CSPCL: A Stepwise, Non-Disruptive Transition to Delay-Tolerant Networking for CubeSat Operators

Mathias Durat[†], Sarah Theoulle[†], Hugo Ponthieu[†], Tristan-Mihai Radulescu[†], Olivier De Jonckère[*†],
[*] LIRMM, Université de Montpellier, France
[†] Polytech, Université de Montpellier, France

***Abstract*—As Delay-Tolerant Networking (DTN) gains increasing traction in the space community, CubeSat operators may face the challenge of adopting DTN capabilities without disrupting existing software stacks built around the CubeSat Space Protocol (CSP). We propose a non-intrusive deployment strategy, allowing operators to experiment with DTN capabilities while preserving the existing infrastructure. We evaluate interoperability with three BPA implementations: Hardy, µD3TN, and UNIBO-BP, with A-SABR contact-based routing integrated into two of them. CSPCL is an adapter that transmits DTN bundles over CSP to enable DTN capabilities on top of an unmodified CSP network. For the existing applications requiring CSP-to-CSP communication across DTN boundaries, we introduce Charon, a network proxy that encapsulates IP, CAN, or CSP traffic over the Bundle Protocol, allowing legacy applications to benefit from DTN features transparently. The full stack is validated on two PolarFire RISC-V boards communicating over a CAN bus, mirroring the hardware and transport layer of the Centre Spatial de l'Université de Montpellier's upcoming 12U CubeSat mission. Our results demonstrate end-to-end feasibility on representative flight hardware, and offer a concrete and accessible integration path for CubeSat operators considering a progressive adoption of DTN.**

***Index Terms*—Delay-Tolerant Networking, CubeSat, Convergence Layer Adapter,**

I. Introduction

Delay-Tolerant Networking (DTN) has emerged as a key technology for future space communication systems [1]. Deep-space networks operate under challenging conditions, including long propagation delays, intermittent connectivity, and constrained data rates resulting from extreme distances and varying link availability. Consequently, the Interagency Operations Advisory Group (IOAG) recommends DTN capabilities for most future surface and orbital assets on Mars [2]. This vision is reflected in major space communication initiatives such as NASA's Lunar Communications Relay and Navigation Systems (LCRNS) [3], ESA's Moonlight program [4], [5], and the NASA–ESA LunaNet interoperability architecture [6], [7], [8], [9].

DTN models space communication opportunities, named *contacts*, are defined as: scheduled, unidirectional transmission windows defined by a start time, an end time, and a nominal data rate, which together bound the available transmission *volume* and propagation *latency*. At its core, the Bundle Protocol (BP) [10], [11] provides end-to-end delivery over these contacts through a store-and-forward architecture: data are encapsulated into *bundles* that can be held for extended periods until a transmission opportunity arises, enabling asynchronous hop-by-hop forwarding without requiring immediate downstream availability. A Bundle Protocol Agent (BPA) implements BP and manages the full bundle lifecycle: generating, storing, routing (via a module such as SABR), transmitting and receiving over the underlying link, and discarding bundles past their time-to-live.

To deploy across heterogeneous infrastructures, BP relies on *Convergence Layer Adapters* (CLAs), which map bundle transmission onto an underlying transport (*e.g.*, a TCP-based CLA lets a BPA use conventional TCP/IP networks) while preserving BP semantics between BPAs that perceive each other as directly connected DTN neighbors. Because different segments of an end-to-end path can use different CLAs, this composition lets DTN operate as an overlay spanning heterogeneous environments—for instance, a gateway node with both TCP and LTP-based CLAs bridges a terrestrial TCP/IP segment to a space link running the Licklider Transmission Protocol (LTP).

Although DTN targets deep-space communication, its principles apply equally to smaller-scale networks with predictable disruptions: DTN already supports reliable payload-data retrieval aboard the ISS. For smaller operators, the *ring-road* concept [12] uses a handful of DTN-enabled satellites to relay data between isolated ground locations (islands, remote stations, oceanic buoys) and Internet-connected gateways, as demonstrated by the OPS-SAT mission [13].

