NASA/TM-20240014467

HDTN Test Framework Software Requirements Specification and Design Description

Ethan Schweinsberg, José Lombay-González, Shaun McKeehan, and Nadia Kortas
Glenn Research Center, Cleveland, Ohio

Eric Brace
HX5, LLC, Brook Park, Ohio

Rachel Dudukovich, Stephanie Booth, Brian Tomko, Timothy Recker, John Nowakowski, and Amber Waid
Glenn Research Center, Cleveland, Ohio

Wade A. Smith
Bastion Technologies Inc., Houston, Texas

November 2024

NASA STI Program Report Series

Since its founding, NASA has been dedicated to the advancement of aeronautics and space science. The NASA scientific and technical information (STI) program plays a key part in helping NASA maintain this important role.

The NASA STI program operates under the auspices of the Agency Chief Information Officer. It collects, organizes, provides for archiving, and disseminates NASA's STI. The NASA STI program provides access to the NTRS Registered and its public interface, the NASA Technical Reports Server, thus providing one of the largest collections of aeronautical and space science STI in the world. Results are published in both non-NASA channels and by NASA in the NASA STI Report Series, which includes the following report types:

- TECHNICAL PUBLICATION.
  Reports of completed research or a major significant phase of research that present the results of NASA programs and include extensive data or theoretical analysis. Includes compilations of significant scientific and technical data and information deemed to be of continuing reference value. NASA counterpart of peer-reviewed formal professional papers but has less stringent limitations on manuscript length and extent of graphic presentations.

- TECHNICAL MEMORANDUM.
  Scientific and technical findings that are preliminary or of specialized interest, e.g., quick release reports, working papers, and bibliographies that contain minimal annotation. Does not contain extensive analysis.

- CONTRACTOR REPORT.
  Scientific and technical findings by NASA-sponsored contractors and grantees.

- CONFERENCE PUBLICATION.
  Collected papers from scientific and technical conferences, symposia, seminars, or other meetings sponsored or cosponsored by NASA.

- SPECIAL PUBLICATION.
  Scientific, technical, or historical information from NASA programs, projects, and missions, often concerned with subjects having substantial public interest.

- TECHNICAL TRANSLATION.
  English-language translations of foreign scientific and technical material pertinent to NASA's mission.

Specialized services also include organizing and publishing research results, distributing specialized research announcements and feeds, providing information desk and personal search support, and enabling data exchange services.

For more information about the NASA STI program, see the following:

- Access the NASA STI program home page at http://www.sti.nasa.gov

NASA/TM-20240014467

HDTN Test Framework Software Requirements Specification and Design Description

Ethan Schweinsberg, José Lombay-González, Shaun McKeehan, and Nadia Kortas
Glenn Research Center, Cleveland, Ohio

Eric Brace
HX5, LLC, Brook Park, Ohio

Rachel Dudukovich, Stephanie Booth, Brian Tomko, Timothy Recker, John Nowakowski, and Amber Waid
Glenn Research Center, Cleveland, Ohio

Wade A. Smith
Bastion Technologies Inc., Houston, Texas

National Aeronautics and
Space Administration

Glenn Research Center
Cleveland, Ohio 44135

November 2024

Level of Review: This material has been technically reviewed by technical management.

This report is available in electronic form at https://www.sti.nasa.gov/ and https://ntrs.nasa.gov/

NASA STI Program/Mail Stop 050
NASA Langley Research Center
Hampton, VA 23681-2199

Contents

Preface

Space Communications and Navigation (SCaN) is developing new communications technologies to increase the amount of science data returned on future space missions. The High-Rate Delay Tolerant Networking (HDTN) project at NASA Glenn Research Center (GRC) will provide reliable internetworking as a high-speed path for moving data between spacecraft payloads, and across communication systems that operate on a range of different rates.

This document specifies the software requirements and describes the software design of the HDTN Test Framework.

HDTN Test Framework Software Requirements Specification and Design Description

Ethan Schweinsberg, José Lombay-González, Shaun McKeehan, and Nadia Kortas
National Aeronautics and Space Administration
Glenn Research Center
Cleveland, Ohio 44135

Eric Brace
HX5, LLC
Brook Park, Ohio 44142

Rachel Dudukovich, Stephanie Booth, Brian Tomko, Timothy Recker,
John Nowakowski, and Amber Waid
National Aeronautics and Space Administration
Glenn Research Center
Cleveland, Ohio 44135

Wade A. Smith
Bastion Technologies Inc.
Houston, Texas 77058

1.0 Introduction

1.1 Purpose

This document specifies the software requirements and outlines the software design for the HDTN Test Framework. It provides a guide for developing and verifying the framework, explaining the major design decisions, and specifying the requirements the framework will be developed to.

1.2 Scope

This document covers the architecture of the HDTN Test Framework, the major decisions associated with its design, a description of how the framework is executed, an overview of how the framework interfaces with external components, and the Test Framework requirements.

1.3 Overview

The HDTN Test Framework is a Python library that provides functionality to streamline the testing process for the HDTN software suite. It is designed to be used by scripts that execute test cases to directly verify software requirements. The framework leverages the Pytest Python library to facilitate test case execution and the Scapy Python library for network protocol support. Additionally, it is built with extensibility in mind, making it adaptable for other Delay Tolerant Networking (DTN) implementations.

