Integrating LoRa nodes into global DTN networks

Carlo Caini
DEI/ARCES
University of Bologna
Bologna, Italy
ORCID 0000-0002-6576-7320

Samo Grasic
IPNSIG

Alberto Soncini
University of Bologna
Bologna, Italy
ORCID 0009-0008-5366-195X

Abstract—Wireless sensor networks have always been considered an appealing application of the DTN architecture, because of their intermittent connectivity. They are however often based on microcontrollers, which complicates the use of the DTN pillar, the Bundle Protocol (BP). This paper shows how it is now possible to overcome this limitation and transform generic microcontrollers with a LoRa (Long Range) interface into true DTN nodes, fully integrated in a global DTN network. At the core of the paper we have muON, a new BPv7 implementation for microcontrollers, including two convergence layers, LoRaCL and UARTCL. The former (quite powerful and innovative) is used between LoRa nodes; the latter to interconnect a LoRa node with a gateway towards the IPNSIG PWG network, an experimental DTN network with nodes on many continents. Both CLs are described in detail in the paper. Test success proves that a LoRa network can not only benefit from BP-based DTN architecture, but thanks to BP interoperability it can also become a component of a global, potentially Interplanetary, DTN network.

Keywords— Delay-/Disruption-Tolerant Networking, Inter-Planetary Networking, Bundle Protocol, LoRa, Wireless Sensor networks

I. INTRODUCTION

Delay-/Disruption-Tolerant Networking (DTN) architecture [1], [2] derived from preliminary studies on Interplanetary Networking (IPN), when IPN pioneers realized that the challenges of space networks were partially in common with other "challenged" networks, such as wireless sensor networks, military tactical networks, underwater networks, emergency networks etc., and that a common solution was possible and preferable.

At the core of the DTN architecture we have the Bundle Protocol (BP), whose version 7 (BPv7), has been standardized by Internet Engineering Task Force (IETF) in RFC 9171 [3]. BP acts as on overlay on existing networks, and its packets, the "bundles", are transmitted end-to-end following a "store and forward" method, which allows bundles to be passed to the next DTN node on the path to destination only when the link is actually available. This property is of the greatest interest not only in space, but also in wireless sensor networks, where radio connectivity can be disrupted or limited by regulations, as in the LoRa (Long Range) nodes considered in this paper [4]. These nodes usually consist of a microcontroller board connected with a sensor or an actuator, and of a LoRa RF interface, which enables communication with other similar nodes. The problem is that BP implementations are usually designed for much more powerful platforms, such as ordinary PCs. In the literature there are examples of LoRa DTN networks which are, however, based on single board PCs, such as those of the Raspberry family [7], [8]. To transform a microcontroller with LoRa into a DTN node, an essential requirement is to build a BP implementation compatible with the limited resources of a microcontroller. The difficulty of this is most likely one of the main reasons for the limited adoption of DTN in wireless networks. However, current microcontrollers are much more powerful than their predecessors, thus the task is hard, but not impossible. A first attempt was made with early versions of µD3TN (ex µPCN) [5], [6], but at present its general compatibility with microcontrollers seems to have been abandoned to focus on POSIX platforms. In the most recent literature there is however a paper describing a BPv7 implementation for a specific microcontroller family (the powerful ESP32), also associated with LoRa RF interfaces [9], which confirms the fact that it is now actually possible to overcome the hardware limits of microcontrollers.

The aim of the present paper is to show how LoRa nodes based on generic microcontrollers can become DTN nodes (running BPv7), and how this network can be interfaced with a pre-existing ordinary network, to form a unique fully integrated global DTN network. At the core of the work there is the description of muON (microcontroller Overlay Network), the BPv7 implementation specifically designed by one of the authors for microcontrollers [10]. It is written in C++ and was first developed for the SAM D architecture and Arduino framework, but thanks to its modular design can easily be ported to other microcontroller platforms, which differentiates it from the implementation presented in [9].