II. Background and Motivation

The Centre Spatial de l'Université de Montpellier (CSUM) develops and operates CubeSats built on the CubeSat Space Protocol (CSP). Integrating Delay/Disruption Tolerant Networking (DTN) capabilities into these systems would enable interoperability with emerging space networking infrastructures while preserving compatibility with existing mission architectures. Since a non-intrusive deployment of DTN must overlay the Bundle Protocol (BP) over CSP, a dedicated Convergence Layer Adapter (CLA) is required, bridging BP-based DTN networks with CSP-based CubeSat systems and enabling delay/disruption-tolerant communications. We therefore introduce the CubeSat Space Protocol Convergence Layer (CSPCL) adapter.

The Adaptive Library for Schedule-Aware Bundle Routing

This work was supported, in whole or in part, by the French National Research Agency (ANR) through project ANR-25-CE25-4175-01.

(A-SABR) [14] instantiates algorithm variants derived from the CCSDS Schedule-Aware Bundle Routing standard [15], and an objective of the upcoming mission with the CSUM is to raise the library's Technology Readiness Level (TRL). Finally, delay/disruption tolerance is provided only at the Bundle Protocol (BP) layer above CSP and is therefore not directly accessible to existing applications operating at the CSP layer.

Since such CSP-layer applications cannot benefit from BP directly, we also introduce Charon (Section IV), a proxy that carries CSP traffic over BP so operators can experience DTN capabilities without modifying those applications.

CSUM's CubeSat avionics rely on the Controller Area Network (CAN) bus as the primary inter-component communication device. CSP operates directly over CAN, making it the natural intra-satellite link layer that any non-disruptive DTN integration must accommodate.

III. CubeSat Space Protocol Convergence Layer

A. Convergence Layer Adapter Library for CSPCL

CSPCL stands for CubeSat Space Protocol Convergence Layer. It is the implementation of a CLA using CSP as communication protocol. The CSPCL aims to regroup and aggregate all the methods needed for a BPA to use CSP as a Convergence Layer. Its goal is to have a common base that will allow an easier integration in any BPA. While other CLAs such as TCPCL are backed by an RFC, CSPCL follows an implementation-first path instead. We integrate a single shared behavioural core into three structurally different BPAs: Unibo-BP's process-isolated IPC daemon, µD3TN's in-process event loop, and Hardy's asynchronous FFI binding, described in the following subsections, showing that one implementation can be adapted to divergent BPA architectures without divergent CLA semantics. We see codifying these three behaviours, already exercised consistently across three independent BPAs, as the concrete next step toward a CSPCL specification suitable for multi-vendor interoperability.

The implementation of the CLA takes advantage of the features implemented in CSP, which are RDP (Reliable Datagram Protocol) enabling reliable CSP connections while using ACKs. It also uses SFP (Small Fragmentation Protocol), which allows fragmentation of the payloads transmitted (the size of a Bundle Protocol packet can greatly exceed the MTU of a CSP packet).

RDP's acknowledgements only confirm that individual CSP packets were enqueued for transmission; libcsp does not expose whether every fragment of a bundle was actually acknowledged as received, only its best-effort attempt to transmit them. To close this gap, CSPCL confirms end-to-end delivery of each bundle itself, at the application level, using nothing beyond stock CSP connection-oriented sockets. After all SFP fragments of a bundle have been transmitted, the sender blocks on a single one-byte acknowledgement, sent by the receiver on the same connection only once it has reassembled the complete bundle, bounded by a per-hop timeout. If no acknowledgement arrives, CSPCL invalidates the pooled connection, retries once, and otherwise reports the failure up to the BPA so it can react (*e.g.*, tear down the link) instead of silently discarding a bundle it believes was sent. We validate this mechanism empirically in Section V: under back-to-back transmission of multi-fragment bundles, timely detection of a corrupted or lost bundle is what allows the failure to be identified and reported rather than silently dropped.

For RDP to work, we need a connection between the two BPAs, in that way it is the same as what is done in the TCPCL: the CSPCL opens a connection used for the whole contact, and we assume the link is broken between two nodes if the connection breaks. The connections are managed with a pool allowing connections to live during the entire contact. The outbound pool uses Least Recently Used (LRU) eviction policy with a monotonic tick counter to track access patterns. When the pool is full, the least-accessed connection is closed to make room for new peers, while frequently-contacted destinations benefit from persistent cached connections. Thread-safe atomic operations via mutex locks ensure concurrent senders safely share the pool without duplication. An optional age-based invalidation mechanism (configurable via environment variable) allows stale connections to be pruned on-the-fly, enabling graceful adaptation to network topology changes while minimizing setup overhead for repeated communications. We validate this pooling logic with integration tests exercising LRU eviction and cache hit/miss accounting under repeated sends, and with a two-node test that kills the receiver mid-session to confirm the timeout and recovery behaviour described above: the very next send correctly reports a timeout within the configured bound, and the link recovers once the peer returns.

