Design and Features of Unibo-BP, the Unibo Implementation of the DTN Bundle Protocol

Carlo Caini, *Member, IEEE*, and Lorenzo Persampieri

***Abstract*—Challenged networks, including space networks, require the Delay-/ Disruption-Tolerant Networking architecture (DTN), which is based on the introduction of a new layer and a new associate protocol, the Bundle Protocol (BP). The recent release of RFC 9171, which standardizes version 7 (BPv7), has led the University of Bologna to develop its own implementation, Unibo-BP. The aim of this paper is to provide the reader with a comprehensive description of its design principles and innovative features. Unibo-BP is written in C++, is fully compliant with RFC 9171, is research-driven, and space-oriented, thus matching the main research interests of the authors. Unibo-BP is not a stand-alone application, but the core of a wide ecosystem that includes DTNsuite applications, LTP and TCPCLv3 convergence layers, and CGR/SABR routing. Unibo-BP interfaces to these additional components are thoroughly analyzed in the paper, as they present a number of advanced features. Unibo-BP supports one or multiple nodes on the same machine and a few template scripts to facilitate the user are described here. The paper also provides a section devoted to interoperability tests and first research applications An appendix, with an overview of Unibo-BP commands, completes this work. Unibo-BP is released as Open Source Software under GPLv3 license.**

***Index Terms*—Delay-/disruption-tolerant networking, interplanetary networking, bundle protocol.**

I. Introduction

THE DELAY-/DISRUPTION-TOLERANT Networking (DTN) architecture was conceived to enable communications in challenged networks [1]. These are networks where long delays, intermittent connectivity and other impairments prevent the use of ordinary Internet protocols. Challenged networks include some peculiar terrestrial networks, such as emergency networks, wireless sensor networks, tactical military networks and space networks. InterPlanetary Networking (IPN) research was actually at the origin of DTN architecture and space applications still play a prominent role in DTN research [2].

The main difference between the DTN and the Internet architecture is the presence in the former of an additional layer between Application and Transport, called Bundle layer. The associate Bundle Protocol (BP) is the pillar of the new architecture and was first standardized as an experimental protocol in 2007 by IRTF (Internet Research Task Force) in RFC 5050 [3], referring to BP version 6, or BPv6. After about a decade standardization moved from IRTF to IETF (Internet Engineering Task Force), and, eventually, in 2022 RFC 9171, describing BP version 7, or BPv7, was released [4]. In parallel the CCSDS (Consultative Committee for Space Data System) published its own specification for BPv6 [5] in 2015, basically adopting RFC 5050 with only two notable differences: the compulsory use of the "ipn" scheme for End Point Identifiers (EIDs) and the suggested use of the ECOS (Extended Class Of Service) extension [6], promoted by NASA. At present, work is in progress on a BPv7 update of the CCSDS BP.

The recent introduction of BPv7 and its planned use in future missions to the Moon has led to both the updating of existing BPv6 implementations to the new standard and to the birth of many others. Among the former we have ION, by NASA-JPL (Jet Propulsion Laboratory) [7], DTNME, by NASA-MSFC (Marshall Space Flight Centre) [8], both currently used between the International Space Station (ISS) and Earth, and EsaBP, by ESA [10]. Among the latter, let us cite μD3TN [11], by the D3TN company, which was used in space to prove the feasibility of the Ring Road concept [12], HDTN, by NASA-GRC (Glenn Research Center) [13] and DTN7-RS [14] by a group of German university researchers.

In this scenario, the University of Bologna has decided to develop its own BPv7 implementation, Unibo-BP, the subject of this paper, a greatly extended version of a preliminary paper presented at WiSEE-2023 [15]; although the topic is the same, the technical content has been considerably extended by the addition of some new sections and by totally revising all pre-existing sections (the main differences will be detailed later when dealing with organization).

Unibo-BP is written in C++ (with APIs in C) and it is fully compliant with RFC 9171, implementing all BPv7 compulsory bundle processing procedures and bundle extensions. Unibo-BP is research-driven, because its modular design allows the easy introduction of new features, and it is integrated into a wide ecosystem including experimental implementations of the LTP protocol [16], [17], [18] and of the CGR/SABR routing algorithm [19], [20] both essential in space networks. It is moreover compatible with Unified-API and thus with all DTNsuite applications [21].

Unibo-BP presents some advanced features, such as an innovative command line interface, the partial support of IRF routing [22], the option of enabling/disabling CGR/SABR extensions at run time, possible remote control of other nodes via commands encapsulated in bundles, the ability to launch multiple instances on the same machine, and others, all detailed in the paper. Unibo-BP comes

Manuscript received 20 December 2023; revised 17 January 2024; accepted 19 January 2024. Date of publication 24 January 2024; date of current version 24 May 2024. *(Corresponding author: Carlo Caini.)*

The authors are with DEI/ARCES, University of Bologna, 40125 Bologna, Italy (e-mail: carlo.caini@unibo.it; lorenzo.persampieri@studio.unibo.it).

Digital Object Identifier 10.1109/JRFID.2024.3358012

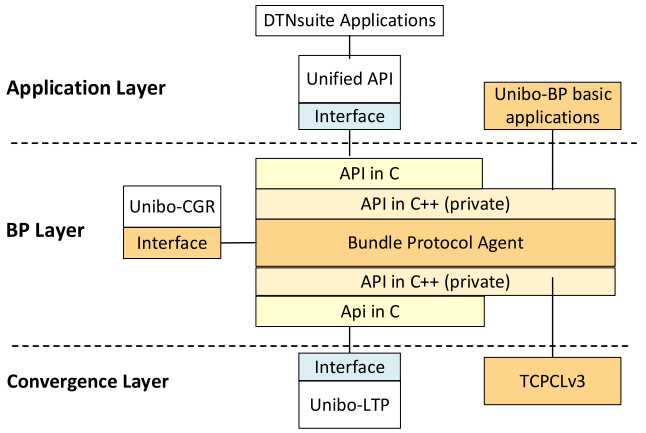


Fig. 1. The Unibo-BP ecosystem. The code of BPA, Unibo-BP basic applications, TCPCLv3 and Unibo-CGR interface is part of the Unibo-BP project; new interfaces to Unified-API and Unibo-LTP have been added to their respective projects.

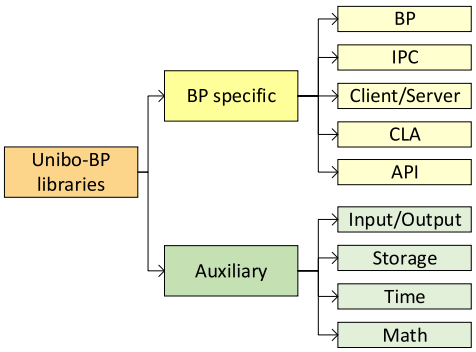


Fig. 2. The structure of Unibo-BP libraries.

with a new implementation of TCP Convergence Layer v3 (TCPCLv3) [23], essential in interoperability tests as it is a sort of "lingua franca", common to all implementations. Unibo-BP is released as free software under the GPLv3 license and can be downloaded from [24].