For connectivity with other DTN nodes, muON has two convergence layer adapters, LoRaCL and UARTCL, described in detail in the paper. The former offers two services, unreliable and "notified", and allows bundles larger than the payload of a LoRa packet to be exchanged by implementing a segmentation mechanism modeled on TCPCL [11] and QUICCL [12]. The LoRaCL is also responsible of making bundle transmission compliant with LoRa limitations on Time on Air, which makes the LoRa channel intermittent. UARTCL is simpler but no less important, and is used to interface a LoRa DTN node with a non-LoRa node, essential to interconnecting the LoRa DTN network with an ordinary DTN network. To this end, UARTCL was also added to the CLs already provided by IONe [13], an experimental fork of ION [14] [15] maintained by Scott Burleigh, the DTN and IPN pioneer.

In the test section of the paper, the LoRa nodes are interfaced with the experimental DTN created by the IPNSIG (IPN Special Interest Group) PWG (Pilot Project Working Group) [16], managed by one of the authors and spanning the globe. The LoRa network is connected with the PWG one by means of a gateway running IONe and connected to other PWG nodes via TCPCL. So far, we have managed to send a Hello message from a LoRa node in Bologna to a bulletin board running on a remote PWG node in Sweden, but we would very much like to send it

to space nodes in a not-too-distant future, should PWG expand and escape Earth's boundaries.

II. DTN ARCHITECTURE AND BUNDLE PROTOCOL

The DTN architecture [2] is based on the insertion of the "Bundle layer", with its related protocol, the BP [3], on end points and some intermediate nodes, which become DTN nodes, as shown in Fig. 1. Between one DTN node and the next, i.e. inside a DTN hop, we can have non-DTN subnets (represented by networks A, B and C in the figure), whose presence is transparent to BP. BP is usually above a transport protocol, as in the figure, but in specific circumstances it can be inserted directly on top of a link protocol, as will be shown in this paper. In both cases, a slim protocol, called Convergence Layer Adapter, works as an interface between BP and the underlying protocol.

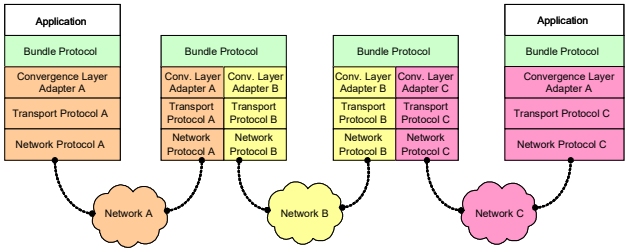

Fig. 1. DTN architecture and Bundle protocol stack

Two key aspects differentiate DTN from standard Internet architecture. First, as shown in the figure above, the semantics of the transport protocol are redefined as it is no longer end-to-end but, when present, operates within a single DTN hop. Second, the BP allows intermediate nodes to store data for extended periods of time. This latter feature is extremely useful in wireless sensor networks, where connectivity may be disrupted or strictly limited by regulation, such as in the case of LoRa. Thanks to this property, a bundle (the packet of BP) is first saved in a local database, then transmitted to the next node when possible. If immediate transmission is not possible, the bundle can even be stored for long periods (a few minutes, or even hours) waiting for an opportunity for transmission to the next node (a "contact", in DTN terminology).

Another important feature, which is however optional in the BPv7 standard, is that bundles can be retransmitted by a BP node when BP is notified of a failure by the convergence layer. Later sections will show how this feature is particularly well-suited for LoRa networks.

III. LORA

LoRa is a proprietary radio communication technology widely used in wireless sensor networks, thanks to its extended range and low power consumption [4]. It is essentially at the physical layer, and should not be confused with LoRaWAN (LoRa Wide Area Network), which is based on LoRa at the physical layer, but also includes link and network layers. In brief, a LoRaWAN node uses LoRa at the physical layer, but the use of LoRaWAN protocols is not compulsory for a LoRa node. The solutions proposed in this paper make no use of LoRaWAN.

LoRa is designed to allow the transmission of small amounts of data, organized in LoRa packets, between nodes located in the range of a few kilometers (from about 5 in urban areas, to about 20 in rural environments). It is based on the use of a variant of the chirp spread spectrum modulation, which, together with the use of FEC (Forward Error Correcting) codes, makes LoRa robust against interference. The achievable data rate depends on many factors, e.g. the Spreading Factor (SF), the Coding Rate (CR), the bandwidth (BW), etc., but is always quite modest (from about 0.25 to 27 kbit/s). The use of large SFs, small BW and small CR reduces the data rate but increases the range, thus the choice of these parameters is a trade-off between data rate and range, obviously crucial to performance. Very relevant for our application is the fact that the payload size of the LoRa packet varies with the data rate (the smaller the rate, the smaller the payload) and that is always quite modest, which makes encapsulating even a small bundle rather difficult. Later sections will explain how we managed to tackle this problem.

