# Open Source Python Implementation Of CCSDS File Delivery Protocol To Support File-Based Operations

Shayan Majumder*, Artur Scholz†, George Goussetis* and Spyridon Nektarios Daskalakis*
*School of Engineering & Physical Sciences, Heriot-Watt University, Edinburgh, UK
{sm3054@hw.ac.uk, g.goussetis@hw.ac.uk, s.daskalakis@hw.ac.uk}
†LibreCube, Griesheim, Germany, artur.scholz@librecube.org

***Abstract*—The Consultative Committee for Space Data Systems (CCSDS) File Delivery Protocol (CFDP) enables reliable file transfer across space communication links that experience interruptions and delays, supporting diverse transfer modes with minimal human intervention. This paper presents an open-source Python library implementing CFDP, following guidelines from the CCSDS Blue Book. To date, this library is the only fully functional CFDP implementation in Python and is among the first open-source versions available. Developed in Python 3, with compatibility for MicroPython for low-power applications, the library supports both Class 1 (unreliable) and Class 2 (reliable) file transfers, using Zero Message Queue (ZMQ), CCSDS Space Packet Protocol, and User Datagram Protocol (UDP) for transport and network layers. An extendable virtual file store is included to support various data storage options. Full CCSDS compliance has been validated through cross-testing with the European Space Agency (ESA) CFDP implementation. A graphical user interface (GUI) enhances usability for non-technical users, supplementing the command-line interface. This open-source CFDP library is ideal for space and terrestrial file transfers and is increasingly valuable for file-based operations in space missions.**

***Index Terms*—CCSDS, CFDP, satellite communication, protocol**

## I. Introduction

The Consultative Committee for Space Data Systems (CCSDS) File Delivery Protocol (CFDP) is a standardized protocol developed to facilitate efficient, reliable, and flexible file transfer across space links. As spacecraft increasingly employ high-capacity solid-state storage, traditional packet-based data transfer methods have become insufficient to meet the demands of modern space missions. CFDP addresses these challenges by enabling the transmission of entire files rather than individual packets, optimizing space communications to handle the complexities associated with deep-space environments and high-latency operations [1]. Moreover implementing CFDP along with other layers of the CCSDS stack will help teams use this in satellite, rover, or any robotic missions as seen in Fig. 1 and hence make communication much easier.

The CFDP is ideal for sending entire files between spacecraft and ground stations, making it easier to handle large data sets like telemetry or images. CFDP offers two transfer modes: Class 1, which provides a "best-effort" transfer where some data loss is acceptable, and Class 2, which ensures data accuracy by retransmitting any missing information. CFDP is designed to work well even with changing bandwidths or long signal delays, such as in interplanetary missions, and it supports both one-way and two-way communication. Because of its efficiency and scalability, CFDP is a cost-effective solution for missions requiring reliable data transfers across multiple ground stations and complex communication setups.

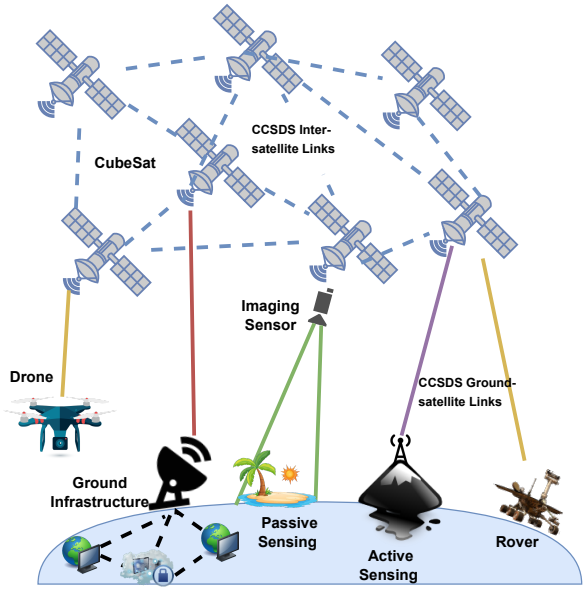


Fig. 1. A space-based internet system utilizing CCSDS standards for inter-satellite, ground-to-satellite, and satellite-to-sensor communications across a range of devices, including CubeSats, drones, imaging sensors, and autonomous rovers.