The paper is organized as follows (main differences with [15] are in parenthesis): an overview of Unibo-BP is presented in Section II, including details of its ecosystem, code organization, routing support and remote administration (greatly extended, with 3 new sub-sections); interfaces to existing modules, such as DTNsuite, Unibo-LTP and Unibo-CGR are discussed in Section III, (entirely new and technically detailed; it highlights the importance of these interfaces for current and future DTN research, with many novel technical remarks); suggestions as to the support of single or multiple instances are given in Section IV (entirely new, it describes how the user can easily carry out multiple-node tests on a single machine, a prominent feature of Unibo-BP); interoperability tests and first use of Unibo-BP in research are described in Section V (the first part has been totally revised, while the second, describing experiments carried out in collaboration with the German Aerospace Center, is entirely new); conclusions are drawn in Section VI. Finally, an overview of Unibo-BP commands is given in the Appendix (partially new; it is based on old material but significantly revised and with new syntax examples).

II. Unibo-BP Overview

A. The Unibo-BP Ecosystem

Unibo-BP is at the core of Unibo-DTN [25], a larger umbrella project including Unibo-CGR [26] and Unibo-LTP [27]. Unibo-BP requires Unibo-CGR code before compilation, while Unibo-LTP is an external, independent module. Unibo-BP can also be used by the DTNsuite project applications [28] through the Unified API [21]. As a result we have the wide ecosystem represented in Fig. 1, with Unibo-BP at the center, because it implements the Bundle Protocol Agent (BPA), i.e., the process that manages a DTN node. The BPA and Unibo-CGR are connected by an interface, used, for example, to pass contact and range information, stored in the BPA database, to Unibo-CGR, or, in the opposite direction, to obtain from Unibo-CGR the proximate node to which the bundle should be forwarded. The Unibo-BP project also includes a few basic applications, such as ping, echo, send and sink, and TCPCLv3. These internal modules, above and below the BPA, use the private APIs of Unibo-BP, in C++.

Below BPA we also have Unibo-LTP, a pre-existing project, originally designed for ION. Including it in the Unibo-BP ecosystem only required the design of a new specific interface. The same holds true for the Unified-API, above BPA. Their interfaces interact with BPA by means of the public API functions, which are written in C. Because of their significance, Unibo-BP interfaces to Unibo-CGR, Unibo-LTP and Unified-API will be described later, in full detail, in Section III.

B. Unibo-BP Code

Unibo-BP is written in C++ 20, an object-oriented language that in this very recent version offers a variety of innovative constructs and an augmented Standard Template Library. Most Unibo-BP classes are organized in libraries, which helps make the code structure clear and modular; as many classes are used by different processes, dynamic libraries are preferred to static to avoid duplication of binary code, heavier processes and a significant waste of memory. Unibo-BP libraries are divided into BP-specific and auxiliary, as shown Fig. 2, and will be briefly described below; see [29] for a comprehensive treatment.

1) BP-Specific Libraries: Starting from the top of Fig. 2, we have the BP library, the most important as it contains the classes that implement the data structures and the services of BPv7 [4]. It is worth stressing that Unibo-BP classes cover not only all compulsory features, but also a number of bundle extensions, including previous-hop, bundle-age, hop-count and also a BPv7 version of the ECOS extension [6] compatible with ION.

Second from the top we have the IPC (Inter Process Communication) library, which enables communications between the different processes of Unibo-BP. Its classes implement a client/server model based on Unix Domain Sockets with Stream type protocol. They were preferred to other IPC

methods, such as Shared Memory (used by ION) because fast and easy to manage.

Then, closely related to the IPC library, we have the Client and Server libraries. The former contains the functions used to send requests and receive responses by means of the IPC library. The latter, the dual functions, used to run servers within the BPA process.

The CLA (Convergence Layer Adapter) library consists of a server, destined to run on the BPA process, and a client, used to implement a specific Convergence Layer. This library also contains utility classes for both server and client.

The API (Application and Convergence Layer) library is the last BP-specific library. It contains Unibo-BP public functions, i.e., those that can be linked by external programs. These are built as C wrappers of internal C++ private methods, mainly offered by the "Client" and CLA library classes. In this way, external programs written in C, such as Unified API and Unibo-LTP, can easily interface with Unibo-BP, despite the latter being written in C++.

2) Auxiliary Libraries: The input/output library, "io", is designed on the model of the "java.io" package, familiar to most programmers. Its classes allow the programmer to abstract the input/output medium present at the lowest level (e.g., a file, a socket, a memory buffer). The "io" library also contains an implementation of CBOR [30], used in Unibo-BP not only for bundle encoding/decoding, as dictated by BPv7 specs, but also for Inter-Process Communication.

The "storage" library implements a file-based memory allocator, similar in use to the C "malloc" dynamic memory allocator. This library makes it possible to allocate "memory blocks" in a file, in order to save copies of volatile structures that must operate in memory for fast processing. At application restart these structures are repopulated by recalling the memory blocks saved. This library is used to store bundles in persistent memory, as well as for many other applications. The advantage with respect to using an external database, as in DTNME, is that this method is faster and that does not depend on an external code. On the other hand, although it normally preserves information between system reboots, it is less safe in case of abrupt termination, e.g., due to power failures.

The "time" library contains a few utility functions, such as converting time from UTC (in string format) to Unix Time (in C/C++ format). It also includes the "dtn_clock" class, which performs conversions between the DTN Epoch [4] and system times obtainable via the C++ system_clock class.

The "math" library contains various classes and utility functions, such as endianness converters, generators of unique, thread-safe, 64-bit identifiers, calculators of Cyclic Redundancy Check bytes, etc.

C. CGR/SABR, IRF and Static Routing

Routing support is an essential feature of any BP implementation. In DTN there is no single solution, and very different approaches are possible, such as replication schemes and forwarding schemes, depending on the kind of node connectivity (random, scheduled, etc.). Unibo-BP is concerned with space and supports CGR/SABR [19], [20] by NASA-JPL, which is the foremost solution for this environment.

Although in a network with few nodes CGR/SABR works well, it does not scale well to large networks. Bearing in mind future space networks consisting of hundreds or thousands of nodes, the author of CGR/SABR himself has proposed a hierarchical routing structure based on "regions". Inter-regional routing, alias inter-regional forwarding (IRF), would require a new routing algorithm [22], while intra-regional routing (within one region) would be left to CGR/SABR. IRF specifications have not been defined yet, but the latest ION releases already include an experimental version and the first papers dealing with it have appeared in the literature [31]. Although Unibo-BP does not yet include IRF, it can be considered IRF-ready, as it already supports the region concept and the consequent distinction between intra- and inter-regional routing. Moreover, and quite significantly, Unibo-CGR has been updated to manage two (or more) distinct contact plans, rather than only one. This feature is necessary for IRF "passageways", which must compute CGR for both their "home" and "outer" regions with different nodes and thus different contact plans.