As part of the CAN standard itself, the bus already provides automatic error detection via CRC checksums, collision avoidance through priority-based arbitration, and low-power, minimal-wiring operation (two wires vs. dedicated point-to-point links). Those properties are inherited from the bus, not contributed by CSPCL, but that make CAN an attractive transport for CubeSats operating under severe mass, volume, and electrical power constraints. CSPCL's own contribution at this layer is narrower: it integrates CAN as a physical transport for CSP via the SocketCAN interface, framing CSP (and, transitively, SFP-fragmented Bundle Protocol) packets onto CAN frames without requiring any change to the existing intra-satellite CAN bus or its attached subsystems. By layering CSP/Bundle Protocol over CAN in this way, CSPCL lets CubeSats adopt standardized DTN bundles for store-and-forward messaging while continuing to rely on the same proven, space-flight-qualified CAN hardware already deployed onboard.

B. CLA for unibo-bp

Unibo-BP is a C++ implementation of Bundle Protocol version 7 (BP7) [10] developed at the University of Bologna as part of the broader Unibo-DTN project [16]. Its architecture is process-isolated: convergence layers run as independent processes, communicating with the core bundle processor over

a UNIX domain socket through the `CLAOverIPC` mechanism. For Unibo-BP, CSPCL is packaged as a standalone C daemon linking `libcspcl` and `libunibo-bp-api`, the C wrapper Unibo-BP exposes over its internal C++ interface; at startup the daemon opens this socket and performs the CLA handshake. This pairing highlights that `libcspcl` itself is BPA-agnostic: each integration couples the same C library to its host BPA through whatever mechanism that BPA's architecture prescribes—an IPC-connected process here, an in-process event loop for µD3TN, and an asynchronous FFI binding for Hardy, described below.

Bundles cross the IPC channel as file descriptors rather than in-memory buffers: outbound, the daemon reads the serialised bundle from a descriptor before handing it to `cspcl_send_bundle()`; inbound, it writes the received bundle bytes to a descriptor obtained from the BPA and submits it back. Link management is event-driven: a link is created on demand from an unknown inbound CSP address or when a contact-plan entry activates, with each peer assigned an IPN endpoint using its CSP node address directly as the IPN node number (*e.g.*, CSP address 2 maps to `ipn:2.0`); the link is torn down when the contact ends.

C. CLA for µD3TN

µD3TN (micro Delay/Disruption Tolerant Networking) is a lightweight, production-grade C implementation of BP7 for resource-constrained embedded and space systems. Unlike Unibo-BP's process isolation or Hardy's FFI binding, µD3TN integrates convergence layers directly as in-process modules through an event-driven callback mechanism. CSPCL registers at startup as a native convergence layer: contacts scheduled or closed by the BPA trigger CSPCL to open or tear down the corresponding CSP connection, and bundles are exchanged in-memory in both directions with no IPC overhead or separate daemon process. In the experimental setup, µD3TN instances serve as gateway nodes, bridging external IP traffic through Charon into the DTN overlay while leveraging A-SABR for routing decisions.

D. CLA for hardy

Hardy is an asynchronous Rust implementation of BP7 [10], structured as a Cargo workspace whose core BPA logic resides in a dedicated crate. As with the other BPAs, CSPCL integrates into Hardy as a CLA that delegates bundle transmission and reception to CSP while preserving BP semantics. Rather than reimplementing the convergence layer in Rust, Hardy reuses the same C `libcspcl` library via a dedicated `cspcl` crate: an FFI binding crate pairs a `bindgen`-generated low-level layer with a safe, asynchronous wrapper adapting the synchronous C interface to Hardy's Tokio runtime without blocking executor threads. Hardy exposes a `Cla` trait (`on_register` for inbound delivery, `forward` for outbound sends); the `cspcl` crate implements this trait and adds CSP as a first-class address type alongside TCP, with a lightweight runtime driving both directions over the chosen CSP interface (loopback or CAN bus).