LoRa operates in a few unlicensed ISM (Industry, Scientific, Medical) bands, but a specific LoRa module is usually optimized for a single band, which can somewhat limit interoperability; we used modules for the EU868 band (863-870 MHz), which is one of the two bands commonly used in Europe.

To reduce interference, LoRa transmission is subject to several fair-access rules, which limit not only the maximum Tx power, but also the percentage of time a single node can use the channel (duty-cycle). Because of these regulatory limitations, LoRa connectivity is intrinsically intermittent, which is what makes the use of DTN technology on LoRa nodes appealing.

IV. SYSTEM MODEL

The application environment considered in this paper comprises two DTN networks: a LoRa-based DTN network consisting of LoRa DTN nodes, and a global DTN network composed of ordinary DTN nodes (the IPNSIG PWG network in tests). Two Gateway nodes allow bundles to be exchanged between the two networks (see Fig. 2).

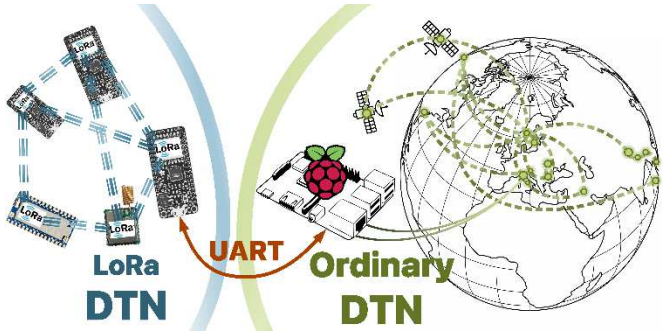

Fig. 2. The application environment considered in this paper. It consists of a LoRa based and an ordinary DTN network (global, but potentially interplanetary).

A. LoRa DTN nodes

The former network consists of LoRa DTN nodes (LoRa nodes in the following), each implemented on one microcontroller. Optionally, these nodes can interface with external peripherals: they may connect to a sensor (e.g., humidity or temperature) to act as a data collector, to an actuator

(e.g., to switch a pump on or off), or to both. If not connected, the unit acts as a pure relay node. All nodes must be interfaced to a persistent storage device and although technically optional the presence of a real-time clock (RTC) module is highly recommended. As this paper is concerned with network aspects, the kind of data being exchanged is irrelevant to our study, provided the amount of data exchanged is compatible with the limited bandwidth of LoRa.

In our tests, we use an "Adafruit Feather m0" board based on the "AT SAM D21 ARM Cortex-M0+" architecture, but the software we have developed can be easily adapted to other microcontroller families, thanks to its modular design. The microcontroller is connected to a LoRa module (in our tests, an RFM95W based on the SX1272 modem) and also features a USB serial port to allow connection to ordinary PCs.

The microcontroller runs muON, a "lightweight" BP implementation developed for the purpose by one of the authors. At present, muON does not yet implement all features specified by RFC9171, but rather a substantial subset, including bundle storage. DTN connectivity with other LoRa nodes is performed by means of a specifically designed LoRaCL, also part of muON.

B. Ordinary DTN nodes

The members of the standard DTN network have nothing special and may run any implementation of BPv7. To clarify: ordinary nodes do not possess any LoRa interface. In our tests, we interconnected the LoRa network with the experimental IPNSIG PWG network, a global DTN network aimed to expand the usage of the DTN Protocols, a pilot project of IPNSIG. At present, PWG nodes are based on ION [14], [15].

C. Gateways

The first gateway is a LoRa DTN node connected to the second gateway via UART over USB. To allow this, muON requires a second convergence layer, UARTCL, which we developed for this purpose. This gateway is connected through LoRaCL to other LoRa nodes.

The second gateway is a regular PC (in our tests a Raspberry Pi5) which in principle can run any BP implementation supporting the same UARTCL used by the LoRa gateway. In practice, as the development of the UARTCL is in a preliminary phase, it must use a special version of IONe [13] which includes a pilot implementation of UARTCL. This second gateway can be connected with ordinary DTN nodes with the usual CLs, such as TCP CL or UDP CL (in our tests we used TCP CL to connect it to the PWG network).