Unibo-BP also supports static routing, which is still useful in small networks. Unlike in ION, in Unibo-BP static rules prevail over CGR/SABR, if present, essentially to allow the network administrator an easy way to force a route, when necessary.

D. Remote Administration

In Unibo-BP, as in all other BP implementations, once the local BPA process starts, it is possible to send it commands from the local node. Some implementations, such as DTNME [8] and μD3TN [10], allow the user to send commands also to an external BPA, by means of a TCP connection. This form of remote administration, however, is possible only when both the controller and controlee nodes are part of the same well-connected portion of the network, where TCP can operate effectively. To overcome this limitation, Unibo-BP allows commands to be sent as bundles from the controller to the controlee nodes, thus extending this form of remote control to the general case of challenged networks.

The procedure is simple: on the controllee a dedicated server must be launched (see Section VI-A5); then on the controller the command to be executed on the remote node is first written on a file, then inserted into a bundle and finally sent to the controlee; on arrival, the payload is extracted by the server and the command executed. In real deployment there would be security issues that would require the use of BPsec (Bundle Protocol Security) [32]. As this protocol has not yet been implemented in Unibo-BP, this form of remote control should be temporarily limited to research testbeds.

E. Commands and Command Line Interfaces

One peculiar feature of Unibo-BP is that it does not rely on configuration files, but is entirely managed by means of a few programs installed on the user's machine. Each of these has a command-line interface (CLI) that can be interpreted

by a command-line shell, such as "bash" (Bourne Again Shell). Users can either enter commands from the terminal or include them in a script file. In the former case, they can use shell shortcuts, such as autocompletion (very convenient as it provides syntax suggestions). In the latter, the advantage is that a script file is much more powerful than a list of static configuration lines.

A complete description of Unibo-BP commands is given in [29]; an overview is provided here in the Appendix.

III. Unibo-BP Interfaces

As we have seen (Fig. 2) Unibo-BP is at the core of a larger project, which includes three pre-existing modules, Unibo-CGR, Unibo-LTP and Unified-API. Unibo-BP interacts with these through three specific interfaces, whose description is important because it introduce new features, as detailed below.

A. Interface to Unibo-CGR

Unibo-CGR [26] was originally developed as an ION plug-in, which aimed to extend the original ION implementation of CGR/SABR with a few optional enhancements, such as "one-route-per neighbor", "queue delay" and two anti-loop mechanisms (proactive and reactive), all described in [20]. Shortly after, "Moderate Source Routing", an alternative technique to prevent loops based on the introduction of two bundle extension blocks, CGRR (CGR Route) and RGR (Record Geographical Route) [33], was also added. Thanks to these innovative features and to its very informative logs, Unibo-CGR was included in the official version of ION, where it can be selected at compilation time as an alternative to the original CGR implementation.

Thanks to its modular design, the Unibo-CGR core can be interfaced with other independent BP implementation, as was done by the authors with DTNME. With Unibo-BP, however, the connection is stronger, as Unibo-BP code must include Unibo-CGR (i.e., it cannot be compiled without it). From the user perspective, the main advantage of this integration is that not only contact plan elements, i.e., contacts and ranges, but also Unibo-CGR optional features are directly managed by the Unibo-BP command interface. In practice, in Unibo-BP any CGR enhancement can be enabled or disabled at run time, instead of at compilation time, as is necessary with ION and DTNME. The advantage is twofold: first, the researcher who wants to study these enhancements has no need to keep recompiling the code; second, and more important, this feature paves the way to the dynamic insertion of anti-loop countermeasures in real deployments. These countermeasures have pros and cons, and are obviously useless in the absence of loops, which, however, may happen in the most demanding scenarios [33]. Thanks to CGR integration, anti-loop features could be left disabled until loops are actually detected. More generally, anti-loop features could be dynamically enabled/disabled, depending on real operational necessity. Moreover, anti-loop insertion could be gradual, from the mild "reactive anti loop" mechanisms to the radical moderate source routing. The essential is to have some indicators of loop presence, and to this end there are various possibilities from the use of Bundle Status Reports (BSR) to track bundle routes to the use of the cited RGR extension. A promising technique, worth studying, could be to enable RGR at periodic short intervals, to check for loops without adding any significant impact on bundle processing on the average; should any loop be detected, immediate countermeasures could then be applied, thanks to the complete control offered by the Unibo-BP integrated interface to CGR.

B. Interface to Unibo-LTP

LTP was designed to cope with long delays and scheduled intermittent connectivity and is thus the "convergence layer" of choice in space networks, i.e., immediately below BP in the protocol stack. One or more bundles are encapsulated into an LTP block, which is then divided into usually much smaller LTP segments, in turn encapsulated into UDP (or other equivalent space protocol) datagrams. The transfer of an LTP block is called a "session".

Like Unibo-CGR, Unibo-LTP [27] was originally developed as an ION plug-in, to implement the experimental features of "Multicolor LTP" [34], which was jointly developed by the University of Bologna and DLR, the German Aerospace Center. Of particular significance to the Unibo-BP interface are the introduction in Multicolor LTP of the new "Orange" color and of session timers. "Orange" is added to the already existing "Red" and "Green", meaning reliable and unreliable service.

Following a universal praxis, Multicolor LTP assumes that a block can only be monochrome, i.e., that it may consist of either a Red or a Green part, but not of both. In this way, the service associated with the color applies to the bundle(s) encapsulated in the block. If the block is Red, the block (and thus the encapsulated bundle), is transferred reliably, i.e., lost segments are retransmitted; if Green, it is transmitted as best effort, without any segment retransmission. Orange provides an additional "notified" service: if an Orange block arrives without losses, a positive reception is notified to the LTP sender, as with Red; if with losses, however the block is dropped, and a failure instead of a success notification is sent back. In both cases at the arrival of Orange notification the LTP sender informs the BP, which, in case of failure, can decide whether to retransmit the bundle, as always when notified of a convergence layer failure; this choice is left to the implementation by RFC 9171 [4] but in fact both ION and Unibo-BP always retransmit. We refer the interested reader to [34] for an exhaustive explanation about the possible advantages of Orange, especially in long-delay high-bandwidth links (e.g., optical links in space).