File-based systems like CFDP are increasingly used by modern autonomous spacecraft, with recent ESA missions like European Cooperation for Lightning Detection (EUCLID) [2] and Jupiter Icy Moons Explorer (JUICE) (both launched in 2023) [3] pioneering its use to efficiently handle large data transfers. By providing a standardized, reliable framework for

LibreCube Initiative, a registered non-profit in Germany.

seamless data exchange across different networks, CFDP helps lower integration costs and promotes collaboration among international space agencies. It also leverages advanced solid-state memory technology, enabling spacecraft to process and transmit large amounts of data more effectively for both operational and scientific needs.

TABLE I
COMPARISON: PACKET-BASED VS. FILE-BASED OPERATIONS

| Feature | Packet-Based Operation | File-Based Operation (CFDP) |
|---|---|---|
| **Reliability** | Involves manual intervention for retransmission of lost packets | Mostly automated transmissions and retransmissions |
| **Data Segmentation** | Transfers information segmented into small packets | Manages data as complete files or larger service data units (SDUs) |
| **Latency Handling** | Difficult over long delays | Handles high-latency links, suitable for deep-space missions |
| **Mission Complexity** | Requires heavy tracking and reassembly of packets | Simplifies operations by managing files as whole entities |
| **Flexibility** | Generally mission specific | Adapts to various network setups |
| **Error Handling** | Error handling at the packet level | End-to-end error detection and correction for entire files |
| **Data Checks** | Integrity checks per packet | Integrity verification for file segments |
| **Scalability** | Limited scalability | Highly scalable |
| **Resource Utilization** | Requires more resources | Optimized for low resource usage |

With the introduction of CFDP and other CCSDS standards, the space industry is shifting from packet-based to file-based operations, as shown in Table I. The push for standardization and the adoption of similar systems by academic missions [4] and new space companies are key drivers of this change.

A comparison of several open-source implementations of the CFDP highlights key differences and advantages of the Python-based CFDP. For instance, the Dario Lucia CFDP implementation [5] in Java is reliable and standards-compliant, but its Java foundation limits its versatility and practical use in some applications. Users unfamiliar with the Core Flight System (cFS) [6] may find the cFS-based CFDP challenging due to its dependency on the cFS software bus, which is specifically designed for the cFS architecture. In contrast, the Rust-based CFDP project [7] is still incomplete and requires additional development.

In contrast, the Python-based CFDP implementation provides a complete version of the CFDP stack, fully adhering to CCSDS guidelines [1] [8]. Python's simplicity and extensive resources make this implementation ideal for quick prototyping and development. As an open-source project, it offers small teams, hobbyists, and emerging space agencies a free tool to build their own communication systems. A notable feature is its compatibility with MicroPython, as shown in Fig. 5, enabling it to run on low-power devices, such as controllers commonly used in space hardware.

To make the CFDP system more user-friendly, a graphical user interface (GUI) was developed. This interface simplifies CFDP setup and operation, making it accessible even to those with limited technical skills. The Python-based implementation, with MicroPython support and an intuitive front-end, enables teams to easily set up, test, and deploy file transfer methods in their space missions. This design makes CFDP especially practical for small teams and hobbyists.

In summary, CFDP is transforming space communications by shifting from packet-based to file-based operations, better suited for the demands of modern space missions and enabling smoother collaboration among international space organizations. Supported by projects like ours at LibreCube, this shift is accelerating, making space communication technology more accessible.

This paper begins by summarizing the implementation, covering the design and development of CFDP. Next, cross-testing is performed to assess the implementation's performance and compatibility with other CFDP versions. Testing on embedded systems is also conducted to evaluate the system's practical feasibility. The section concludes with a summary of findings and new perspectives. To support seamless communication in space exploration and automation, the paper closes by discussing future plans, including integrating the LibreCube protocol stack with rovers and other autonomous systems.