IV. Charon: IP/CAN proxy over DTN

Charon is a lightweight IP, CSP, and CAN network tunnel that makes legacy CSP communications delay- and disruption-tolerant without requiring direct integration with DTN protocols. Related approaches at the BP layers include the BP socket [17]. It proxies three protocols—IP, CAN, and CSP: IP and CAN modes encapsulate packets, whereas in CSP mode Charon acts as a local proxy between two peers. Charon relies on Protobuf for BPA-side serialization via AAP2 [18] and on libcsp [19] to support the CSP stack.

In IP mode, Charon reads packets from a Linux TUN device (`/dev/net/tun`) and forwards each to the BPA as an Application Data Unit (ADU) within a bundle; the encapsulated packet traverses the DTN topology (Figure 1) until the receiving Charon peer extracts, decapsulates, and writes it to its own TUN interface, so the packet arrives as if sent over a standard IP socket—letting in-satellite applications reach a ground station without integrating the Bundle Protocol themselves.

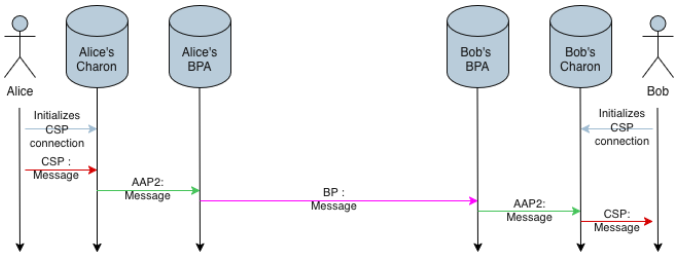

Fig. 1. Flow diagram of the CSP mode in Charon

CSP mode instead lets two CSP peers communicate directly: libCSP v1.6 (used by the CSUM) exposes no virtual network interface such as TUN or VCAN, so CSP packets cannot be reliably intercepted outside the process owning the CSP socket. Charon works around this by acting as a CSP peer-to-peer association server: Alice's local Charon forwards her packets to her BPA over AAP2; the bundle reaches Bob's BPA, which hands it to Bob's Charon instance for transmission to Bob. This gives peer-to-peer CSP communication over a DTN topology; its routing-scope limitation is discussed in Section IV-A.

Charon's AAP2-based integration currently targets µD3TN, whose AAP2 interface [18], a Protobuf-based API, is its primary means of sending bundles; Charon is therefore directly compatible with any BPA supporting AAP2, and transitively with every BPA µD3TN interoperates with.

Charon also supports CAN tunneling, using SocketCAN's [20] VCAN interface with the `CAN_RAW` flag to achieve a similar abstraction for CAN frames.

A. Limitations and Scope

We summarize Charon's current limitations explicitly, as boundaries of the contribution rather than incidental gaps:

- **Single BPA support.** Charon currently integrates with µD3TN only, through its AAP2 interface; it is transitively compatible with any BPA that µD3TN itself interoperates with, but has no direct integration with Unibo-BP or Hardy.
- **No local packet queue.** Charon holds no buffer of its own and depends entirely on the front-end BPA to accept outgoing packets; if the BPA is unavailable, packets in flight are lost rather than queued. A local buffer would mitigate this single point of failure and is left as future work.
- **No CSP routing in CSP mode.** Because each peer only interacts with its own local Charon instance (Figure 1), CSP-mode tunneling does not preserve CSP's native routing features — it provides point-to-point association between two named peers, not general CSP routing across a DTN topology. However, this could be mitigated if the bundle protocol and its implementation guarantee in-order delivery.
- **CAN frame reassembly is unresolved.** CAN-mode tunneling can split the CAN frames of a single CSP packet across bundle boundaries, risking reassembly failure at the receiving peer if those bundles are delivered out of order. Reassembling CSP packets from CAN frames before encapsulation is required to make CAN tunneling genuinely disruption tolerant, and is in progress at the time of writing. A countermeasure would be to rely on a bundle extension that allows bundle streaming with ordering preservation.

B. Security Considerations

Neither CSP nor the SocketCAN/CAN-bus transport CSPCL relies on provide built-in authentication or confidentiality; CSPCL and Charon inherit this open trust model, which mirrors the physical isolation typically assumed for an intra-satellite bus. Bundle Protocol Security (BPSec, RFC 9172) can be layered above CSPCL at the BPA to provide bundle-level integrity and confidentiality end-to-end, independent of the convergence layer, but neither CSPCL nor Charon currently enforce or configure it. Charon's ground-facing IP and CSP tunnels are a comparatively more exposed surface than the intra-satellite CAN link, since they may traverse links outside the physically isolated bus; hardening these tunnels (*e.g.*, authenticating CLA sessions, enabling BPSec on tunneled traffic) is left as future work.