The original ION implementation sends a bundle by default in a Red block, i.e., reliably, while Green blocks are used only when the "best effort" (alias "unreliable") flag of the ECOS extension is set (each bundle has its own ECOS settings, so consecutive bundles can alternatively be sent either as Red or as Green on the same link, depending on their flag). To let LTP set the desired color, the ION interface passes this "unreliable" flag from BP to LTP. Although simple, this can be interpreted as embryonic mapping between the QoS

TABLE I
MAPPING OF BUNDLE QOS TO LTP COLORS

"best effort" ECOS flag	"reliable" ECOS flag	Number of reTx	LTP Color
0	0		Default
1	0		Green
0	1		Red
1	1	0	Orange
1	1	>=1	Red

required at BP layer and that provided by LTP. The Unibo-BP interface builds on this concept by passing not only the "unreliable" flag, but also all other ECOS parameters (several flags plus other information, such as priority) [6], as well as the number of times a bundle has been retransmitted. By using this abundant information, Unibo-LTP can now make a more educated choice of LTP colors, i.e., a better QoS mapping. At present two ECOS flags, "best effort" and "reliable", are used together with the number of retransmissions to override the default LTP color, preset in the LTP configuration file. The exact QoS mapping, still under study, is given in the Table below.

The last two rows require a brief explanation. A bundle with both the ECOS flags set is sent the first time as Orange, but as Red if retransmitted by BP. The rationale is that we want to avoid a series of consecutive Orange session failures should the block be large and the loss rate high, because incomplete blocks are discarded with Orange. This is also a very straightforward way to implement the concept behind BSSP (Bundle Streaming Service Protocol), which does essentially the same thing, but in a much more complex way, as it lacks the notified service offered by Orange [34]. Lastly, note that LTP Orange with BP retransmitting bundles is intrinsically different from both the old (BPv6 only) "Custody option" [3] and the "aggregate status report" currently under study by CCSDS, as the first operates at the convergence layer, and thus necessarily on one DTN hop, i.e., between two consecutive nodes, while old and new custodial retransmissions operate at bundle layer, which therefore may happen on multiple hops or even end-to-end. Each solution has its pros and cons, which would however be too longthy to discuss here and beyond the scope of the present article.

The Unibo-BP interface also provides a simple solution to the following Red session problem: in space it may happen that the bundle contained in a Red block exceeds its lifetime (it becomes "stale"), before the session is completed, essentially because the retransmission of missing LTP segments adds one RTT (Round Trip Time) for each re-Tx cycle, up to 44 minutes on Earth to Mars links, to the expected bundle delivery time [34]. In such a situation, LTP Red would waste time and bandwidth resources by insisting on completing a block whose content (the bundle) is destined to be dropped as soon as received. The problem is solved by our interface, which passes the bundle lifetime to Unibo-LTP, to set the Red session lifetime (an extension of Multicolor-LTP) accordingly. This way, Red sessions can now be canceled as soon as their content becomes useless. With bundle aggregation (not implemented by Unibo-BP) [18], the solution would be less effective, but still possible: it would be enough to set the session timer to the maximum aggregate-bundle deadlines, so as not to cancel a session before all bundles aggregated into the same block have expired.

C. Interface to Unified-API

The Unified-API [21] is an intermediate abstraction layer that aims to decouple DTN applications from the APIs of specific BP implementations. It originates from the "abstraction layer" introduced with DTNperf_3, the third version of the DTNperf tool for DTN performance evaluation [35]. It was designed to allow the development of a single DTNperf version, compatible with both DTN2 (the precursor of DTNME) and ION, with obvious advantages in terms of code maintenance and consistency with respect to the easier alternative of developing two parallel versions. This concept was later extended to support other BP implementations, such as IBR-DTN [36] and μD3TN [10], and also other DTN applications. The combination of Unified-API and the DTN applications based on it forms the DTNsuite package [28].

The Unified-API consists of three layers (Fig. 3). The first contains the functions to be called by DTNsuite applications; the second layer is a sort of "clutch", designed to make the first layer independent of BP implementations; the third layer contains interfaces with specific APIs. This layered approach seems complex, but it allows support for new BP implementations to be added simply by writing a new interface at the third layer (plus minor changes at the second). This is exactly what was done for Unibo-BP, whose interface with Unified-API is given at the bottom right of Fig. 3.

With the interface to Unified-API, all DTNsuite applications (DTNperf, DTNbox, DTNchat, DTNproxy, DTNfog) can be used with Unibo-BP, without any change to their code. In particular, they can either be compiled for Unibo-BP alone, or for multiple BP implementations at the same time. With the latter option, the very same executable file can be used on top of whatever subset of the five BP implementations supported so far, which is very convenient when performing interoperability tests as those shown below, in Section V-B.

IV. RUNNING SINGLE AND MULTIPLE INSTANCES

Unibo-BP can implement either one or multiple DTN nodes on the same machine: template scripts for both cases are provided in the code.

A. Single Node

To implement one DTN node, the normal case, it is enough to start one instance of the BP Agent; then the node must be configured by adding information regarding the contact plan and convergence layers. The "start.sh" template, in the "Single_node" directory of Unibo-BP configuration files (Fig. 4) is an easy way to do that, with all commands grouped together. Before proceeding, a brief explanation about inter-process communications in Unibo-BP is necessary. Once launched, the BPA is directly reachable by all other processes

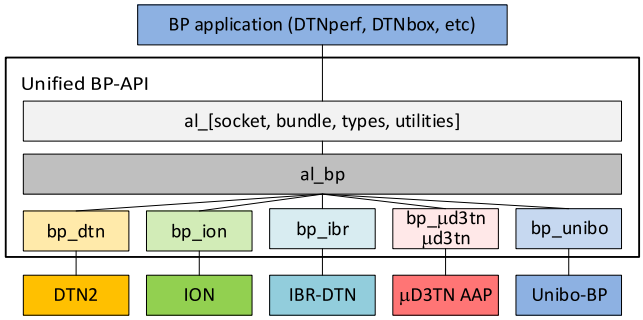


Fig. 3. Scheme of the Unified-API hierarchical structure.

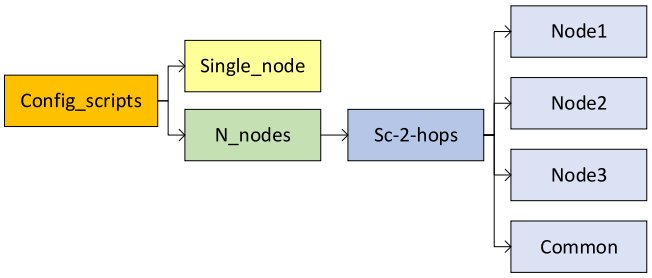


Fig. 4. Directory structure of the Unibo-BP configuration scripts provided as templates.

that need to communicate with it (Unibo-BP commands, Unibo-BP and DTNsuite applications, LTP protocol, etc.) through its Unix Domain Socket. To keep command syntax light, the socket name created by the BPA is not specified by the user, but found automatically by processes that need it, provided that they are launched from the same directory as the BPA. Although this mechanism simplify the management of multiple instances, it may be inconvenient when there is only one BPA and the user wants to launch commands (or DTN applications) from a separate directory. The problem is solved by setting the UNIBO_BP environment variable to the directory from which the BPA is started; once this is set it is possible to call commands and applications from anywhere.