V. muON

The aim of muON is to enable DTN technology on microcontrollers with limited system resources. It is written in C++ and has been designed with a heavy emphasis on modularity, so that, although first developed for the SAM D architecture and Arduino framework, it may easily be ported to other architectures and frameworks that support the C++ language, such as STM32, ESP32, RP2XX0, etc. The constraints posed by the hardware limitations make it difficult to build a fully compliant BP implementation for microcontrollers, but this is also not strictly necessary to take advantage of DTN technology and to have a good level of interoperability with ordinary DTN nodes. The essential BP features already implemented in muON are: the ability to generate and read bundles fully conforming with the standard, which is essential for interoperability with ordinary nodes; the ability to store bundles, which is paramount when coping with LoRa intermittent connectivity; a convergence layer for LoRa and a convergence layer for UART. Furthermore, it also supports priorities (bulk, normal and expedited [2]), which match well with the limited transmission capabilities of LoRa nodes. To support priorities without the burden of an extension block entirely devoted to QoS, such as ECOS [17], the current version of muON uses the two bits of the primary block devoted to priority in BPv6, no longer used in BPv7. Future versions will aim at full compatibility with BPv7 standard, and may include ECOS for better compatibility with other nodes.

A. The LoRa Convergence Layer

The aim of the LoRaCL is to interface BP with LoRa by enabling the transmission of bundles on LoRa packets. The main issue is that the LoRa payload is both very small and also variable with the modulation settings (SF, CR, BW), which makes the obvious solution of limiting the bundle size to the LoRa payload very constraining. A better solution is to organize the transmission of one bundle, a "Bundle Transfer", in one or more XFER_MESSAGEs, each of which is then encapsulated into a LoRa packet, a mechanism borrowed from TCPCL [11] and QUICCL [12].

All LoRaCL messages begin with a polymorphic header structure spanning from 1 to 3 octets.

TABLE I: LoRaCL HEADER

Octet/Bit	0	1	2	3	4	5	6	7
0	MT		SC	Transfer ID				
1	Total Segments / Reason Code							
2	Segment Index							

The first octet serves as the Control Word, with the first two bits dictating Message Type (MT), followed by a single-bit Service Class (SC) flag selecting Unreliable or Notified service classes. The remaining 5 bits are allocated to a Transfer ID which serves to prevent cross-talk between concurrent Bundle Transfers. The semantics of the following octets is multiplexed based on MT. For XFER_SEGMENT messages Octets 1 and 2 define the fragmentation space, allowing for up to 256 Segments per bundle. For error signaling (bit 0 = 1) Octet 2 is omitted and Octet 1 is overloaded to carry the Reason Code.
For BDL_XFER_ACK both Octets 1 and 2 are omitted.
The four possible messages which LoRaCL can exchange are listed in Table II. Their names and functions (but not the format) are inspired to the analogous messages of QUICCL (in turn derived from those of TCPCL).

TABLE II: LoRaCL MESSAGES

MT	Name	Description
00	XFER_SEGMENT	Contains a segment of bundle data
01	BDL_XFER_ACK	Acknowledges a Bundle Transfer
10	XFER_REFUSE	Refuses a bundle transfer initiated by the peer.
11	MSJ_REJECT	Reports an unrecoverable error

- XFER_SEGMENT messages carry in their payload one chunk of bundle data; in the header they have a sequential Segment Index and a Total Segments field which allow the receiver to reassemble the original bundle and check if all segments have arrived.
- BDL_XFER_ACK messages have no payload and use the Transfer ID to identify the bundle transfer they acknowledge, which, as said, may consist of one or more segments.
- XFER_REFUSE is sent by the receiver to signal that the ongoing bundle transfer cannot succeed (e.g. because the bundle is too large for Rx buffers or because the reassembling timer has fired before receiving all expected XFER_SEGMENTS in the notified service).
- MSJ_REJECT is an error message the receiver sends in response to an unrecognized or unexpected message.