V. Evaluation

The integration was validated on flight-representative hardware: two PolarFire RISC-V boards of Microchip linked by a CAN bus, on which two communicating Hardy nodes were successfully deployed. The setup described below is slightly different and uses VCAN interfaces for convenience.

A. Network Topology

The end-to-end validation topology forms a linear four-node chain in which each hop uses a distinct transport and BPA combination, exercising the full heterogeneity of the stack (Figure 2):

1) **Node 0 (alice)** µD3TN, It exposes an AAP2 socket to Charon and connects outward to Node 1 over TCPCLv3. An A-SABR Bundle Dispatch Manager (BDM) attaches to µD3TN over AAP2 and makes all forwarding decisions for traffic entering the network at this node.
2) **Node 1 (unibo)** It receives bundles from Node 0 via TCPCLv3 and forwards them over the virtual CAN bus using the `unibo-bp-cspcl` daemon.
3) **Node 2 (hardy)** It communicates with both neighbouring nodes over the virtual CAN bus via its built-in CSPCL CLA. A-SABR is integrated directly into Hardy as its routing agent.
4) **Node 3 (bob)** µD3TN, It receives bundles from Node 2 over the virtual CAN bus via the CSPCL CLA (CSP address 3) and delivers them to Charon over AAP2. An A-SABR BDM drives its forwarding table.

At both ends of the chain, a Charon instance bridges IP applications to the DTN layer.

B. Routing and Contact Plan

A-SABR runs as the routing algorithm on Node 0 and 3 (alice and bob) and Node 2 (hardy). Those instances share a common contact plan defining four nodes and six unidirectional contact entries. The alice–unibo hop is modelled with a nominal rate of 100 kbps and negligible propagation delay. The two CAN hops (unibo–hardy and hardy–bob) are each modelled at 50 kbps with a 5 ms one-way light-time, consistent with the CAN bus bandwidth budget. All contacts span the full experiment window with no scheduled disruptions, making this a connectivity validation rather than a disruption-tolerance test; Section V-G exercises disruption separately, by directly terminating a node.

C. Hardware Platform Validation

The physical validation platform consists of two PolarFire SoC RISC-V development boards connected by a CAN bus. This configuration mirrors the avionics architecture of the Centre Spatial de l'Université de Montpellier's upcoming 12U CubeSat mission, which uses CAN as the primary inter-board communication medium. The CAN interface is exposed to the Linux operating system through the SocketCAN subsystem, allowing standard `ip link` tooling to manage the bus and the CSP stack to bind to it directly using the `can_socketcan` driver. The PolarFire boards were used to test and validate the communication between two Hardy nodes using A-SABR for routing the bundles. They also validated that two Unibo nodes were communicating with each other.

D. Per-Layer Throughput, Latency, and Overhead

To isolate what each protocol layer adds, separately from the four-node chain's routing and multi-hop confounds, we benchmarked three configurations on a single node pair over one

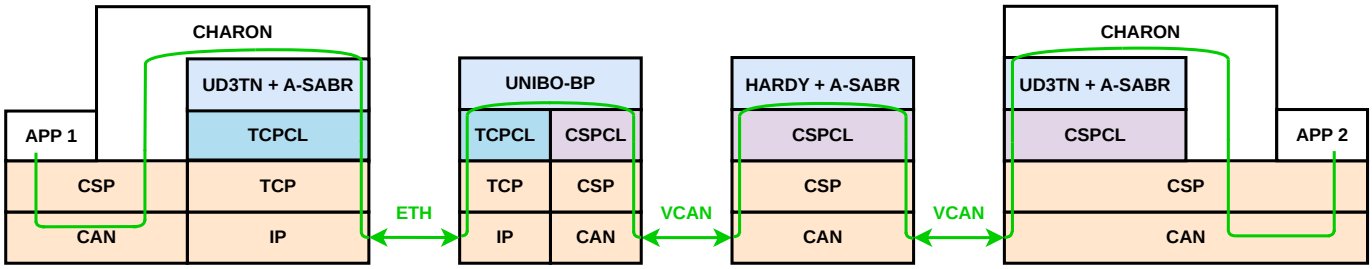

Fig. 2. Experimental setup

rate-unconstrained virtual CAN link: raw CSP (CAN/CSP), CSP under a real µD3TN BPA via CSPCL (CAN/CSP/BP), and the same path additionally tunneled through Charon (CAN/CSP/BP/CSP). Table I reports achieved throughput and mean single-bundle latency, mean ± stddev over $N=100$ independent repetitions for every row; CAN/CSP has no bundling concept and is therefore size-independent.