An interesting feature of Unibo-BP is that it prevents the user from unintentionally launching multiple BPAs from the same directory, which would result in erratic behavior. This safety mechanism is based on a hidden "lock" file, created when the BPA is started and eliminated when stopped. However, if the BPA is closed abruptly, i.e., by closing the machine before launching the BPA stop command, the lock file survives; it can easily be removed by means of the "stop.sh" script template, also in the "single node" directory.

B. Multiple Nodes

To support multiple nodes we need to start one BPA instance for each node, on separate directories. This feature, inspired by ION, is useful when performing simple tests. The use of the template scripts provided in the "N-nodes" directory (Fig. 4) is recommended. These are based on a hierarchical structure, similar to that used in the NASA development kit and in ION. Each network layout, or "scenario", has its own umbrella directory (e.g., "Sc-2-hops" in the figure), which includes a directory for each node plus the Common directory. The former contain node-specific configuration commands, e.g., to start the node, to configure "inducts" and "outducts" etc., the latter commands common to all nodes of the scenario, such as those related to contact plan information. The "N_nodes" directory also contains a few scripts to start or stop all nodes with just one command. Starting from the templates provided, it should be straightforward for the user to build his own scenarios.

V. Tests and First Uses in Research

The development of Unibo-BP required continuous tests to check the correct implementation of bundle processing procedures. These preliminary tests were then followed by a comprehensive series of interoperability tests, complemented by Wireshark analysis [37]. Last, Unibo-BP has recently been used in an extensive series of research experiments.

A. Development Tests

These tests required either one or multiple nodes, but in both cases they were performed on a single machine. In this way, it was possible to check new features as soon as developed, without the need to copy the code to other machines and recompile. In practice, one instance of BPA was launched on the IDE platform and here debugged, while companion instances were launched on separate directories to implement other nodes.

B. Interoperability Tests

A different approach, based on virtual machines (VMs) was followed to test Unibo-BP interoperability. We used the testbed manager Virtualbricks [38] to build the testbed layout shown in Fig. 5, consisting of 3 VMs connected by virtual switches. All VMs ran GNU/Linux and were equipped with all major BPv7 implementations, such as ION, DTNME, and later also μD3TN. Depending on the specific test, Unibo-BP was launched on one of the three machines, and two of the other implementations on the others. Thanks to the Unified-API, all tests were performed with DTNperf client and server on source and destination nodes, with the same syntax, independently of the BP implementation actually running on these specific nodes, which greatly simplified tests.

As an example, here we report an interoperability test where the DTNperf client on the Unibo-BP node was set to generate 5 small bundles destined to DTNperf server on the ION node, with TCPCLv3 as convergence layer. DTNperf client (Fig. 6) allowed us to select the amount of data to be transferred (-D5k) and bundle characteristics, e.g., payload dimension (-P1k), while the server to check bundle traffic regularity, by direct inspection of its output (Fig. 7). Note that the BP implementation running on each node was correctly identified by both client and server (in bold on line 3 of each output).

The interested reader can find similar tests in [26]; they demonstrated not only Unibo-BP interoperability with other BP implementations, but also interoperability between any pair of the four BPv7 implementations considered.

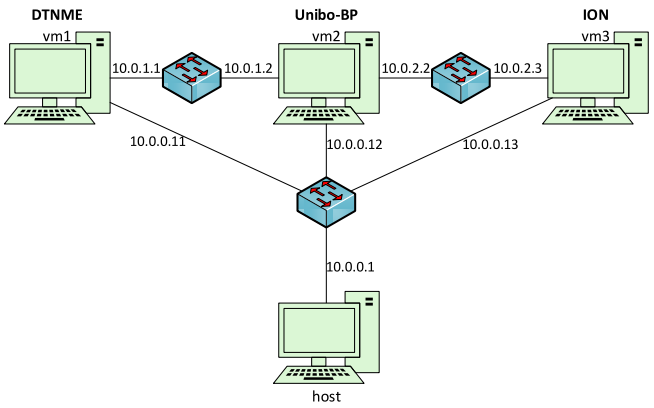


Fig. 5. The network topology used in interoperability tests. BP implementations actually used varied as required by different tests.

```
$dtnperf --client -d ipn:2.2000 -D5k -P1k -R1b
DTNperf version: 4.0.1
Running BP implementation: Unibo-BP
UNIFIED_API version:   2.0.17
Registered at: char* Eid: "ipn:1.650"
Bundle timestamp: 754159635263.0
Bundle timestamp: 754159636264.0
Bundle timestamp: 754159637264.0
Bundle timestamp: 754159638264.0
Bundle timestamp: 754159639265.0
Bundles sent = 5, total data sent = 5.000 kbyte
Total execution time 6.003 seconds
Throughput = 6.663 kbit/s
```

Fig. 6. Interoperability test between Unibo-BP and ION: output of the DTNperf client on Unibo-BP node.

```
$dtnperf --server
DTNperf version: 4.0.1
Running BP implementation: ION
UNIFIED_API version:   2.0.17
Registered at: char* Eid: "ipn:2.2000"
bundle 1: 1000 bytes from ipn:1.650
bundle 2: 1000 bytes from ipn:1.650
bundle 3: 1000 bytes from ipn:1.650
bundle 4: 1000 bytes from ipn:1.650
bundle 5: 1000 bytes from ipn:1.650
```

Fig. 7. Interoperability test between Unibo-BP and ION: output of the DTNperf server on ION node.

C. RFC 9171 "On the Wire" Compliance

The Wireshark traffic analyzer supports both BPv7 and TCPCLv3, therefore it can be suitably used to check format compliance with RFC 9171 of bundles generated by Unibo-BP. In the example shown in Fig. 8. Wireshark was able to recognize all bundle fields and all bundle blocks but one (type 193). This corresponds to the ECOS extension, which is not supported yet by Wireshark, most likely because its BPv7 version is still work in progress. It is worth stressing that although Wireshark analysis is powerful, it cannot replace interoperability tests: it can only analyze format compliance of generated or received bundles, not whether bundles are correctly processed by the BPA. This is why interoperability tests and Wireshark analysis are complementary and not alternative.

```
DTN TCP Convergence Layer Protocol Version 3
DTN Bundle Protocol Version 7, Src: ipn:1.762, Dst: ipn:2.2000,
  Indefinite Array: 9f
> Primary Block, CRC Type: CRC-16
  [Bundle Identity: Source: ipn:1.762, DTN Time: 754163839456,
  [Destination Service: 2000]
> Canonical Block: Previous Node, Block Num: 2, CRC Type: None
> Canonical Block: Bundle Age, Block Num: 3, CRC Type: None
> Canonical Block: Hop Count, Block Num: 4, CRC Type: None
> Canonical Block: Type 193, Block Num: 5, CRC Type: None
> Canonical Block: Payload, Block Num: 1, CRC Type: None
  Indefinite Break: ff
Data (1000 bytes)
```

Fig. 8. Wireshark dissection of one bundle generated by Unibo-BP. All blocks are correctly recognized, except for ECOS extension (type 193).