The Reason Codes for the error messages are summarized in Table III:

TABLE III: LORACL ERROR REASON CODES

Message Type	RC	Description
XFER_REFUSE	**0x01**	Insufficient space for announced size
	0x02	Duty Cycle Exhausted (for ACK)
	0x03	Administrative discard (e.g. not authorized or node in an unready state)
	0x04	NACK in the notified service
MSJ_REJECT	**0x10**	Missing Segment
	0x11	Unknown Transfer

Through these messages LoRaCL can provide both an unreliable and a notified service, as described below.

a) Unreliable service (SC = 0)

In this service class a bundle is sent through one or more XFER_SEGMENTs. On the sender node, as soon as the last segment is sent, LoRaCL notifies BP of a success, meaning only that the whole bundle was sent. On the receiver side, upon arrival of the first segment a reassembling timer is started: if all segments arrive before the timer expires, the original bundle is reassembled and passed to the local BP; otherwise, the segments are discarded.

b) Notified service (SC = 1)

In this class the arrival of the final segment triggers a positive or negative response. If all segments have arrived, i.e. there are no gaps, a BDL_XFER_ACK is sent back. At its arrival, the LoRaCL notifies BP of a success. In the unlucky case of some segments being missing at the receiver, or the if the reassembling timer expires before receiving the last segment, the receiver sends back a XFER_REFUSE, which serves as a NACK (Negative ACK) of the bundle transfer. At its reception, LoRaCL notifies BP of a failure. A sender side timer protects against the possible loss of either the BDL_XFER_ACK or the XFER_REFUSE: in the absence of any feedback, it fires and BP is notified a failure.

RFC 9171 leaves implementations free whether to resend the bundle or not in response to a failure declared by the CLA. In muON the bundle is resent up to a predefined number of retries, provided its lifetime has not expired.

Although the combination of notified service and BP retransmission is not equivalent to a reliable service, which would be hard to implement on a microcontroller, it ensures better chances of delivery than the unreliable service, especially when losses are moderate.

From the description above, it should be clear that for a given loss rate the chances of success decrease with the number of segments necessary to convey the original bundle. For this reason, LoRaCL allows the user to set a limit to the maximum number of segments of a bundle transfer. Although this is functionally equivalent to limiting the maximum bundle size, the good news is that this limit can now be much larger than the payload of a LoRa packet.

B. Management of LoRa link intermittency

As said, LoRa interfaces are subject to regulations which limit both transmit power and Time on Air (ToA). In our tests we used devices working in the EU868 band, which has a 1% Duty Cycle limitation to be calculated over a one-hour sliding window [19], [20], [21]. To comply with these limits, LoRaCL adopts a token bucket strategy by maintaining a ToA budget which is constantly replenished at a rate of 1ms every 100ms and can never exceed 36s (1/100 of hour).

Before each bundle transmission, LoRaCL calculates the expected transmission time. If this is larger than the available ToA budget, LoRaCL signals the temporary unfeasibility to the BP Agent, which can then decide how to proceed; for example, to postpone the transmission, or to ask the routing engine to look for a different proximate node.

Regulations also permit more lenient limitations for devices adopting a Polite Spectrum Access policy such as the Listen Before Talk (LBT) technique with random backoff and Clear Channel Assessment pause time. In practice, LoRaCL already uses LBT and random backoff, but since its features involving transmission time are still in a development phase, we chose to be conservative and temporarily abide by the standard 1% parameter.

C. The UART Convergence Layer

UARTCL has been designed as an asymmetric point-to-point link responsible for encapsulating a CBOR-encoded bundle in a byte stream, providing protection against corruption, and delivering it over a UART connection. In our scenario it is used only between the two Gateways; since these devices are running two different BP implementations (muON and IONe), this protocol must be implemented in both.

Unlike LoRaCL, UARTCL does not need to divide bundles into segments. With no regulations to abide by and thanks to the generous data rate of a serial connection, each bundle can be encapsulated in a single UARTCL Frame. Since bundles are encoded in a binary format, UARTCL must use an unambiguous packet framing algorithm. Consistent Overhead Byte Stuffing (COBS) [18] was selected for this purpose due to its proven reliability, efficiency, predictability and low computational requirements, all making it especially appealing for implementation on limited hardware.

UARTCL Frames are always strictly delimited by 0x00 bytes, giving both peers a robust resynch mechanism. Within these delimiters, the UARTCL Frame comprises a 1-octet Control Header, a variable-length CBOR payload (the bundle) and a CRC-16 checksum appended at the tail to ensure data integrity over the physical layer, calculated using the CCITT-FALSE algorithm, selected for its excellent performance on microcontroller hardware. Frames with mismatching CRC-16 are immediately dropped by the receiver. The Control Header structure is shown in Table IV.