TABLE I
PER-LAYER THROUGHPUT AND MEAN SINGLE-BUNDLE LATENCY ON A SINGLE NODE PAIR OVER AN UNSHAPED VIRTUAL CAN LINK: RAW CSP (CAN/CSP), CSP UNDER A REAL µD3TN BPA VIA CSPCL (CAN/CSP/BP), AND THE SAME PATH ADDITIONALLY TUNNELED THROUGH CHARON (CAN/CSP/BP/CSP). MEAN ± STDDEV OVER $N=100$ INDEPENDENT REPETITIONS PER ROW/SIZE. CAN/CSP HAS NO BUNDLING CONCEPT AND IS THEREFORE SIZE-INDEPENDENT (FIXED 64 B UNIT).

Layer	Metric	64 B	256 B	1024 B	4096 B
CAN/CSP	kB/s	1699.6 ± 519.5			
	ms	0.16 ± 0.06			
CAN/CSP/BP	kB/s	0.6±0.0	2.5±0.0	9.9±0.1	39.5±0.4
	ms	8.80±4.71	6.09±3.00	6.32±3.19	8.55±3.07
CAN/CSP/BP/CSP (Charon)	kB/s	0.7±0.0	2.8±0.0	11.2±0.0	14.0±0.2
	ms	9.46±4.94	7.15±2.75	7.59±2.60	208.7±3.4

Throughput grows with bundle size across all three configurations because the round trip needed to send and acknowledge one bundle is largely fixed, and so is amortized over more bytes rather than reduced outright. This can appear to conflict with Table II's analytic BP7/CSPCL framing overhead, which as a fraction of bundle size falls from 85.9% at 64 B to 6.5% at 4096 B: larger bundles are in fact more byte-efficient, even though throughput does not scale linearly with size to match. The gap is explained, not contradicted: `cspcl_send_bundle()` holds a single pool-wide mutex across a bundle's full send-and-acknowledge cycle (Section III-A), so measured latency is dominated by this fixed round trip rather than by wire-framing bytes. We discuss narrowing this lock's scope as future work.

Charon tracks the BP-only row closely at 64–1024 B, but at 4096 B its latency jumps roughly 24× (208.7 ms vs. 8.55 ms) and its throughput falls below the BP-only row despite identical CSPCL/SFP fragmentation on both paths. This matches the payload exceeding Charon's TUN interface's 1500 B MTU at that size, triggering kernel-level IP fragmentation into multiple packets independently tunneled over the same BP hop: a concrete, measured cost of the IP-tunneling design already flagged as a limitation in Section IV-A. An independent rerun reproduced this within 1 ms, so the effect is not a one-off artifact.

TABLE II
ANALYTIC BP7 (RFC 9171 CBOR) PLUS CSPCL/SFP FRAGMENTATION OVERHEAD AS A FRACTION OF BUNDLE SIZE. APPLIES IDENTICALLY TO THE CAN/CSP/BP AND CAN/CSP/BP/CSP ROWS ABOVE (SAME FRAMING ON BOTH); INDEPENDENT OF THE MEASURED RUNS.

Payload	64 B	256 B	1024 B	4096 B
SFP fragments	1	2	5	17
Framing (B)	55	69	108	264
Overhead (%)	85.9	27.0	10.6	6.5

E. CSP-over-BP Header Cost and Bundling Granularity

Charon's CSP mode (Section IV) strips the original CSP header before the CSP payload is handed to the BPA as an AAP2 ADU, and a new CSP header is regenerated locally once the bundle reaches the receiving Charon instance. The CSP header is thus never carried across the radio link: paying for it twice is real, but confined to the two local CSP hops on either side of the DTN hop (ground-side application to Charon, and Charon to the satellite-side application), not to the constrained radio segment itself.