## II. IMPLEMENTATION OVERVIEW

This project is implemented as free, open-source software under the MIT License and follows the PEP8 coding style. The package, named 'cfdp', can be easily installed using Python's pip package manager or directly from the source [9]. The CFDP implementation supports two main types of file transfers: Class 1 (unreliable) and Class 2 (reliable).

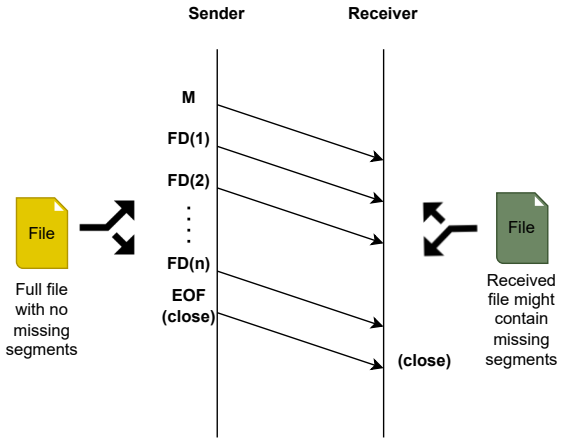


Fig. 2. Class 1 Transfer: In this mode, the sender transmits the file in segments without feedback or error recovery. The receiver may receive an incomplete file with missing segments, as there is no mechanism to request retransmission.

In Class 1 transfers (as shown in Fig. 2), the file is sent from sender to receiver without any feedback on successful receipt or attempts to recover lost data. On the receiver side, an inactivity timer monitors incoming protocol data units (PDUs) and resets with each new PDU. If the timer expires, the transfer is marked inactive, and the fault handler is triggered.

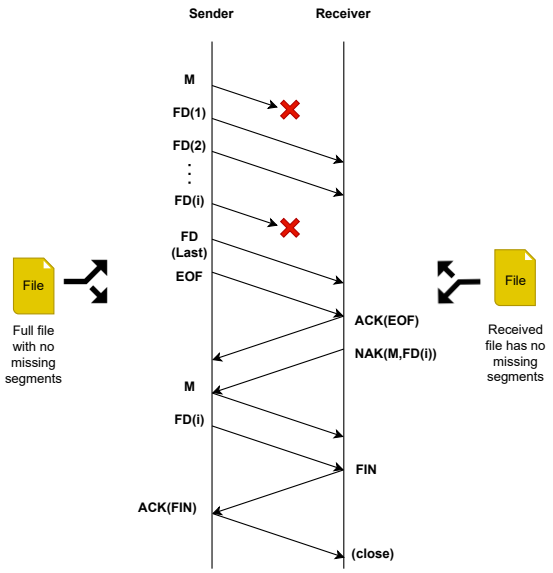


Fig. 3. Class 2 Transfer: The sender transmits the file with error-checking. The receiver sends acknowledgments (ACK) and requests retransmissions (NAK) for any missing segments, ensuring complete and accurate file transfer.

In Class 2 transfers, as shown in Fig. 3, the CFDP protocol uses negative acknowledgments (NAKs) and acknowledgments (ACKs) to ensure data reliability. NAKs request the retransmission of any lost data, while ACKs confirm the receipt of both the end-of-file (EOF) and finished PDUs. To ensure all data has been received, the receiving entity sends a finished PDU after successfully assembling the file. An ACK is then required to confirm that no data is missing after the EOF sequence.

There are four NAK strategies [8] available, which dictate when and where NAKs should be sent. This implementation uses the deferred NAK mode, where the receiving entity gathers all information on missing data until the EOF is received, then requests any missing data by issuing a NAK.

On the sender side, two timers are active. The inactivity timer, monitoring outgoing PDUs, triggers the problem handler if it times out, indicating idle transfer. If an ACK for the EOF $\text{ACK}_{\text{EOF}}$ is not received within a set period, the ACK timer is activated.

