacts	956
Durations of contacts	1" / 12'32" / 1'05" / 1'52" $^{(*)}$
Number of messages published	541
Number of messages deliveries	7513 (delivery ratio: 99.2%)
Messages delivery delays	0" / 20'23" / 3'01" / 2'25" $^{(*)}$

Table II
STATISTICS ABOUT THE FIELD EXPERIMENT

smartphone with the *Ligo* unit. During the experiment, the volunteers were asked to walk freely around our university campus, while exchanging text messages using a dedicated WebApp. Every message was sent on a public channel, so it could be received by all other volunteers. Log files were collected after the experiment and used to produce statistical data, which are presented in Table II. In this table the figures about the durations of radio contacts confirm the dynamics of the network, and those about the number of neighbors observed by each node are typical of a sparse distribution of nodes in the network. Of the 541 messages that have been published by the volunteers during the experiment, it can be observed that almost all messages have been delivered to all potential receivers. Some message deliveries occurred almost instantaneously (when the receivers happened to be direct neighbors of the senders), but the average delivery delay is around 3 minutes. These figures confirm the effect of the store-carry-and-forward principle enforced by the *Ligo* unit.

Overall this experiment performed in real conditions shows that our prototype is fully functional, and that it makes it possible for off-the-shelf smarpthones to engage in opportunistic communication.

VIII. CONCLUSION

In this paper, we have presented an architecture and described a prototype we have developped to enable opportunistic communication between handheld devices such as smartphones. Device-to-device communication is delegated to a peripheral device, called *Ligo*, that communicates with the smartphone through a Bluetooth link. The *Ligo* device feature a Wi-Fi interface operating in ad hoc mode, allowing discovery and direct communication between peers. The DoDWAN middleware runs on the *Ligo* unit in order to support opportunistic content-driven message dissemination. Experiments have been performed to measure the battery life, bitrates and transmission range of the prototype, and the whole system has been validated with a field experiment. The validation of this prototype shows the interest of our approach, which makes it possible to bring opportunistic communication to standard, off-the-shelf smartphones. Future work should notably include running further field experiments (if possible at a larger scale), considering alternative opportunistic networking middleware (such as aDTN or IBR-DTN) to be deployed in *Ligo*, and investigating other hardware platforms to bypass the inefficiencies of the Raspberry Pi Zero.

AKNOWLEDMENT

This work was supported by the French ANR (Agence Nationale de la Recherche) under grant number ANR-16-CE25-0005-02.t

REFERENCES

[1] K. Fall, "A Delay-Tolerant Network Architecture for Challenged Internets," in *Proceedings of ACM SIGCOMM03*, Karlsruhe, Germany, Aug. 2003, pp. 27–34.

[2] A. Triviño Cabrera and S. Cañadas Hurtado, "Survey on Opportunistic Routing in Multihop Wireless Networks," *International Journal of Communication Networks and Information Security (IJCNIS)*, vol. 3, no. 2, pp. 170–177, Aug. 2011.

[3] Z. Zhang, "Routing in intermittently connected mobile ad hoc networks and delay tolerant networks: overview and challenges," *IEEE Communications Surveys & Tutorials*, vol. 8, no. 1, pp. 24–37, 2006.

[4] M. Conti, C. Boldrini, S. S. Kanhere, E. Mingozzi, E. Pagani, P. M. Ruiz, and M. Younis, "From MANET to People-Centric Networking: Milestones and Open Research Challenges," *Computer Communications*, vol. 71, pp. 1–21, 2015.

[5] M. J. Khabbaz, C. M. Assi, and W. F. Fawaz, "Disruption-Tolerant Networking: A Comprehensive Survey on Recent Developments and Persisting Challenges," *IEEE Communications Surveys and Tutorials*, vol. 14, no. 2, pp. 607–640, 2012.

[6] S. Grasic and A. Lindgren, "An Analysis of Evaluation Practices for DTN Routing Protocols," in *Seventh ACM International Workshop on Challenged Networks (CHANTS'12)*. ACM, 2012, pp. 57–64.

[7] O. Helgason, S. T. Kouyoumdjieva, L. Pajević, E. A. Yavuz, and G. Karlsson, "A Middleware for Opportunistic Content Distribution," *Computer Networks*, vol. 107-2, pp. 178–193, Oct. 2016.