The Test Framework consists of a single Computer Software Configuration Item (CSCI).

2.0 Documents

Documents listed in this section are the standards, specifications, handbooks, and other publications that guide this document.

2.1 Applicable Documents

Applicable documents are either cited in the body of this document or contain provisions directly related to, and necessary for, the performance of the activities described herein. Applicable documents are listed in Table 2-1.

TABLE 2-1.—APPLICABLE DOCUMENTS

Document number	Revision	Document title	Effective date
NPR 7150.2	D	NASA Software Engineering Requirements	03/08/2022

2.2 Reference Documents

Reference documents are those that **may** assist in understanding the intent of this document but are not required to execute the processes contained in this document. Reference documents are listed in Table 2-2.

TABLE 2-2.—REFERENCE DOCUMENTS

Document number	Revision	Document title	Effective date
HDTN-PLAN-022	A	HDTN Software Verification and Validation Plan	01/19/2024
HDTN-PLAN-003	A	HDTN Software Development and Management Plan	01/19/2024

3.0 Requirements

This section specifies the requirements for the Test Framework CSCI. The intent of this section is to document the expected behavior and functionality that meets the needs of stakeholders. The requirements along with the corresponding rationale and verification method are listed in Table 3-1.

TABLE 3-1.—TEST FRAMEWORK REQUIREMENTS

SW Req ID	Title	Requirement	Rationale	Verification method	Verification statement
HDTNTEST-001	Send Transport Layer Packet	The Test Framework shall provide a method to send a transport layer packet containing a user-defined payload.	Sending packets is a fundamental operation for testing convergence layer and bundle protocol requirements.	Test	This requirement is verified when a test shows the Test Framework delivers a properly structured user-defined payload.
HDTNTEST-002	Configure Destination	The Test Framework shall allow the user to specify the destination Internet Protocol (IP) address and port number for the transport layer packet.	The ability to control the destination of a packet is essential for simulating communication with various endpoints.	Test	This requirement is verified when a test shows a received packet contains the desired destination IP address and destination port number for the transport layer packet.
HDTNTEST-003	Receive Transport Layer Packet	The Test Framework shall provide a method to receive a transport layer packet.	Receiving packets is a fundamental operation for testing convergence layer and bundle protocol requirements.	Test	This requirement is verified when a test shows the Test Framework can receive a transport layer packet.
HDTNTEST-004	Configure Receiver	The Test Framework shall support specifying a timeout, port number, and maximum buffer size for receive operations.	These options provide flexibility to the test scripts, which is necessary for testing various scenarios.	Demonstration	This requirement is verified when a demonstration shows the Test Framework can accepts a timeout, port number, and buffer size for receive operations.

SW Req ID	Title	Requirement	Rationale	Verification method	Verification statement
HDTNTEST-005	Transport Layer Protocols	The Test Framework shall send and receive packets over all supported transport protocols.	The list of supported protocols is in *Table 5-4: Transport Protocols Supported by the Test Framework.* These are the protocols needed to exercise all DTN convergence layers.	Analysis	This requirement is verified when HDTNTEST-001 and HDTNTEST-003 are verified using all supported transport protocols.
HDTNTEST-006	Configure Unit Under Test (UUT)	Upon starting the UUT, the Test Framework shall provide a method to apply any supported configuration option available for the UUT.	Configuring the UUT is essential for testing different scenarios. A list of supported configuration options is in the following tables: *Table 6-1: Egress Configuration Options* *Table 6-2: Ingress Configuration Options* *Table 6-3: Bundle Relay Configuration Options* *Table 6-4: Bundle Originator Configuration Options* *Table 6-5: Contact Configuration Options*	Demonstration	This requirement is verified by demonstrating that the application of each supported configuration option results in a valid configuration for a UUT.
HDTNTEST-007	Start UUT	The Test Framework shall provide a method to start an instance of a UUT.	This automates the testing step of bringing up the UUT, a frequent operation.	Demonstration	This requirement is verified when a demonstration shows the Test Framework can start an instance of a UUT.
HDTNTEST-008	UUT Start Status	The Test Framework shall indicate whether the UUT started successfully.	This provides immediate feedback on the state of the UUT. A test case or test conductor may need to take action if the UUT is not able to be successfully started.	Demonstration	This requirement is verified when a demonstration shows the Test Framework reports a) the success of a UUT that initializes without failure and b) the failure of a UUT that fails to initialize.

SW Req ID	Title	Requirement	Rationale	Verification method	Verification statement
HDTNTEST-009	Identify and Report UUT Errors	The Test Framework shall identify and report errors related to the UUT process being terminated unexpectedly.	A UUT is considered terminated unexpectedly if its execution stops before being deliberately shut down by the user or before the process reaches its intended conclusion. This allows a test script or test conductor to respond to unforeseen issues.	Demonstration	This requirement is verified when a demonstration shows the Test Framework surfacing an error related to the UUT process being terminated unexpectedly.
HDTNTEST-010	Stop UUT	The Test Framework shall provide a method to stop a previously started UUT.	Stopping the UUT is necessary to reset the test environment or to gracefully shut down processes after testing.	Demonstration	This requirement is verified when a demonstration shows the Test Framework can stop a previously started UUT.
HDTNTEST-011	Get UUT Attribute	The Test Framework shall provide a method for getting supported attributes from the UUT.	Attributes from the UUT are necessary for verifying certain requirements. A list of supported attributes is listed in *Table 5-2: Attributes Supported by the Test Framework*	Test	This requirement is verified when a test shows the Test Framework can retrieve all supported attributes.
HDTNTEST-012	Poll UUT Event	The Test Framework shall provide a method for polling supported events from the UUT.	Events from the UUT are necessary for verifying certain requirements. A list of supported events is listed in *Table 5-3: Events Supported by the Test Framework.*	Demonstration	This requirement is verified when a demonstration shows the Test Framework can poll all supported events.
HDTNTEST-013	Execute Test Case Suite	The Test Framework shall provide a method for executing a suite of test cases, without user intervention.	Automating the execution of a suite of test cases efficiently facilitates the verification process for the test conductor.	Demonstration	This requirement is verified when a demonstration shows a suite of test cases being automatically executed.