On the receiver side, three timers operate. The inactivity timer monitors incoming PDUs and triggers a fault if no data is received within the timeout period. If this occurs, the NAK timer initiates a verification process by issuing NAK PDUs to check for any missing file data. If an ACK for the final transfer state $\text{ACK}_{\text{FIN}}$ is not received, the ACK timer is activated. Each timer $\text{ACK}_{\text{EOF}}$, $\text{ACK}_{\text{FIN}}$, and NAK is stopped if the expected response is received; otherwise, the item is reissued. A counter tracks retransmissions, and if the maximum limit is exceeded, a fault is declared.

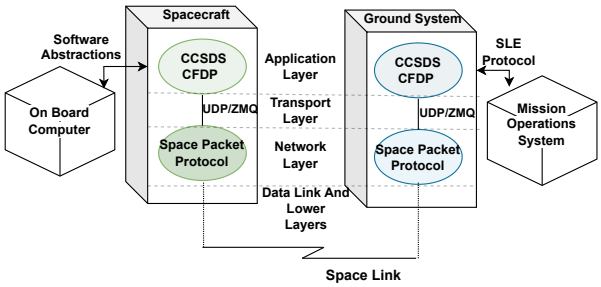


Fig. 4. CFDP integrated with additional protocols in LibreCube's open-source communication stack, demonstrating the layered structure between spacecraft and ground systems. CFDP operates at the application layer, supported by transport (UDP/ZMQ) and network layers, with the Space Packet Protocol ensuring compatibility across the data link and lower layers over space links.

To facilitate file transfers between CFDP entities, the CFDP library supports multiple transport and network layer protocols. As shown in Fig. 4, CFDP integrates with various CCSDS protocol layers [10]. One supported protocol is the User Datagram Protocol (UDP), a transport layer protocol that, although less reliable than Transmission Control Protocol (TCP), is lightweight and ideal for missions requiring fast response times. At higher CFDP levels, error handling is managed to ensure data reliability.

Additionally, the library includes support for Zero Message Queue (ZMQ), a flexible and efficient messaging system that provides dependable data transfer for both terrestrial and space networks. Finally, the CFDP library integrates with the CCSDS Space Packet Protocol [11] developed by LibreCube, enabling smooth operation within the CCSDS protocol stack. This integration is crucial for space missions that depend on reliable communication across space links.

The Python CFDP library allows for the creation of independent, custom classes for the transport layer, making it easier to add new features and update transport methods as project requirements evolve.

A virtual file store is a key feature that enhances CFDP's flexibility. It abstracts file storage, allowing CFDP operations to remain consistent regardless of the specific storage technology or platform used. This virtual storage uses Python's native file handling class, enabling standard actions like viewing, editing, or deleting files. It also supports both relative and absolute file paths. The following Python code shows an example of CFDPs virtual storage use.

```
from cfdp.filestore import NativeFileStore

# Initialize a relative file store path
fs = NativeFileStore("./files")

# Initialize an absolute file store path
fs = NativeFileStore("/tmp/files")
```

Designed for expandability, the virtual file system allows developers to adapt it for various storage systems, such as packet-based or cloud-based storage, based on mission or application needs.

## III. Cross-Testing and Compliance with CCSDS Standards

To ensure this Python-based CFDP implementation adheres to CCSDS standards, extensive testing with the European Space Agency's (ESA) internal CFDP system was conducted. These tests covered a variety of file transfer scenarios, including both Class 1 and Class 2 transfers. While specific test details and results cannot be disclosed due to the classified nature of ESA's internal CFDP Java code, it can confirmed that this implementation passed all critical test cases, fully aligning with the CCSDS Blue Book standards [8]. This compliance guarantees compatibility with other CCSDS-compliant systems, promoting broader acceptance in the space industry.

This implementation aligns with most of the latest features in the CFDP standards [1] [8]. However, some elements, such as cyclic redundancy check (CRC) and checksum verification, are still under development. While these features are essential for data integrity over unreliable links, their absence does not prevent successful file transfers. These capabilities are planned for the next development phase, further enhancing the protocol's ability to detect and resolve data inconsistencies.

## IV. Testing and User Interfaces

With the growth of compact, energy-efficient devices for space, implementing CFDP on small microcontrollers is essential. The CFDP code has been adapted for embedded systems with a custom port to MicroPython [12], providing a simple, low-power solution for file management on platforms with limited processing power and memory. This enables CFDP to be used on resource-constrained systems such as CubeSats, rovers, and compact scientific instruments where reliable file sharing is critical despite limited hardware resources.