The BP7 bundle header does costs bytes on the radio link, paid once per bundle (~43 B of primary- and payload-block CBOR framing). Because this cost is per bundle rather than per CSP message, the lever that determines how much of the radio budget it consumes is how many CSP messages are batched into a single bundle before transmission. Table III reports this overhead as a percentage of total bytes crossing the radio link, across four batching granularities: it falls from 40.2% at one message per bundle to 6.3% at ten for 64 B messages, and from 14.4% to 1.7% for 256 B messages.

TABLE III
BP7 BUNDLE-HEADER OVERHEAD (~43 B, PAID ONCE PER BUNDLE) AS A PERCENTAGE OF TOTAL BYTES CROSSING THE RADIO LINK, BY CSP MESSAGES BATCHED PER BUNDLE. ANALYTIC, DERIVED FROM THE RFC 9171 CBOR FRAMING SIZE UNDERLYING TABLE II; INDEPENDENT OF THE MEASURED RUNS.

Payload	1 msg/bundle	2	5	10
64 B	40.2%	25.1%	11.8%	6.3%
256 B	14.4%	7.7%	3.3%	1.7%

F. Full-Chain Throughput and Latency vs. Modeled Contact Rate

We validated the complete four-node chain (Section V-A) against the contact plan's modeled 50/100 kbps rates (Section V-B), enforced as real bandwidth limits at the two payload sizes representative of this mission's CSP telemetry/command traffic (64 B, 256 B; larger sizes are not operationally relevant here and are characterized separately for connection-pool behavior in Section V-D). Table IV reports mean $\pm$ stddev over $N{=}100$ independent 10-bundle bursts. Delivery is fully reliable at both sizes, and achieved throughput reaches 79–94% of the modeled ceiling, consistent with the per-hop overhead already characterized in Section V-D, compounded here across three hops and two routing decisions.

TABLE IV
FULL FOUR-NODE CHAIN (ALICE→UNIBO→HARDY→BOB, VIA CHARON), CAN HOPS AND THE ALICE↔UNIBO LINK `tc`-SHAPED TO ENFORCE THE CONTACT PLAN'S MODELED 50/100 KBPS RATE. MEAN $\pm$ STDDEV ACROSS $N{=}100$ INDEPENDENT 10-BUNDLE BURSTS.

Payload	Mean lat. (ms)	Achieved (kB/s)	Ceiling (kB/s)	% of ceiling
64 B	609.5 $\pm$ 95.4	0.53 $\pm$ 0.08	0.67	78.9%
256 B	1263.0 $\pm$ 103.5	0.96 $\pm$ 0.05	1.02	94.1%

G. Store-and-Forward Recovery Under Node Disruption

We validated store-and-forward recovery under node failure in two configurations. In an earlier two-node configuration using Hardy, one node was killed outright while a bundle was in flight; the bundle was held in local storage by the surviving peer and successfully delivered once the killed node was restarted, confirming that a lost CSPCL connection triggers BP's store-and-forward path rather than a silently dropped bundle.

We repeated this at node granularity on the full four-node chain by terminating and restarting the `unibo-bp-cspcl` daemon for a few seconds while bundles were in transit. This exercises node failure more directly than a link-level interruption, since it discards the daemon's own connection-pool state along with its CSP connection. Traffic directed at the disrupted node during the outage was retained, and delivery resumed and completed fully once the daemon restarted.

VI. FUTURE WORK

The target 12U platform is linux-based, although other CubeSats from the CSUM are micro-controller-based. A-SABR is currently being ported for micro-controller support, but not the CSPCL stack. Future work includes a full experiment on microcontroller, with a compatible Bundle Protocol Agent, and extending Charon's BPA support beyond µD3TN (Unibo-BP, ION-DTN, Hardy).

The pool-serialization boundary identified in Section V-D indicates that the connection pool's locking granularity should be reduced by limiting the mutex to pool bookkeeping (lookup, creation, and eviction) and releasing it before fragmented SFP transmission and the application-level acknowledgement wait.

REFERENCES

[1] V. Cerf, S. Burleigh, A. Hooke, L. Torgerson, R. Durst, K. Scott, K. Fall, and H. Weiss, "Delay-tolerant networking architecture," Internet Requests for Comments, RFC Editor, RFC 4838, April 2007. [Online]. Available: http://www.rfc-editor.org/rfc/rfc4838.txt

[2] "The future mars communications architecture," *Interagency Operations Advisory Group*, 2022. [Online]. Available: https://www.ioag.org/Public%20Documents/MBC%20architecture%20report%20final%20version%20PDF.pdf

[3] "LCRNS Project - TEMPO," https://tempo.gsfc.nasa.gov/projects/LCRNS/, 2023, accessed: 2024-03-15.