TABLE IV: UARTCL CONTROL HEADER

Bit	0	1	2	3	4	5	6	7
Use	Version (= 0001)				CF	Reserved (= 000)		

The Control Flag (CF) bit differentiates between standard Data Frames (CF = 0) and technical Sync Frames (CF = 1).

In terms of service class, UARTCL is a best-effort delivery duct. It provides no ACK nor retransmission capabilities and only guarantees that corrupted frames will be discarded. The intentional omission of Link-Layer reliability prevents the architectural anti-pattern of nested retransmission loops. Given the parameters of a typical direct serial bus, we decided to delegate the eventual recovery operations to the overlying BP Agent. This allows UARTCL to remain lightweight, avoid state machine complexity and use processing power conservatively.

VI. TESTS

A. Testbed layout

As a proof of concept, we have built a testbed consisting essentially of three nodes, A, B and C, plus the IPNSIG PWG network. Node A and B are LoRa nodes (see Fig. 3); the former can be either the bundle source or the bundle destination, the latter is the LoRa Gateway. Node C, the Gateway to the IPNSIG PWG network, is implemented as a Raspberry Pi 5 running IONe (Fig. 4). A and B communicate via LoRa; B and C via UART over USB; C with the PWG nodes via TCPCL. The protocol stack of nodes A, B and C is shown in Fig. 5.

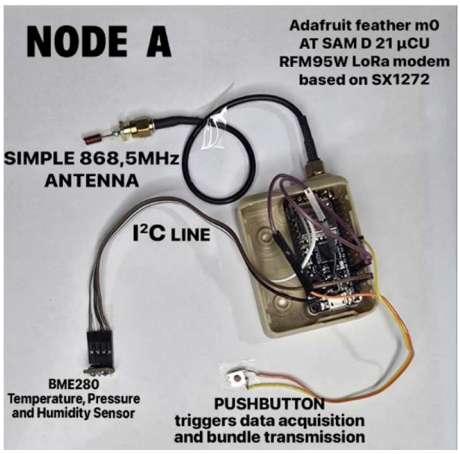

Fig. 3. Node A: it consists of an Adafruit microcontroller with an onboard LoRa module. Peripherals include a BME280 temperature, pressure and humidity sensor, a button to trigger sensor acquisition, a (not visible) RTC clock module and a SPI Flash memory chip for bundle storage.

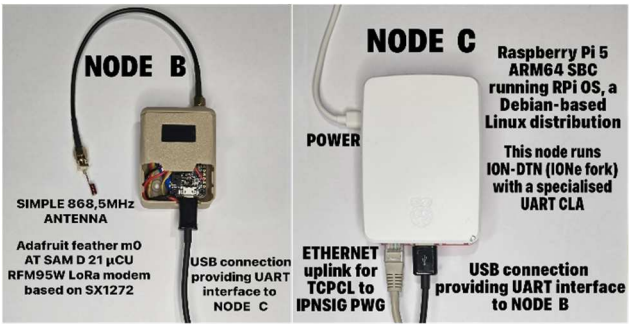

Fig. 4. Nodes B and C. B is the same as node A, but is connected via USB to node C, the Raspberry. This in turn is connected via Internet to the IPNSIG PWG network.

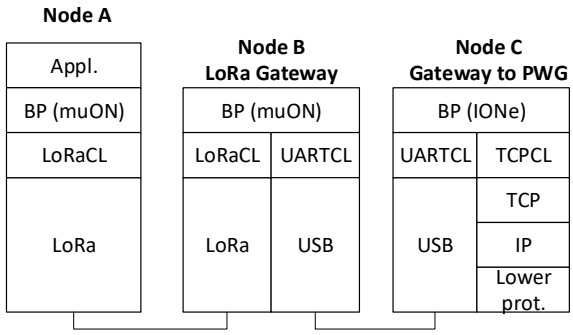

Fig. 5. The protocol stack used on nodes A, B and C. Node C is connected via Internet to the IPNSIG PWG network (not shown in the figure) and uses TCPCL as the convergence layer towards a node of this network. Note that nodes A and B lack both Transport and Networking protocols, as the LoRa network is not a TCP/IP network.