[8] V. Arnaboldi, M. Conti, and F. Delmastro, "CAMEO: a Novel Context-Aware Middleware for Opportunistic Mobile Social Networks," *Pervasive and Mobile Computing*, 2013.

[9] S. Trifunovic, M. Kurant, K. A. Hummel, and F. Legendre, "WLAN-Opp: Ad-hoc-less opportunistic networking on smartphones," *Ad Hoc Networks*, vol. 25, Part B, pp. 346–358, Feb. 2015, special isssue on New Research Challenges in Mobile, Opportunistic and Delay-Tolerant Networks.

[10] "goTenna," `https://gotenna.com`, december 2020.

[11] R. Ramanathan, C. Servaes, and W. Ramanathan, "ECHO: Efficient Zero-Control Network-Wide Broadcast for Mobile Multi-Hop Wireless Networks," in *IEEE Military Communications Conference (MILCOM 2018)*, 2018, pp. 1–6.

[12] A. Dusian, R. Ramanathan, W. Ramanathan, C. Servaes, and A. S. Sethi, "VINE: Zero-Control-Packet Routing for Ultra-Low-Capacity Mobile Ad Hoc Networks," in *IEEE Military Communications Conference (MILCOM 2019)*, 2019, pp. 521–526.

[13] R. Ramanathan, C. Servaes, W. Ramanathan, A. Dusia, and A. S. Sethi, "Long-Range Short-Burst Mobile Mesh Networking: Architecture and Evaluation," in *16th Annual IEEE International Conference on Sensing, Communication, and Networking (SECON 2019)*, 2019, pp. 1–2.

[14] "Beartooth," `https://beartooth.com`, december 2020.

[15] "Zoleo," `https://www.zoleo.com`, december 2020.

[16] C. Boldrini, M. Conti, and A. Passarella, "Autonomic Behaviour of Opportunistic Network Routing," *Inderscience International Journal of Autonomous and Adaptive Communications Systems*, vol. 1, no. 1, pp. 122–147, 2008.

[17] H. A. Nguyen and S. Giordano, "Routing in Opportunistic Networks," *International Journal of Ambient Computing and Intelligence (IJACI)*, vol. 1, no. 3, pp. 19–38, 2009.

[18] Z. Zhang, "Routing in Intermittently Connected Mobile Ad Hoc Networks and Delay Tolerant Networks: Overview and Challenges," *IEEE Communications Surveys and Tutorials*, vol. 8, no. 1, pp. 24–37, Jan. 2006.

[19] Z. Zhang and Q. Zhang, "Delay/disruption tolerant mobile ad hoc networks: latest developments," *Wireless Communications and Mobile Computing*, vol. 7, no. 10, pp. 1219–1232, May 2009.

[20] M. Doering, S. Lahde, J. Morgenroth, and L. Wolf, "IBR-DTN: an efficient implementation for embedded systems," in *Proceedings of the 4th ACM Workshop on Challenged Networks (CHANTS 2008)*. San Francisco, CA, USA: ACM, 2008, pp. 117–120.

[21] C. Borrego, S. Robles, A. Fabregues, and A. Sánchez-Carmona, "A Mobile Code Bundle Extension for Application-Defined Routing in Delay and Disruption Tolerant Networking," *Computer Networks*, vol. 87, pp. 59 – 77, 2015.

[22] K. Scott and S. Burleigh, "Bundle Protocol Specification," IETF RFC 5050, Nov. 2007.

[23] "DoDWAN: Document Dissemination in Wireless Ad hoc Networks," `https://casa-irisa.univ-ubs.fr/dodwan`, december 2020.

[24] A. Datta, S. Quarteroni, and K. Aberer, "Autonomous Gossiping: a Self-Organizing Epidemic Algorithm for Selective Information Dissemination in Mobile Ad-Hoc Networks," in *1st International Conference on Semantics of a Networked World (ICSNW'04)*, ser. LNCS, no. 3226, Paris, France, Jun. 2004, pp. 126–143.

[25] A. Vahdat and D. Becker, "Epidemic Routing for Partially Connected Ad Hoc Networks," Duke University, Durham, USA, Tech. Rep. CS-200006, Apr. 2000.