[4] P. Giordano, A. Grenier, P. Zoccarato, L. Bucci, A. Cropp, R. Swinden, D. Gomez Otero, W. El-Dali, W. Carey, L. Duvet *et al.*, "Moonlight navigation service-how to land on peaks of eternal light," in *Proceedings of the 72nd International Astronautical Congress (IAC), Dubai, United Arab Emirates*, 2021, pp. 25–29.

[5] "ESA's Moonlight: Connectivity and Secure Communications," https://www.esa.int/Applications/Connectivity_and_Secure_Communications/Moonlight, 2023, accessed: 2024-03-15.

[6] D. J. Israel, K. D. Mauldin, C. J. Roberts, J. W. Mitchell, A. A. Pulkkinen, D. C. La Vida, M. A. Johnson, S. D. Christe, and C. J. Gramling, "Lunanet: a flexible and extensible lunar exploration communications and navigation infrastructure," in *2020 IEEE Aerospace Conference*. IEEE, 2020, pp. 1–14.

[7] P. Giordano, R. Swinden, C. Gramling, J. Crenshaw, and J. Ventura-Traveset, "Lunanet position, navigation, and timing services and signals, enabling the future of lunar exploration," in *Proceedings of the 36th International Technical Meeting of the Satellite Division of The Institute of Navigation (ION GNSS+ 2023)*, 2023, pp. 3577–3588.

[8] NASA, "NASA's LunaNet: Empowering Artemis with Communications and Navigation Interoperability," 2023, [Online]. Available: https://www.nasa.gov/humans-in-space/lunanet-empowering-artemis-with-communications-and-navigation-interoperability/, Accessed: Mar. 15, 2024.

[9] NASA, "Lunanet interoperability specification version 5 - draft," https://www.nasa.gov/wp-content/uploads/2023/09/lunanet-interoperability-specification-v5-draft.pdf?emrc=6f4483, 2023, accessed: Mar. 15, 2024.

[10] K. Scott, S. Burleigh, and E. Birrane, "Bundle protocol version 7," Internet Requests for Comments, RFC Editor, RFC 9171, January 2022, http://www.rfc-editor.org/rfc/rfc9171.txt. [Online]. Available: http://www.rfc-editor.org/rfc/rfc9171.txt

[11] Consultative Committee for Space Data Systems (CCSDS), "Ccsds bundle protocol specification (blue book, recommended standard CCSDS 734.2-B-1," https://public.ccsds.org/Pubs/734x2b1.pdf, September 2015.

[12] M. Feldmann, J. A. Fraire, F. Walter, and S. C. Burleigh, "Ring road networks: Access for anyone," *IEEE Communications Magazine*, vol. 60, no. 4, pp. 38–44, 2022.

[13] D. Evans and M. Merri, "Ops-sat: A esa nanosatellite for accelerating innovation in satellite control," in *SpaceOps 2014 Conference*, 2014, p. 1702.

[14] O. De Jonckère, L. Ma, and J. A. Fraire, "A-sabr: The adaptive library for schedule-aware bundle routing," in *2025 IEEE International Conference on Wireless for Space and Extreme Environments (WiSEE)*. IEEE, 2025, pp. 1–6.

[15] B. Book, "Schedule-aware bundle routing," *Consultative Committee for Space Data Systems*, 2019.

[16] C. Caini and L. Persampieri, "Design and features of Unibo-BP, the Unibo implementation of the DTN Bundle Protocol," *IEEE Journal of Radio Frequency Identification*, 2024, early access.

[17] S. Pierrot, O. De Jonckère, S. Burleigh, and C. Geoffroy, "Design and implementation of a native socket interface for the bundle protocol," in *2025 IEEE International Conference on Wireless for Space and Extreme Environments (WiSEE)*, 2025, pp. 1–6.

[18] F. Walter, M. Feldmann, J. A. Fraire, and S. Burleigh, "The architectural refinement of µd3tn: Toward a software-defined dtn protocol stack," in *2024 IEEE 10th International Conference on Space Mission Challenges for Information Technology (SMC-IT)*, 2024, pp. 161–170.

[19] https://github.com/libcsp/libcsp.

[20] https://docs.kernel.org/networking/can.html.