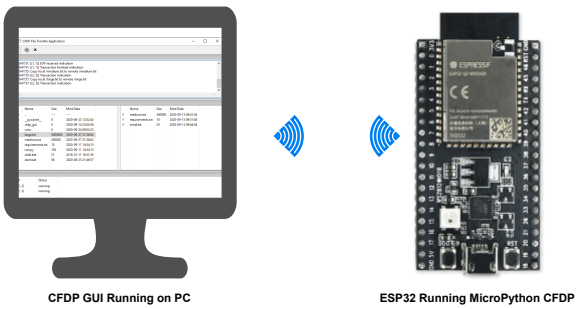


Fig. 5. ESP32 connected to PC over WiFi: The CFDP GUI runs on the PC, while the ESP32 executes CFDP using MicroPython, enabling wireless communication between the devices.

This is the first implementation of any file delivery protocol in MicroPython. For testing, an ESP32 (Fig. 5, right) board was programmed to run the MicroPython CFDP protocol. This implementation is fully open-source and can be easily replicated. The recommended setup is to connect it with the CFDP GUI over WiFi, as shown in Fig. 5, using any of the supported transport/networking protocols.

The primary goal of these tests was to demonstrate Class 1 and Class 2 transfers, along with additional functionalities such as filestore requests, proxy operations, directory listing requests, native filestore implementation, and the UDP transport layer. However, the limited memory capacity of ESP32 boards restricted testing with larger file transfers. Additionally, each change requires compiling to bytecode, which complicates the tracking process. Since CFDP has a relatively large footprint, running it on complex systems with concurrent processes can be challenging.

Verifying CFDP operability through command-line tools is essential, especially in automated environments where GUIs are unavailable. The following Python code provides an example of setting up a client-side application for file transfers, demonstrating the necessary configurations for successful file delivery. In this example, UDP transport is used with space packet routing. Configurations include setting IP addresses, binding ports, defining file paths, and selecting transmission modes.

```
import time

import spacepacket

import cfdp
from cfdp.transport.spacepacket import
    SpacePacketTransport
from cfdp.transport.udp import UdpTransport
from cfdp.filestore import NativeFileStore


udp_transport = UdpTransport(routing={"*": [("
    127.0.0.1", 5222)]})
udp_transport.bind("127.0.0.1", 5111)

spacepacket_transport = SpacePacketTransport(
    apid=111, transport=udp_transport, packet_type=
        spacepacket.PacketType.TELECOMMAND
)

cfdp_entity = cfdp.CfdpEntity(
    entity_id=1,
    filestore=NativeFileStore("../files/local"),
    transport=spacepacket_transport,
)

transaction_id = cfdp_entity.put(
    destination_id=2,
    source_filename="/abc.txt",
    destination_filename="/abc.txt",
    transmission_mode=cfdp.TransmissionMode.
        ACKNOWLEDGED,
)

while not cfdp_entity.is_complete(transaction_id):
    time.sleep(0.1)

input("Press <Enter> to finish.\n")

cfdp_entity.shutdown()
udp_transport.unbind()
```

The following Python code illustrates how to configure a server-side application to send and receive files.

```
import spacepacket
import cfdp
from cfdp.transport.spacepacket import
    SpacePacketTransport
from cfdp.transport.udp import UdpTransport
from cfdp.filestore import NativeFileStore


udp_transport = UdpTransport(routing={"*": [("
    127.0.0.1", 5111)]})
udp_transport.bind("127.0.0.1", 5222)

spacepacket_transport = SpacePacketTransport(
    apid=222, transport=udp_transport, packet_type=
        spacepacket.PacketType.TELEMETRY
)

cfdp_entity = cfdp.CfdpEntity(
    entity_id=2,
    filestore=NativeFileStore("../files/remote"),
    transport=spacepacket_transport,
)

input("Running. Press <Enter> to stop...\n")

cfdp_entity.shutdown()
udp_transport.unbind()
```