SW Req ID	Title	Requirement	Rationale	Verification method	Verification statement
HDTNTEST-014	Execute Test Case Subset	The test framework shall provide a method for executing a subset of test cases.	Automating the execution of a subset of test cases provides quick feedback to test case developers. It is also necessary to properly scope regression testing. A subset of test cases could be a single test case or all the test cases in a requirement group, e.g., Bundle Protocol Version 6 (BPv6).	Demonstration	This requirement is verified when a demonstration shows: 1) A single test case being automatically executed and 2) A group of test cases being automatically executed.
HDTNTEST-015	Convert to Human Readable Representation	The Test Framework shall provide a method to convert raw data into a list of packet/bundle fields and their corresponding values.	Converting packet/bundle data to a human-readable format is essential for debugging and analyzing the results of a test case.	Test	This requirement is verified when a demonstration shows the Test Framework can convert raw data to a list of packet/bundle fields and their corresponding values.
HDTNTEST-016	Generate Test Data	The Test Framework shall provide a method for generating test data, in raw byte format, from a list of packet/bundle fields and values.	This includes malformed and nominal packets/bundles. Generating packet/bundle data automates the tedious and repetitive task of creating test data.	Test	This requirement is verified when a test shows the Test Framework can generate malformed and nominal packet/bundle data from a list of fields and their corresponding values.
HDTNTEST-017	Generation Support	The Test Framework shall support generation of test data for all supported packet/bundle formats.	A list of supported packet/bundle formats is listed in *Table 5-1: Packet/Bundle Formats Supported by the Test Framework.* These formats were chosen because they are the most complex and provide the most benefit to test script developers.	Analysis	This requirement is verified when HDTNTEST-016 is verified with all supported packet/bundle formats.

SW Req ID	Title	Requirement	Rationale	Verification method	Verification statement
HDTNTEST-018	Log Test Status	The Test Framework shall log the pass/fail status of each test case to a file.	A record of the pass/fail status of each test case is needed for verification evidence. It also provides the test conductor with insight into the testing status.	Demonstration	This requirement is verified when a demonstration shows the Test Framework logs the pass/fail status of a test case to a file.
HDTNTEST-019	Log Assertion Results	The Test Framework shall log the "Associated requirement ID", "Expected result", and "Actual result" for assertions.	A record of each assertion associated with a test case is needed for verification evidence.	Demonstration	This requirement is verified when a demonstration shows the Test Framework logs the associated requirement ID, expected result, and actual result for every assertion in a test case.
HDTNTEST-020	Save UUT Logs	The Test Framework shall store logs from the UUT in a way that ensures traceability to the corresponding test case(s).	A record of the UUT log file(s) is needed for verification evidence.	Demonstration	This requirement is verified when a demonstration shows the Test Framework stores logs from the UUT in a way that ensures traceability to the corresponding test case.
HDTNTEST-021	Log Communication	The Test Framework shall be capable of logging all information exchanged between itself and the UUT.	The Test Framework communicates with the UUT via sockets to trigger scenarios necessary for verification. A record of this communication is needed for verification evidence.	Demonstration	This requirement is verified when a demonstration shows the Test Framework stored all communication between itself and the UUT during a test case execution.
HDTNTEST-022	Conversion Support	The Test Framework shall support conversion of binary packet/bundle data to human-readable text for all supported formats.	A list of supported packet/bundle formats is listed in *Table 5-1: Packet/Bundle Formats Supported by the Test Framework.* These formats were chosen because they are the most complex and provide the most benefit to test script developers.	Analysis	This requirement is verified when HDTNTEST-015 is verified with all supported packet/bundle formats.

SW Req ID	Title	Requirement	Rationale	Verification method	Verification statement
HDTNTEST-023	Receive Timeout	The Test Framework shall abort receive operations when the specified timeout is reached.	Setting a timeout ensures a test script can verify data reception within a defined interval.	Test	This requirement is verified when a test shows the Test Framework a) successfully receives a packet that arrives within the specified timeout period and b) fails to receive a packet that arrives after the specified timeout period.
HDTNTEST-024	Receive Buffer Size	The Test Framework shall return all available data that fits within the specified buffer size for each receive operation.	Defining a buffer size allows a test script to receive only the desired amount of data, supporting data formatting needs.	Test	This requirement is verified when a test shows the Test Framework a) receives a complete packet when the packet size is within the specified buffer size and b) receives only the portion of a packet that fits within the specified buffer size when the packet size exceeds the specified buffer size.