D. Unibo-BP Use in Recent LTP Studies

As soon as developed, Unibo-BP was used to investigate LTP performance on optical links, in cooperation with DLR, the German Aerospace Center [39]. The environment considered was characterized by a short RTT, a relatively high Tx speed and correlated losses; each experiment involved the transmission at high speed of many thousands of bundles, which was a sort of stress-test for Unibo-BP. Tests were performed on real machines (ordinary x86 PCs) connected with 1 Gbit/s Ethernet cards, as high speed was important, as said. In all tests Unibo-BP proved reliable and able to support the nominal Tx speed set for LTP, i.e., 500 Mbit/s. These experiments involved the concurrent use of Unibo-BP, Unibo-LTP and Unified-API and represent the first use for research purposes of the whole Unibo-DTN package, with Unibo-BP at its core.

VI. Conclusion

Unibo-BP is a new implementation of Bundle Protocol version 7 by the University of Bologna. It is research driven and space-oriented, as demonstrated by the support of experimental versions of both the LTP protocol and the CGR/SABR routing algorithm. It is IRF-ready as it supports the novel concepts of regions and hierarchical routing. The paper starts with a broad overview of Unibo-BP and then focuses on interfaces to DTNsuite, Unibo-CGR and Unibo-LTP, to highlight some features of great interest for future research, such as the ability to counteract loop insurgence on-the-spot, or to map bundle QoS parameters to LTP colors. Unibo-BP configuration is performed by commands instead of configuration files, which facilitates the use of scripts to launch both single and multiple instances on the same machine, as detailed in the paper. Unibo-BP is fully compliant with RFC 9171 and interoperability with major BPv7 implementations, such as ION, DTNME and μD3TN has been thoroughly tested. Although very recent, it has already proved its usefulness in research, as described.

Unibo-BP is released as Open Source Software under GPLv3 license and as this is available to all the DTN community.

APPENDIX

This is an overview of the most important Unibo-BP commands. Syntax examples are taken from the "start.sh" bash file provided as a template in the Unibo-BP package.

A. Commands

Each command consists of a command name, which may be followed by parameters (optional unless otherwise specified) and subcommands (grouping options or other subcommands), as shown below:

```
<command name> [options] [<subcommand name>[options]]
```

1) "unibo-bp": This command is used to start (or resume) the BPA process, hence we find it at the beginning of the "start.sh" file. With the options below, it starts the BPA as a daemon, with a BP database of 50 MB, setting the local node EID in both the schemes, "ipn" and "dtn", allowed by RFC 9171 [4]. The type of storage used by the BP database is not defined, which means that the default "mmap" (memory mapped), is implicitly used. Possible alternatives are "persistent" (on file) and "volatile" (RAM only).

```
unibo-bp start --set-storage-size
50000000 --dtn-admin dtn://myNode.dtn/
--ipn-admin ipn:101.0 -daemon
```

Note that the existence of two valid EID schemes, mainly for historical reasons ("dtn" was preferred by people dealing with terrestrial networks, while "ipn" is the only scheme allowed in space by CCSDS [5]) has often led to interoperability problems between DTN nodes not using the same scheme, as it was predictable. To eliminate these trivial but annoying problems, in Unibo-BP each node must have both EIDs.

2) "unibo-utility": This command is mainly designed as a general purpose container for future needs. At present is only used in the "script.sh" file to set the environment variable REFERENCE_TIME at the current time, with the following syntax:

```
export REFERENCE_TIME=$(unibo-bp-utility
--get-utc-time +0)
```

3) "unibo-bp-tcpcl": This command starts an instance of the TCP Convergence Layer process and in the "start.sh" file comes immediately after BPA start:

```
unibo-bp-tcpcl -daemon
```

The TCPCL process is unique, because although is initiated separately, once started it is controlled by BPA: user commands to TCPCL, such as those to create "inducts" and "outducts", must be entered using "unibo-bp-admin".

4) "unibo-bp-admin": This is the most comprehensive command, including all administration directives used to "configure" the BPA (plus the TCPCL process, as said). They are organized into ten subcommands: "stop", "logger", "storage", "region", "routing", "contact", "range", "extension", "whoami" and "tcpcl". Because of their importance, they will be treated apart in next sub-section.

5) "unibo-bp-remote-admin": This command starts a server that enables reception and execution on the local node of administration commands generated on other nodes. In order to use this feature, it is necessary to specify a listening EID, i.e., the local EID to which bundles containing remote administration commands must be sent. More details in [29]. The following command is present in the "script.sh" template, but is commented, i.e., disabled by default:

```
unibo-bp-remote-admin ipn:2.3 --daemon
```

6) "unibo-bp-ping/echo/send/sink": As with all BP implementations, a few basic programs to send or receive bundles are also available in Unibo-BP. As an example, at the end of the start.sh script two instances of the echo program are started, for the "ipn" and "dtn" schemes, respectively:

```
unibo-bp-echo --scheme ipn --daemon
unibo-bp-echo --scheme dtn -daemon
```

In the absence of other options, the first "echo" program will register with the local node number and default service number 7, e.g., as ipn:12.7 on node 12. The second, with the demux token "echo", appended to the "dtn" EID of the local node, e.g., as dtn://mynode/echo: The program does what expected, i.e., it sends back any bundle it receives, and thus it is suited to work in partnership with the ping program.

In the "start.sh" script, there is also the command that can be used to send a "hello world" bundle, ready to be used; the options shown below ask for the generation of a "bundle status report" once the bundle is delivered, which may be useful for testing.

```
unibo-bp-send --destination ipn:123.4
--report-to ipn:789.101 --delivered
--payload-string "Hello, World!"
```

B. "unibo-bp-admin" Sub-Commands

This command includes ten sub-commands, the most important of which are presented below. In turn, these subcommands may be organized into second layer subcommands, or sub-subcommands.