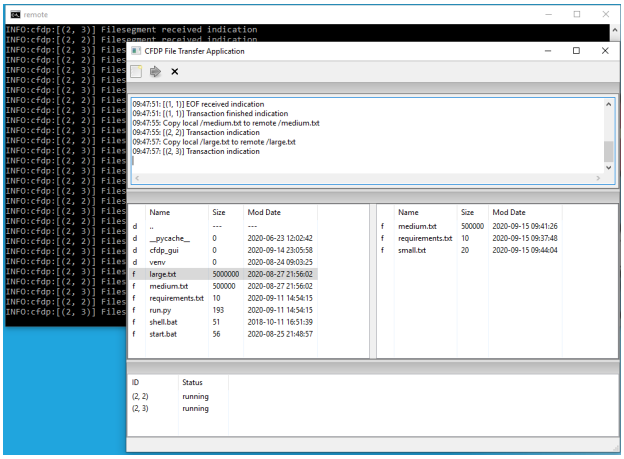

Fig. 6. CFDP GUI application: Interface for file transfers, resembling a typical FTP application, showing file status, transfer details, and transaction logs.

To make CFDP accessible for users unfamiliar with command-line operations, a GUI is also available, as shown in Fig. 6. The CFDP GUI simplifies tasks such as uploading and downloading files, browsing directories, deleting files, and more. It abstracts the protocol's complexities, allowing users to perform comprehensive tests without needing to write code.

## V. Future Work

The next steps for this implementation involve integrating CFDP functionality with other layers of the CCSDS protocol stack, such as the data link layer. Currently, LibreCube is focusing on protocols like the Unified Space Data Link Protocol (USLP) [13], which ensures seamless data transfer over diverse communication links. This integration will significantly enhance the robustness of LibreCube's CCSDS stack for real-world applications, enabling reliable file transfers even in challenging space environments.

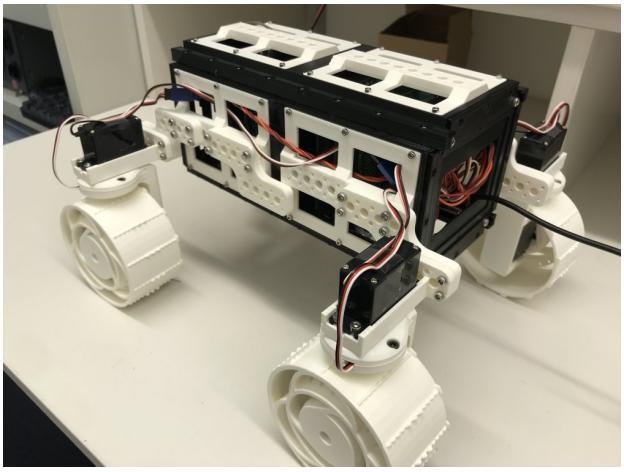

Fig. 7. LibreCube's 2U Rover Prototype under Development: Early-stage prototype showcasing the modular design and wiring layout for future space exploration applications.

Plans are also underway to demonstrate CFDP's operation over radio frequency (RF) using open-source tools like GNU Radio, allowing for practical, real-time RF-based testing [14]. Following successful tests, the CFDP will be deployed in the double-unit (2U) or three units (3U) rover project at LibreCube, as shown in Fig. 7, where it will facilitate file-based communications for remote rover operations. The success of these initiatives will provide valuable insights into how well this technology stack performs in actual space missions.

These developments aim to create a comprehensive, open-source framework for space communication protocols. This framework will align with international standards while being adaptable to the evolving needs of future space missions, from rovers and drones to satellite explorations.

## VI. Conclusion

This work presents an open-source Python implementation of the CFDP, addressing the growing need for file-based operations in space communications. It provides a reliable, flexible, and scalable solution for data transfer across deep-space and terrestrial environments, adhering to CCSDS standards. Key features include support for Class 1 and Class 2 transfers, enabling robust communication between space-to-ground, ground-to-space, and inter-space entities. The integration of various transport layers such as ZMQ, UDP, and the CCSDS Space Packet Protocol ensures adaptability to diverse mission profiles and network configurations.