4.0 Design Decisions

The primary goal of the HDTN Test Framework is to provide a simple library for test script developers, automating the most common and time-consuming tasks associated with testing the HDTN software suite. To achieve this goal, the framework must be adaptable and capable of integrating with a wide range of test cases. Additionally, the Test Framework itself must be easily verifiable, so that project resources can be concentrated on verifying the HDTN software suite, the actual software under test. A secondary goal is to make the Test Framework extensible for testing other DTN implementations besides HDTN.

The sections below describe the specific decisions associated with the design of the Test Framework.

4.1 Programming Language: Python

Python was chosen as the programming language for the HDTN Test Framework due to its extensive library support and ease of use.

4.2 Operating System Support

The Test Framework is designed to run on the Ubuntu 20.04 operating system (OS), which is the platform for HDTN verification. Support for other operating systems can be incorporated as needed to meet developer needs.

4.3 Packaging

The Test Framework is packaged as Python modules to allow for easy integration with individual test scripts via function calls. This design decision was made to simplify the process of writing test scripts.

4.4 Scapy

The Scapy library was chosen to provide network protocol support for the Test Framework. Scapy is a powerful packet manipulation tool, written in Python, that allows for the creation, manipulation, and decoding of network packets. Scapy's extensible interface was used to implement BPv6, Bundle Protocol Version 7 (BPv7), and Transmission Control Protocol Convergence Layer (TCPCL) support. Although Scapy provides some built-in support for BPv6, it is incomplete. Scapy does provide complete support for Licklider Transmission Protocol (LTP). All of these protocols are important in any DTN implementation. Scapy is released under the GNU General Public License 2.0 (GPL-2.0) license.

4.5 Pytest

Pytest was chosen to facilitate test case execution. Pytest is a mature testing framework that provides a wide range of features. Pytest is widely used in the Python community and is well documented. Pytest is released under the MIT license.

5.0 Architectural Design

5.1 Description

The HDTN Test Framework is a single CSCI whose main purpose is to provide test script automation, assisting both the test conductor and the developers writing scripts for requirements verification. It consists of the following Computer Software Components (CSC's):

1. **DTN Test Module**—the DTN Test Module provides an interface to the UUT, managing tasks such as launching DTN applications as subprocesses, configuring them, and stopping them after the relevant test cases are complete. It also establishes socket connections to the DTN applications, enabling test scripts to exercise the DTN application's functionality.
2. **Protocols Module**—the Protocols Module, built using the Scapy library, offers functionality for creating, manipulating, and decoding Bundle Protocol and Convergence Layer packet formats. Test scripts use these protocols to generate test data and verify the DTN application outputs.
3. **Test Executor**—the Test Executor is used by the Test Conductor to initiate, manage, and record the testing process, utilizing Pytest to accomplish these functions.

A detailed description of each CSC is available in section 6.0 CSC Detailed Design. Figure 5-1 illustrates the Test Framework architecture.

5.2 Concept of Execution

The execution of the Test Framework varies based on the needs of each test case, but generally follows a common pattern. Initially, a test script is prepared that contains function calls to the Test Framework. Then, the process begins when the test conductor commands the Test Framework to execute the tests. Figure 5-2 illustrates the concept of execution, which is then explained step by step.