B. Preliminary tests

As preliminary tests, we used short messages generated on node A, encoded in JSON format, to be sent via DTN to a bulletin board application, running on a remote PWG node. An example is shown in Fig. 6. To test the reverse direction, we also sent a command to node A, encapsulated in a bundle, to turn on/off a LED on the microcontroller board. Other preliminary tests were performed to verify the ability of the LoRaCL to segment bundles larger than the LoRa payload.

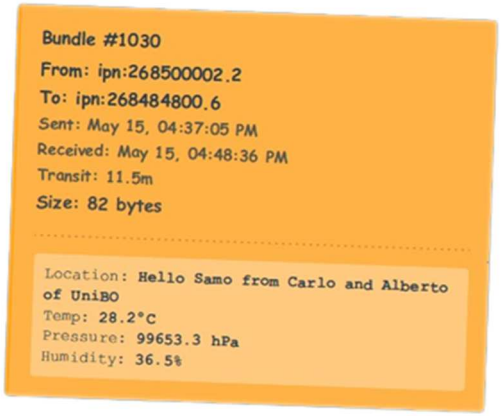

Fig. 6. The Hello message sent by node A via BP to the PWG bulletin board.

C. On the field tests

To prove the actual viability of our solution in real-world scenarios, we carried out a few field-tests. Node B was fitted to a custom-made quarter-wave monopole antenna with a conical ground plane designed to complement the wideband characteristics of LoRa and raised on a telescopic pole about 20 meters above ground; node A was mounted within a handheld housing fitted with a custom-made 6-element Yagi-Uda folded dipole antenna, a BME280 sensor and powered via Li-Ion battery for mobile testing. With this setup, special care was taken to configure the node LoRa modems to prevent exceeding the maximum ERP (Effective Radiated Power) imposed by regulations when paired with the antenna gains.

During the test we sent a bundle containing a JSON-encoded sensor reading from node A to node B; in response, node B sent to A one bundle containing RSSI (Received Signal Strength Indicator) and SNR (Signal to Noise Ratio) values as measured by its local LoRa modem. This procedure was repeated at increasing distances from node B in steps of 1 kilometer, until the link failed and no response was received. In an urban setting with no line of sight, significant obstruction of the first Fresnel zone and a conservative Tx power of 6 dBm the link endured up to 6 kilometers, after which the signal disappeared under the noise floor.

D. Future work

While the current tests demonstrate the feasibility of BPv7 delivery over LoRa, a comprehensive quantitative evaluation of the muON stack remains a priority for future work. Given our target hardware constraints, our next step is to profile the memory footprint to evaluate the effectiveness of our storage management strategy in minimizing RAM usage. Furthermore, although end-to-end transmission latency is inherently dominated by the LoRa physical layer and regulatory limitations, we plan to isolate and characterize the processing latency introduced by the DTN stack itself, which we expect to be minimal due to our selection of highly efficient algorithms. Finally, we will quantify the impact of LoRaCL segmentation both in terms of bandwidth and delay.

VII. Conclusions

This paper has shown how a network of microcontrollers with LoRa RF modules can become a DTN network and how this network can be successfully interfaced with a much larger DTN network, such as the IPNSIG PWG. The key to transforming microcontrollers into DTN nodes is muON, a lightweight BPv7 implementation specifically designed for this task. Its features are described in the paper, with focus on its two convergence layers. The former, LoRaCL, allows bundles to be transmitted between two LoRa nodes; the latter, UARTCL, is necessary to transfer bundles between a LoRa node and the PC acting as a gateway towards the ordinary DTN network. To make integration possible, UARTCL has also been added to IONe, an experimental version of the ION package. The work is completed by the description of a series of tests, which prove the feasibility, as well as the convenience, of integrating LoRa microcontrollers with a large global DTN network.

References

[1] Burleigh S, Hooke A, Torgerson L, et al. Delay-tolerant networking: an approach to interplanetary Internet. *IEEE Commun Mag*. 2003; 41(6): 128-136. https://doi.org/10.1109/mcom.2003.1204759

[2] V. Cerf , A. Hooke, L. Torgerson, R. Durst, K. Scott, K. Fall, H. Weiss "Delay-Tolerant Networking Architecture," Internet RFC 4838, Apr. 2007.

[3] S. Burleigh, K. Fall, E. Birrane, "Bundle Protocol Version 7", Internet RFC 9171, Jan. 2022.