This CFDP implementation has been rigorously tested with ESA's CFDP system and complies with the latest CCSDS standards, making it a dependable resource for space missions. Additionally, the MicroPython support enables lightweight,

energy-efficient communication for embedded systems, making it ideal for small-scale operations such as CubeSats and scientific rovers.

## ACKNOWLEDGMENT

This work was supported by the LibreCube initiative, a non-profit organization in Germany. The authors would like to thank Sven Steinert, Nikolas Bonisch, Sarah Quehl, Sidharth Shambu, and many other open-source contributors for their invaluable efforts. Special thanks to the Google Summer of Code program for funding open-source student projects.

## REFERENCES

[1] *Report Concerning Space Data System Standards: CCSDS File Delivery Protocol (CFDP) Part 1: Introduction and Overview*, Consultative Committee for Space Data Systems (CCSDS), May 2021, Green Book. [Online]. Available: https://public.ccsds.org/Pubs/720x1g4.pdf

[2] ESA, *Euclid's ground segment: prepared to download the Universe*, European Space Agency (ESA), accessed: 2024-10-31. [Online]. Available: https://esoc.esa.int/euclids-ground-segment-prepared-download-universe

[3] A. Valverde, C. Taylor, J. A. Montesinos, E. Maiorano, C. Colombo, C. Erd, and G. Magistrati, "CFDP configuration: EUCLID and JUICE scenarios," in *Proc. DASIA Data Systems In Aerospace*, Warsaw, Poland, June 2014.

[4] S. Majumder, H. Tambi, K. Mathur, P. Putrevu, and H. Venkat, "Design and Development of a High-Speed Communication System for a LEO Nano Satellite," in *Proc. Int. Astronautical Congress (IAC)*, Dubai, United Arab Emirates, 2021.

[5] D. Lucia, M. Doyle, and L. Bremond, "An open source implementation of CCSDS protocols and formats in Java," https://github.com/dariol83/ccsds, 2022, accessed: 2024-10-23.

[6] J. Wilmot, *Use of CCSDS File Delivery Protocol (CFDP) in NASA/GSFC's Flight Software Architecture: Core Flight Executive (cFE) and Core Flight System (CFS)*, Consultative Committee for Space Data Systems (CCSDS).

[7] M. Kolopanis and N. Mihalache, "cfdp-rs," https://github.com/ASU-cubesat/cfdp-rs, 2022, accessed: 2024-10-24.

[8] *Recommendation for Space Data System Standards: CCSDS File Delivery Protocol (CFDP)*, Consultative Committee for Space Data Systems (CCSDS), July 2020, Blue Book. [Online]. Available: https://public.ccsds.org/Pubs/727x0b5e1.pdf

[9] S. Majumder, S. Quehl, S. Shambu, and A. Scholz, "Python CFDP," https://gitlab.com/librecube/lib/python-cfdp, 2021, accessed: 2024-10-23.

[10] *Report Concerning Space Data System Standards: Overview of Space Link Protocols*, Consultative Committee for Space Data Systems (CCSDS), June 2001, Green Book. [Online]. Available: https://public.ccsds.org/Pubs/130x0g4e1.pdf

[11] A. Scholz, S. Majumder, and S. Shambu, "Python Space Packet," https://gitlab.com/librecube/lib/python-spacepacket, 2023, accessed: 2024-10-24.

[12] S. Majumder, "Implementing CCSDS File Delivery Protocol (CFDP) in MicroPython for GSoC 2021," https://medium.com/, 2021, accessed: 2024-10-25.

[13] *Report Concerning Space Data System Standards: Overview of the Unified Space Data Link Protocol*, Consultative Committee for Space Data Systems (CCSDS), June 2020, Green Book. [Online]. Available: https://public.ccsds.org/Pubs/700x1g1.pdf

[14] H. Tambi, S. Majumder, I. Khare, R. Hulsurkar, and K. Mathur, "Testing and Implementation of Communication Subsystem of a 3U CubeSat using Software-Defined Radio," in *Proc. Int. Astronautical Congress (IAC)*, Dubai, United Arab Emirates, 2021.