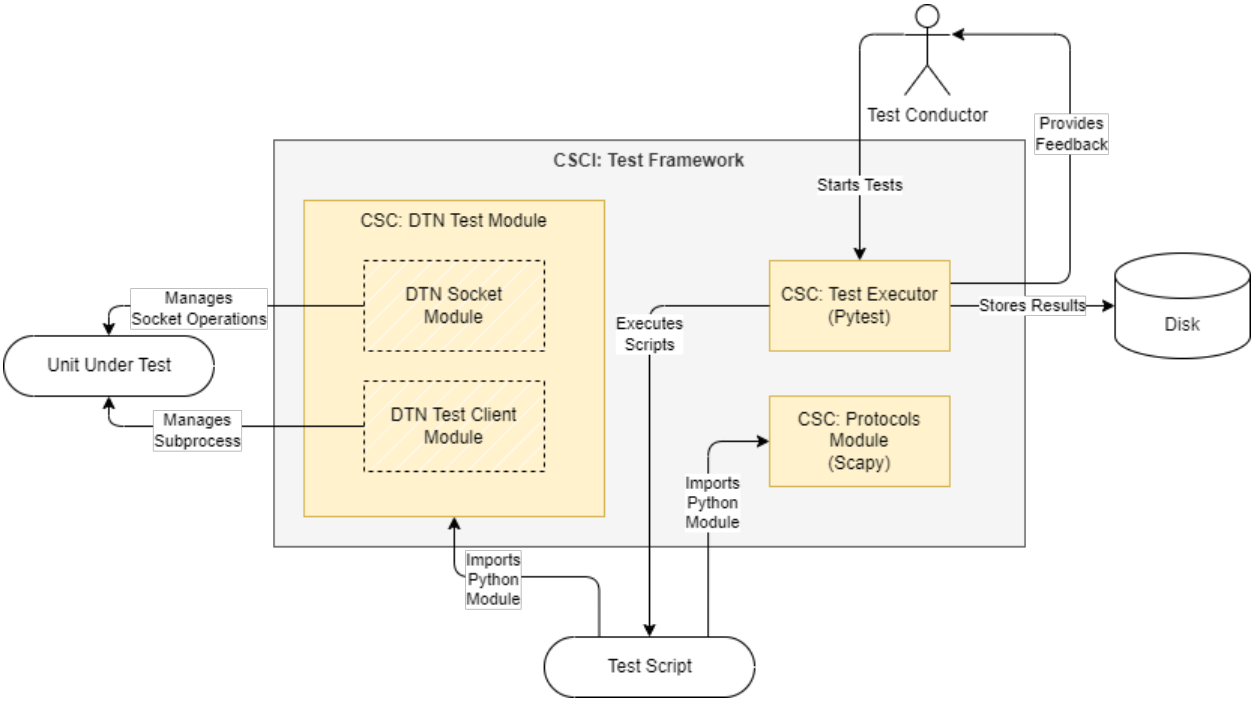

Figure 5-1.—Test Framework Architecture.

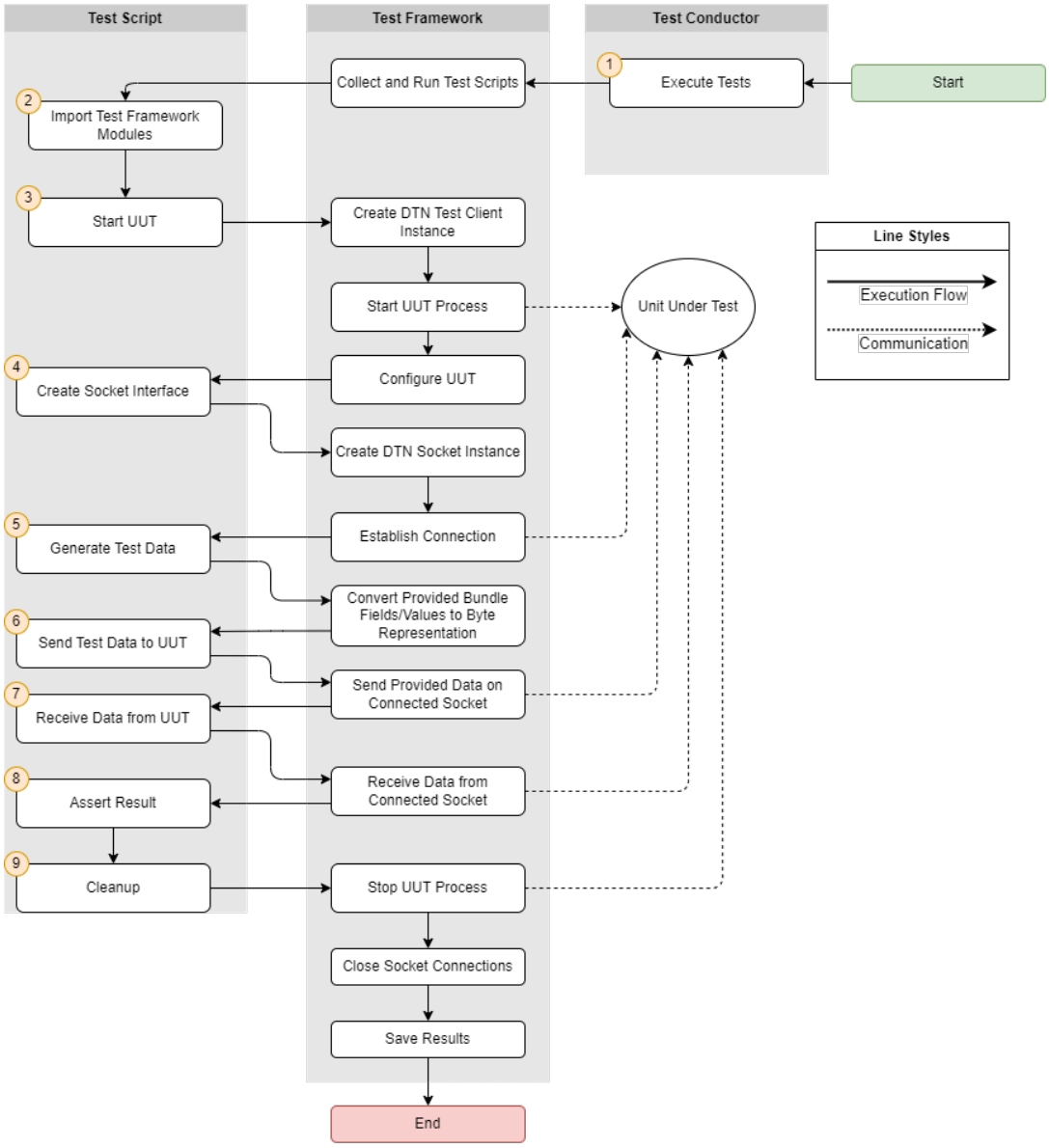

Figure 5-2.—Concept of Execution: Sending and Receiving Data with the UUT.