[4] LoRa web site: https://www.etsi.org/deliver/etsi_en/300200_300299/30022002/03.03.01_60/en_30022002v030301p.pdf

[5] F. Walter, M. Feldmann, J. A. Fraire and S. Burleigh, "The Architectural Refinement of µD3TN: Toward a Software-Defined DTN Protocol Stack," 2024 IEEE 10th International Conference on Space Mission Challenges for Information Technology (SMC-IT), Mountain View, CA, USA, 2024, pp. 161-170, doi: 10.1109/SMC-IT61443.2024.00025.

[6] microD3TN code repository: https://gitlab.com/d3tn/ud3tn

[7] L. Baumgärtner, P. Lieser, J. Zobel, B. Bloessl, R. Steinmetz and M. Mezini, "LoRAgent: A DTN-based Location-aware Communication System using LoRa," *2020 IEEE Global Humanitarian Technology Conference (GHTC)*, Seattle, WA, USA, 2020, pp. 1-8, doi: 10.1109/GHTC46280.2020.9342886.

[8] D. Schmidt, F. Kuntke, M. Bauer and L. Baumgärtner, "BPoL: A Disruption-Tolerant LoRa Network for Disaster Communication," *2023 IEEE Global Humanitarian Technology Conference (GHTC)*, Radnor, PA, USA, 2023, pp. 440-447, doi: 10.1109/GHTC56179.2023.10354717.

[9] J. Zobel, T. Öhlenschläger and B. Scheuermann, "DTN7-ESP: a Native DTN-Based Communication System for ESP32 Microcontroller," *2025 IEEE Global Humanitarian Technology Conference (GHTC)*, Golden, CO, USA, 2025, pp. 01-08, doi: 10.1109/ GHTC66843.2025.11266441.

[10] muON-DTN code repository: https://github.com/albertos642/muON-DTN

[11] B. Sipos, M. Demmer, J. Ott, and S. Perreault, "Delay-Tolerant Networking TCP Convergence-Layer Protocol Version 4", RFC 9174, DOI 10.17487/RFC9174, January 2022, https://www.rfc-editor.org/info/rfc9174.

[12] C. Caini, T. de Cola and M. Moffa, "Delay-Tolerant Networking QUIC Convergence Layer Protocol Version 1", Internet Draft, Jan. 2026, work in progress, https://datatracker.ietf.org/doc/draft-caini-dtn-quiccl/

[13] IONe-uartcl-cobs code repository: https://github.com/albertos642/ione-uartcl-cobs

[14] S. Burleigh, "Interplanetary Overlay Network: An Implementation of the DTN Bundle Protocol,"in the Proc. of 4th IEEE Consumer Commun. and Networking Conference, 2007, pp. 222-226, doi: 10.1109/CCNC.2007.51. Code: https://github.com/nasa-jpl/ION-DTN

[15] ION code repository: https://github.com/nasa-jpl/ION-DTN

[16] IPNSIG PWG web site: https://www.ipnsig.org/ipnsig-pilotprojects-wg

[17] S. Burleigh, F. Templin, "Bundle Protocol Extended Class Of Service (ECOS)", Internet draft (expired), May 2021. https://datatracker.ietf.org/doc/ draft-burleigh-dtn-ecos/

[18] S. Cheshire and M. Baker, "Consistent overhead byte stuffing," in IEEE/ACM Transactions on Networking, vol. 7, no. 2, pp. 159-172, April 1999, doi: 10.1109/90.769765.

[19] European Parliament, Council of the European Union. Directive 2014/53/EU. 2014;L153:62-106. https://eur-lex.europa.eu/legal-content/EN/TXT/?uri=CELEX:32014L0053

[20] European Communications Committee (ECC). ERC Recommendation 70-03: Relating to the use of Short Range Devices (SRD). European Communications Office (ECO). 2022. https://docdb.cept.org/document/845

[21] European Telecommunications Standards Institute (ETSI). ETSI EN 300 220-1 V3.1.1 - Short Range Devices (SRD) operating in the frequency range 25 MHz to 1000 MHz; Part 1: Technical characteristics and methods of measurement. ETSI. 2017. https://www.etsi.org/deliver/etsi_en/300200_300299/30022001/03.01.01_60/en_30022001v030101p.pdf