1) "region": With "home" and "outer" sub-subcommands it is used either to allocate a node to a region or to administer the "home" and "outer" regions of an IRF "passageway". In the "script.sh" file, it is used to define the nodes that belong to the local node region, e.g.:

```
unibo-bp-admin region home --register-
node ipn:102.0
```

2) "routing": This subcommand has two second layer subcommands, "static" and "cgr". The latter is used to tailor Unibo-CGR features to user need; it has 15 parameters, but luckily for the user they are simple to understand and use, as can be seen from the example below, which enables the moderate source routing option of CGR [33]. See [29] for a comprehensive list of options.

```
unibo-bp-admin routing cgr --
enable-moderate-source-routing
```

3) "contact": This subcommand is used to manage contact plan information, saved by the local BPA in the BP database. It has four sub-subcommands: "add", "remove", "change" and "get". By default, a contact is inserted/found within the

region to which the contact's sender and receiver nodes belong, therefore it is necessary to pre-register the nodes in the local node common region before adding the contact.

The syntax is the same as that used in ION, except that it is a bit more verbose in order to be self-explanatory.

```
unibo-bp-admin contact add --start-time
+0 --end-time +86400 --sender ipn:101.0
--receiver ipn:102.0 --xmit-rate 1000000
--reference-time $REFERENCE_TIME
```

Contacts play a key role in DTN, especially in space environments. They are essential in CGR but are also used by LTP, to stop segment transmission and also to freeze RTO timers when contacts end. They are also used by Unibo-BP to schedule the passage of bundles from the BP to the convergence layer, a feature taken from ION that allows BP to enforce scheduled transmissions on top of convergence layers that are not LTP, such as TCP and UDP. Note that in Unibo-BP, the node EIDs can be inserted not only in the "ipn" but also in the "dtn" scheme, which could pave the way to a possible extension of CGR and LTP to "non-ipn" nodes, thus simplifying network management.

4) "range": This is very similar to the "contact" subcommand. The example below sets the range, between the same two nodes and for the same interval as in the contact example previously presented.

```
unibo-bp-admin range add --start-time
+0 --end-time +86400 --sender ipn:101.0
--receiver ipn:102.0 --owlt
1 --reference-time $REFERENCE_TIME
```

As the One Way Light Time (OWLT) parameter is given in seconds, range updates are necessary only when the distance between the two nodes changes by more than 1 light second, i.e., by more than 300 000 km. We can assume that these updates will be few, even in space. Anyway, as range information is essential in CGR, ranges must be provided for all node pairs appearing in contacts, even if the nodes are static and even close to each other. For these nodes a minimum safety value of 1s is recommended, to help prevent possible routing instabilities. By contrast to ION, ranges in Unibo-BP are strictly unidirectional, like contacts.

5) "extension": This subcommand enables/disables the insertion of experimental extension blocks into the bundles created by the local node. At present it can only be used for RGR (Record Geographical Route) and CGRR (CGR-Routes) extensions, while all other extensions are always on.

```
unibo-bp-admin extension --enable-rgr
```

6) "tcpcl": This subcommand is used to configure the "unibo-bp-tcpcl" process, with "induct" and "outduct" sub-subcommands. The example below spawns an "induct" (i.e., a TCPCL server) on the local node, listening on the default IP (0.0.0.0) and port (1000):

```
unibo-bp-admin tcpcl induct add
```

The next example starts an "outduct" (TCPCL client) towards a neighbor node identified by both its BP EID and IP address. Of course, to establish the connection, an "induct" must already have been started on the neighbor node.

```
unibo-bp-admin tcpcl outduct add --peer
ipn:102.0 --hostname 10.0.0.102
```

ACKNOWLEDGMENT

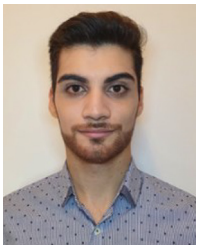
The author thanks to all the authors of BP implementations cited in the paper and in particular to Scott Burleigh and Felix Walter for their support on ION and μD3TN, respectively.

REFERENCES

[1] V. Cerf et al., "Delay-tolerant networking architecture," Internet RFC 4838, Apr. 2007.
[2] "Rationale, scenarios, and requirements for DTN in space," Consult. Comm. Space Data Syst., Washington, DC, USA, Rep. CCSDS 734.0-G-3, 2014.
[3] K. Scott, S. Burleigh, "Bundle protocol specification," Internet RFC 5050, Nov. 2007.
[4] S. Burleigh, K. Fall, E. Birrane, "Bundle protocol version 7," Internet RFC 9171, Jan. 2022.
[5] "CCSDS bundle protocol specification," Consult. Comm. Space, Syst., Washington, DC, USA, Rep. CCSDS 734.2-B-1, Sep. 2015.
[6] S. Burleigh, F. Templin, "Bundle protocol extended class of service (ECOS)," Internet Engineering Task Force, Internet-Draft draft-burleigh-dtn-ecos-00, May 2021. Work in progress.
[7] S. Burleigh, "Interplanetary overlay network: An implementation of the DTN bundle protocol," in *Proc. 4th IEEE Consum. Commun. Netw. Conf.*, 2007, pp. 222–226, doi: 10.1109/CCNC.2007.51. [Online]. Available: http://sourceforge.net/projects/ion-dtn/
[8] "DTNME (DTN Marshall Enterprise) code." github. Accessed: Feb. 2, 2024. [Online]. Available: https://github.com/nasa/DTNME
[9] A. Schlesinger, B. M. Willman, L. Pitts, S. R. Davidson, and W. A. Pohlchuck, "Delay/disruption tolerant networking for the international space station (ISS)," in *Proc. IEEE Aerosp. Conf.*, 2017, pp. 1–14, doi: 10.1109/AERO.2017.7943857.
[10] A. Hassan, A. Trigui, Y. Savaria, and M. Sawan, "High-temperature fully integrated wireless monitoring systems for aerospace applications," in *Proc. IEEE Int. Conf. Wireless Space Extreme Environ. (WiSEE)*, 2023, pp. 47–52, doi: 10.1109/WiSEE58383.2023.10289273.
[11] "μD3TN." d3tn. Accessed: Feb. 2, 2024. [Online]. Available: https://d3tn.com/ud3tn.html
[12] M. Feldmann, J. A. Fraire, F. Walter, and S. C. Burleigh, "Ring road networks: Access for anyone," *IEEE Commun. Mag.*, vol. 60, no. 4, pp. 38–44, Apr. 2022, doi: 10.1109/MCOM.001.2100835.
[13] R. Dudukovich, B. LaFuente, A. Hylton, B. Tomko, and J. Follo, "A distributed approach to high-rate delay tolerant networking within a virtualized environment," in *Proc. IEEE CCAAW*, 2021, pp. 1–5, doi: 10.1109/CCAAW50069.2021.9527297. [Online]. Available: https://github.com/nasa/HDTN
[14] L. Baumgärtner, J. Höchst, and T. Meuser, "B-DTN7: Browser-based Disruption-tolerant networking via bundle protocol 7," in *Proc. ICT-DM Conf.*, Paris, France, 2019, pp. 1–8, doi: 10.1109/ICT-DM47966.2019.9032944. [Online]. Available: https://github.com/dtn7/dtn7-rs
[15] C. Caini and L. Persampieri, "Unibo-BP: A new bundle protocol implementation," in *Proc. IEEE WiSEE*, Aveiro, Portugal, 2023, pp. 29–34, doi: 10.1109/WiSEE58383.2023.10289353.
[16] M. Ramadas, S. Burleigh, and S. Farrell, "Licklider transmission protocol—Motivation," Internet RFC 5325, Sep. 2008.
[17] M. Ramadas, S. Burleigh, and S. Farrell, "Licklider transmission protocol—Specification," Internet RFC 5326, Sep. 2008.
[18] "Licklider transmission protocol (LTP) for CCSDS," Consult. Comm. Space Data Syst., Washington, DC, USA, Rep. CCSDS 734.1-B-1, 2015.
[19] "Schedule-aware bundle routing," Consult. Comm. Space Data Syst., Washington, DC, USA, Rep. CCSDS 734.1-B-1, Jul. 2019.
[20] C. Caini, G. M. De Cola, L. Persampieri, "Schedule-aware bundle routing: Analysis and enhancements," *Int. J. Satel. Commun. Netw.*, vol. 39, no. 3, pp. 237–243, May/Jun. 2021, doi: 10.1002/sat.1384.
[21] A. Bisacchi, C. Caini, and S. Lanzoni, "Design and implementation of a bundle protocol unified API" in *Proc. ASMS/SPSC Conf.*, Graz, Austria, 2022, pp. 1–6, doi: 10.1109/ASMS/SPSC55670.2022.9914734.
[22] N. Alessi, "Hierarchical inter-regional routing algorithm for interplanetary networks," M.S. thesis, Dept. Comput. Sci. Eng., University of Bologna, Bologna, Italy, 2019. [Online]. Available: https://amslaurea.unibo.it/17468/
[23] M. Demmer, J. Ott, and S. Perreault "Delay-tolerant networking TCP convergence-layer protocol" Internet Eng. Task Force, RFC 7242, Jun. 2014
[24] "Unibo-BP code." gitlab. Accessed: Feb. 2, 2024. [Online]. Available: https://gitlab.com/unibo-dtn/unibo-bp