1. The Test Framework is responsible for collecting and executing all test scripts. The test conductor initiates this process either through a command-line interface or by launching a GitLab Continuous Integration (CI) pipeline.
2. The test script imports components of the Test Framework as Python modules, which provide the necessary functionality for the subsequent steps.
3. The test script calls a function to instruct the framework to start the UUT. The Test Framework initiates the UUT as a subprocess and captures a reference to monitor and eventually terminate it. As part of this step, the Test Framework also configures the UUT for the test scenario.
4. The test script calls a function to establish a socket connection with the UUT. This connection enables the transmission of application data to the UUT for exercising and verifying the UUT's functionality. The framework creates a "DTN Socket" instance, responsible for establishing and managing a Berkeley Software Distribution (BSD) or Winsock socket connection to the UUT.

5. The test script calls a function to generate test data. The framework's Protocol library converts a field/value representation of a bundle into a byte stream that can be transmitted over the socket.
6. The test script sends the test data to the UUT through a function call. The framework manages the network operations to deliver the data to the UUT.
7. The test script receives data from the UUT via a function call. The framework handles the network operations for reading the data from the socket connection and then returns the data to the test script.
8. The test script performs an assertion to verify that the received data matches the expected result. The Test Framework tracks this assertion for reporting.
9. Once the test script completes execution, the Test Framework stops the UUT, closes the socket connection, and saves the test results.

5.3 Interface Design

Figure 5-3 shows how the Test Framework interfaces with test scripts and test environment.

The Test Scripts interact with the Test Framework through Python functions calls. Specifically, they call functions within the Protocols module to generate test data and use the DTN Test Module for actions that engage with the UUT and the Test Framework, as well as for collecting results from those interactions. The data formats supported by the Protocols module are in Table 5-1.

The Test Conductor uses a command-line interface (CLI) to interact with the Test Executor, running tests and monitoring their status. The Test Executor interfaces with the file system via OS calls, gathering test scripts to execute and storing test artifacts during execution.

The DTN Test Module communicates with the UUT through OS calls, which includes sending and receiving transport layer packets, configuring, starting, stopping, and monitoring the UUT, as well as capturing logs, events, and attributes from the UUT. A list of supported events, attributes, and transport protocols are in Table 5-2, Table 5-3, and Table 5-4.

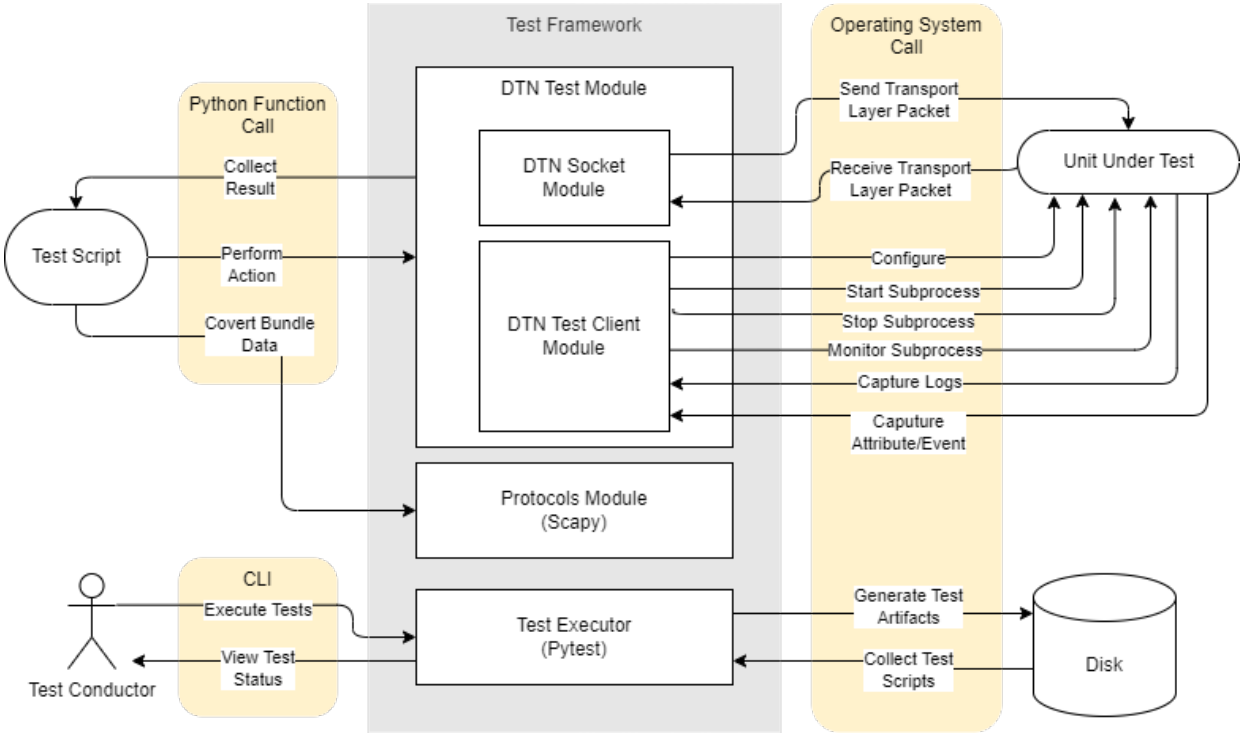

Figure 5-3.—Test Framework Interfaces.

TABLE 5-1.—PACKET/BUNDLE FORMATS SUPPORTED BY THE TEST FRAMEWORK

Packet/Bundle format name
BPv6
BPv7
LTP
TCPCL

TABLE 5-2.—ATTRIBUTES SUPPORTED BY THE TEST FRAMEWORK

Attribute name	Attribute description
Bundle in Storage	Gets whether a sequence of bundle bytes is in storage on the UUT. Only applicable to UUTs that are full DTN implementations.

TABLE 5-3.—EVENTS SUPPORTED BY THE TEST FRAMEWORK

Event name	Event description
Discarded LTP Segment	Triggered when the UUT receives an LTP segment that cannot be processed and discards it

TABLE 5-4.—TRANSPORT PROTOCOLS SUPPORTED BY THE TEST FRAMEWORK

Transport protocol name
Transmission Control Protocol (TCP)
User Datagram Protocol (UDP)

6.0 CSC Detailed Design

6.1 DTN Test Module

The DTN Test Module consists of two sub-modules, the DTN Socket Module and the DTN Test Client Module.

The DTN Socket module is a Python class that interfaces with the UUT to enable the sending and receiving of application data using BSD/Winsock sockets over the UDP and TCP transport protocols. It provides a function for a test script to create a socket connection to a specified port, and once the connection is established, offers functions for sending and receiving data. The DTN Socket module is responsible for maintaining the socket connection throughout the testing.

The DTN Test Client module is a Python class that provides a standardized interface to the UUT, abstracting the details of the underlying DTN implementation. The Test Framework defines a standard set of configuration options and methods for initializing and interacting with the UUT, enabling easy swapping between different implementations. To integrate a new DTN implementation into the framework, developers can extend the DTN Test Client class and implement support for all necessary configuration options, events, and attributes. Figure 6-1 illustrates the DTN Test Client Interface.

The list of the configuration options supported by the DTN Test Client Interface is in Table 6-1, Table 6-2, Table 6-3, and Table 6-4.

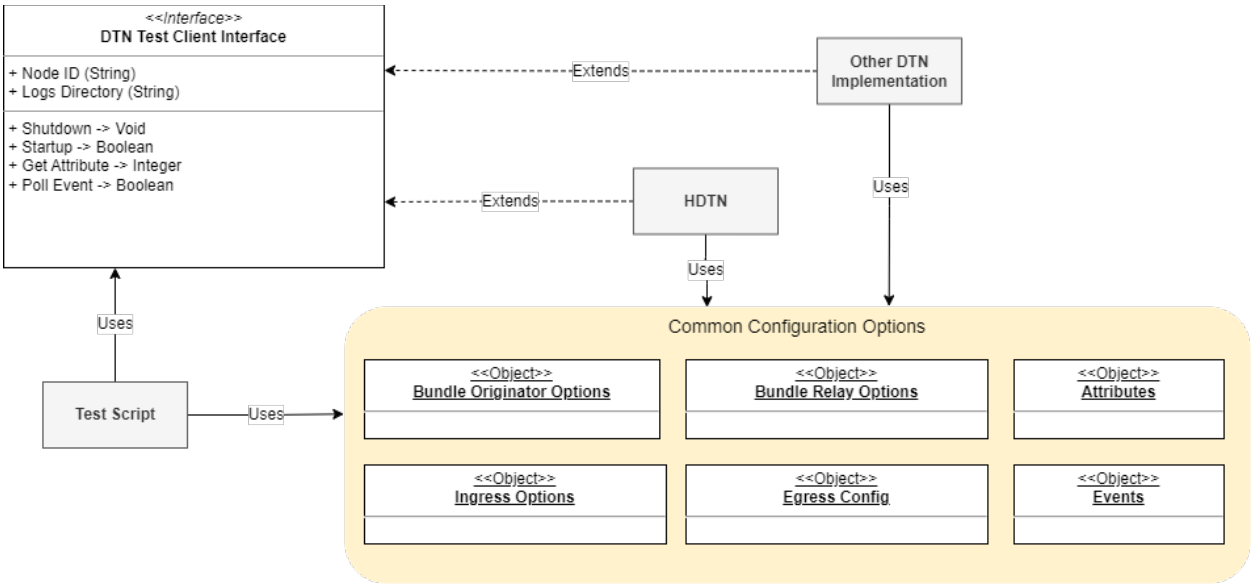

Figure 6-1.—Usage of the DTN Test Client Interface.

TABLE 6-1.—EGRESS CONFIGURATION OPTIONS

Option name	Data type	Scope
Convergence Layer Adapter (CLA)	String	----------
Destination Port	Integer	----------
Next Hop ID	Integer	----------
Data Segment Maximum Transmission Unit	Integer	LTP only
Bound Port	Integer	LTP only
This Engine ID	Integer	LTP only
Remote Engine ID	Integer	LTP only
Retry Limit	Integer	LTP only
Checkpoint Interval	Integer	LTP only
Retransmit Time	Integer	LTP only
Transmit Rate	Integer	LTP or UDPCL only

TABLE 6-2.—INGRESS CONFIGURATION OPTIONS

Option name	Data type	Scope
Convergence Layer Adapter (CLA)	String	----------
Bound Port	Integer	----------
Next Hop ID	Integer	----------
Report Maximum Transmission Unit	Integer	LTP only
Destination Port	Integer	LTP only
This Engine ID	Integer	LTP only
Remote Engine ID	Integer	LTP only
Retransmit Time	Integer	LTP only