[25] "Unibo-DTN code." gitlab. Accessed: Feb. 2, 2024. [Online]. Available: https://gitlab.com/unibo-dtn
[26] "Unibo-CGR." gitlab. Accessed: Feb. 2, 2024. [Online]. Available: https://gitlab.com/unibo-dtn/unibo-cgr
[27] "Unibo-LTP." gitlab. Accessed: Feb. 2, 2024. [Online]. Available: https://gitlab.com/unibo-dtn/ltp
[28] "DTNsuite." gitlab. Accessed: Feb. 2, 2024. [Online]. Available: https://gitlab.com/dtnsuite
[29] L. Persampieri, "Unibo-BP: An innovative free software implementation of bundle protocol version 7 (RFC 9171)," M.S. thesis, Dept. Comput. Sci. Eng., University of Bologna, Bologna, Italy, Feb. 2023. [Online]. Available: https://amslaurea.unibo.it/27740/
[30] C. Bormann and P. Hoffman, Concise binary object representation (CBOR), Internet Eng. Task Force, RFC 8949, Dec. 2020.
[31] O. De Jonckère and J. A. Fraire, "Inter-regional routing in interplanetary networks with shortcuts and contact passageways," in *Proc. WiSEE Conference*, Aveiro, Portugal, 2023, pp. 87–92, doi: 10.1109/WiSEE58383.2023.10289358.
[32] E. Birrane and K. McKeever, "Bundle protocol security (BPSec)," Internet Eng. Task Force, RFC 9172, Jan. 2022.
[33] E. J. Birrane, C. Caini, G. M. De Cola, F. Marchetti, L. Mazzuca, and L. Persampieri, "Opportunities and limits of moderate source routing in delay-/disruption-tolerant networking space networks," *Int. J. Satell. Commun. Netw.*, vol. 40, no. 6, pp. 428–444, Nov./Dec. 2022, doi: 10.1002/sat.1421.
[34] A. Bisacchi, C. Caini, and T. De Cola, "Multicolor licklider transmission protocol: An LTP version for future interplanetary links," *IEEE Trans. Aerosp. Electron. Syst.*, vol. 58, no. 5, pp. 3859–3869, Oct. 2022.
[35] C. Caini, A. D'Amico, and M. Rodolfi, "DTNperf_3: A further enhanced tool for delay-/disruption- tolerant networking performance evaluation," in *Proc. IEEE Globecom*, Atlanta, GA, USA, 2013, pp. 3009–3015.
[36] S. Schildt, J. Morgenroth, W.-B. Pottner, and L. Wolf, "IBR-DTN: A lightweight, modular and highly portable bundle protocol implementation," *Electron. Commun. EASST*, vol. 37, pp. 1–11, Jan. 2011. [Online]. Available: https://github.com/ibrdtn/ibrdtn
[37] (Wireshark Found., San Francisco, CA, USA). Accessed: Feb. 2, 2024. [Online]. Available: https://www.wireshark.org/
[38] P. Apollonio, C. Caini, M. Giusti, and D. Lacamera, "Virtualbricks for DTN satellite communications research and education," in *Proc. PSATS*, Genoa, Italy, 2014, pp. 1–14.
[39] C. Caini, T. De Cola, A. Shrestha, and A. Zappacosta, "LTP performance on near-earth optical links," *IEEE Trans. Aerosp. Electron. Syst.*, vol. 59, no. 6, pp. 9501–9511, Dec. 2023, doi: 10.1109/TAES.2023.3322392

Carlo Caini (Member, IEEE) received the master's degree in electrical engineering "summa cum laude" from the University of Bologna, Italy, in 1986. Since 1990, he has been with the Department of Electrical, Electronics and Information Engineering, University of Bologna, where is working as an Associate Professor of Telecommunications. He is a coauthor of more than 100 papers on international conferences and journals; he has also supervised the development of many free software implementation of DTN protocols and applications. His main research interests are focused on delay-/disruption-tolerant networking and transport protocols for space and satellite networks.



Lorenzo Persampieri received the bachelor's degree (summa cum laude) in computer engineering from the University of Bologna in July 2020, with a thesis on the development of the Unibo-CGR implementation of the SABR routing algorithm, and the master's degree (summa cum laude) from the University of Bologna and in March 2023, with a thesis on the implementation of Unibo-BP. Although he left University for the Industry, he is still interested to research topics related to delay-/disruption-tolerant networking and its routing protocols.

Open Access funding provided by "Alma Mater Studiorum - Università di Bologna" within the CRUI CARE Agreement