TABLE 6-3.—BUNDLE RELAY CONFIGURATION OPTIONS

Option name	Data type
Node ID	Integer
Fragmentation Size	Integer
Custody Retransmit Time	Integer
Aggregate Custody Signal Aware	Boolean
Storage Capacity	Integer

TABLE 6-4.—BUNDLE ORIGINATOR CONFIGURATION OPTIONS

Option name	Data type
Bundle Protocol Version	Integer
Application Data Port	Integer
Source Node ID	Integer
Destination Node ID	Integer
Bundle Lifetime	Integer

TABLE 6-5.—CONTACT CONFIGURATION OPTIONS

Option name	Data type
Source Node ID	Integer
Destination Node ID	Integer
Start Time	Integer
End Time	Integer

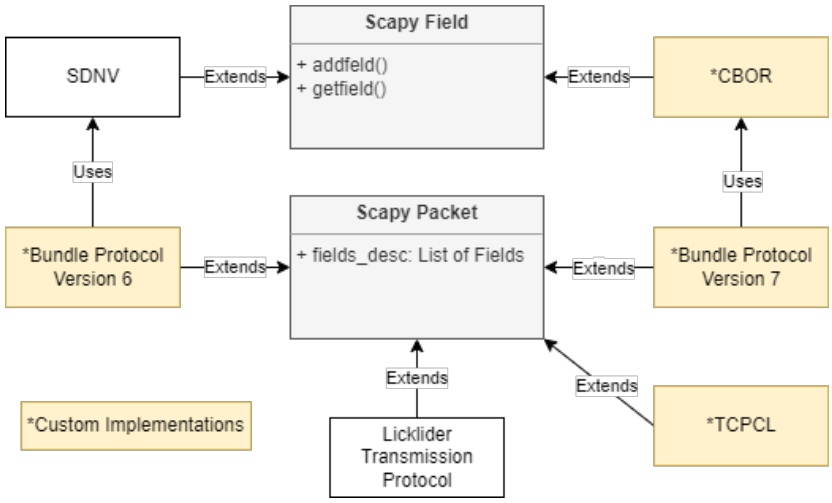

Figure 6-2.—Scapy Packet and Field Relationships.

6.1.1 Protocols Module

The Protocols Module includes a set of Python classes that implement Scapy packet definitions for Bundle Protocol Version 6, Bundle Protocol Version 7, and LTP. In Scapy, a packet is a data structure that defines the content of a network packet in a human-readable format, which can then be converted to its binary form for transmission or analysis. A packet in Scapy can consist of multiple layers, with each layer representing a different network protocol (e.g. Ethernet/Internet Protocol/TCP). These layers are made up of Scapy Fields, which specify the data elements of a protocol and how they are converted to and from binary format. Scapy's design is extensible, allowing users to define custom packet and field types using class inheritance. To define packets, the "fields_desc" member variable is overridden to list the fields in the packet. For fields, the "addfield" and "getfield" methods are overridden to handle the conversion between human-readable and binary formats.

The Scapy library already includes an LTP packet type and a Self-Delimiting Numeric Value (SDNV) field type. However, Bundle Protocol Version 6, Bundle Protocol Version 7, TCPCL, and Concise Binary Object Representation (CBOR) are custom implementations in the Test Framework. Figure 6-2 shows the relationship between Scapy packet and field types.

6.1.2 Test Executor

The Test Executor is Pytest, an open-source testing framework for Python. Pytest consists of two main components: an executable and a Python library. The executable is invoked by the Test Conductor from the command line to start the testing process. It handles test case discovery, execution, and reporting. The Python library is utilized by test case developers to structure and organize test cases. Pytest offers various testing features, such as fixtures, which provide a reliable and consistent context for setting up test environments. Fixtures are particularly useful for preparing a shared setup that can be accessed by multiple test cases.

7.0 Requirements Traceability

The verification methods are documented in the HDTN-PLAN-022 Software Verification and Validation Plan. The Requirements Traceability Matrix is currently maintained in the HDTN MagicDraw project.

8.0 CSCI Implementation Plan

The Test Framework CSCI has been implemented as part of the development of the HDTN project. It will undergo its own formal verification and once verified, be used in the verification of the HDTN software suite.

Appendix—Acronyms

The following list contains definitions for all abbreviations and acronyms used in this document.

BSD	Berkley Software Distribution
BPv6	Bundle Protocol Version 6
BPv7	Bundle Protocol Version 7
CAGE	Contract and Government Entity
CBOR	Concise Binary Object Representation
CI	Continuous Integration
CLA	Convergence Layer Adapter
CLI	Command Line Interface
CM	Configuration Management
CSC	Computer Software Component
CSCI	Computer Software Configuration Item
DTN	Delay Tolerant Networking
GRC	NASA John H. Glenn Research Center
HDTN	High-Rate Delay Tolerant Networking
IP	Internet Protocol
LTP	Licklider Transmission Protocol
MTU	Maximum Transmission Unit
NASA	National Aeronautics and Space Administration
NPR	NASA Procedural Requirements
OS	Operating System
SCaN	Space Communications and Navigation
SDD	Software Design Document
SDNV	Self-Delimiting Numeric Value
SOMD	Space Operations Mission Directorate
TCP	Transmission Control Protocol
TCPCL	Transmission Control Protocol Convergence Layer
UDP	User Datagram Protocol
UUT	Unit Under Test