Consultative Committee for Space Data Systems

DRAFT REPORT CONCERNING SPACE DATA SYSTEM STANDARDS

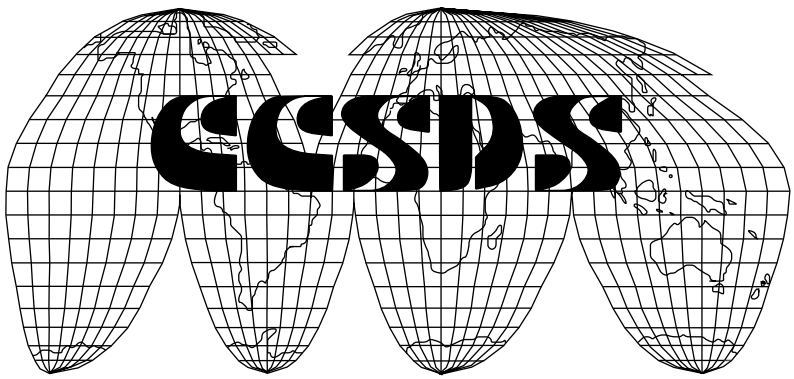
CCSDS FILE DELIVERY PROTOCOL (CFDP)—

Part 2
Implementers Guide

CCSDS 720.2-G-0.9

DRAFT GREEN BOOK

July 2001



AUTHORITY

Issue:	Draft Green Book, Issue 0.9
Date:	July 2001
Location:	N/A

(WHEN THIS RECOMMENDATION IS FINALIZED, IT WILL CONTAIN THE FOLLOWING STATEMENT OF AUTHORITY:)

This document has been approved for publication by the Management Council of the Consultative Committee for Space Data Systems (CCSDS) and reflects the consensus of technical panel experts from CCSDS Member Agencies. The procedure for review and authorization of CCSDS Reports is detailed in reference [2].

This document is published and maintained by:

CCSDS Secretariat
Program Integration Division (Code MT)
National Aeronautics and Space Administration
Washington, DC 20546 USA

FOREWORD

This document is a CCSDS Report, which contains background and explanatory material to support the CCSDS Recommendation, *CCSDS File Delivery Protocol* (reference [1]).

Through the process of normal evolution, it is expected that expansion, deletion, or modification to this Report may occur. This Report is therefore subject to CCSDS document management and change control procedures, which are defined in reference [2]. Current versions of CCSDS documents are maintained at the CCSDS Web site:

http://www.ccsds.org/

Questions relating to the contents or status of this report should be addressed to the CCSDS Secretariat at the address on page i.

At time of publication, the active Member and Observer Agencies of the CCSDS were

Member Agencies

- Agenzia Spaziale Italiana (ASI)/Italy.
- British National Space Centre (BNSC)/United Kingdom.
- Canadian Space Agency (CSA)/Canada.
- Centre National d'Etudes Spatiales (CNES)/France.
- Deutsches Zentrum für Luft- und Raumfahrt e.V. (DLR)/Germany.
- European Space Agency (ESA)/Europe.
- Instituto Nacional de Pesquisas Espaciais (INPE)/Brazil.
- National Aeronautics and Space Administration (NASA)/USA.
- National Space Development Agency of Japan (NASDA)/Japan.
- Russian Space Agency (RSA)/Russian Federation.

Observer Agencies

- Austrian Space Agency (ASA)/Austria.
- Central Research Institute of Machine Building (TsNIIMash)/Russian Federation.
- Centro Tecnico Aeroespacial (CTA)/Brazil.
- Chinese Academy of Space Technology (CAST)/China.
- Commonwealth Scientific and Industrial Research Organization (CSIRO)/Australia.
- Communications Research Centre (CRC)/Canada.
- Communications Research Laboratory (CRL)/Japan.
- Danish Space Research Institute (DSRI)/Denmark.
- European Organization for the Exploitation of Meteorological Satellites (EUMETSAT)/Europe.
- European Telecommunications Satellite Organization (EUTELSAT)/Europe.
- Federal Service of Scientific, Technical & Cultural Affairs (FSST&CA)/Belgium.
- Hellenic National Space Committee (HNSC)/Greece.
- Indian Space Research Organization (ISRO)/India.
- Institute of Space and Astronautical Science (ISAS)/Japan.
- Institute of Space Research (IKI)/Russian Federation.
- KFKI Research Institute for Particle & Nuclear Physics (KFKI)/Hungary.
- MIKOMTEK: CSIR (CSIR)/Republic of South Africa.
- Korea Aerospace Research Institute (KARI)/Korea.
- Ministry of Communications (MOC)/Israel.
- National Oceanic & Atmospheric Administration (NOAA)/USA.
- National Space Program Office (NSPO)/Taipei.
- Swedish Space Corporation (SSC)/Sweden.
- United States Geological Survey (USGS)/USA.

DOCUMENT CONTROL

Document	Title	Date	Status
CCSDS 720.7-G-0.3	CCSDS File Delivery Protocol (CFDP)—Summary of Concept and Rationale	August 1998	Superseded.
CCSDS 720.7-G-0.4	CCSDS File Delivery Protocol (CFDP)—Summary of Concept and Rationale	April 1999	Superseded.
CCSDS 720.2-G-0.5	CCSDS File Delivery Protocol (CFDP)—Part 2: Implementers Guide	July 1999	Original draft Green Book divided into two documents (parts 1 and 2); superseded.
CCSDS 720.2-G-0.6	CCSDS File Delivery Protocol (CFDP)—Part 2: Implementers Guide	May 2000	Current draft. Draft Green Book part 2 divided into two documents (parts 2 and 3); superseded.
CCSDS 720.2-G-0.7	CCSDS File Delivery Protocol (CFDP)—Part 2: Implementers Guide	August 2000	Superseded.
CCSDS 720.2-G-0.8	CCSDS File Delivery Protocol (CFDP)—Part 2: Implementers Guide	February 2001	Superseded.
CCSDS 720.2-G-0.9	CCSDS File Delivery Protocol (CFDP)—Part 2: Implementers Guide	July 2001	Current draft.

CONTENTS

Table

1 INTRODUCTION

1.1 PURPOSE

This report is an adjunct document to the Consultative Committee for Space Data Systems (CCSDS) Recommendation for File Delivery Protocol (reference [1]). It contains material which will be helpful in understanding the primary document, and which will assist decision makers and implementers in evaluating the applicability of the protocol to mission needs and in making implementation, option selection, and configuration decisions related to the protocol.

1.2 SCOPE

This report provides supporting descriptive and tutorial material**. This document is not part of the Recommendation.** In the event of conflicts between this report and the Recommendation, the Recommendation shall prevail.

1.3 ORGANIZATION OF THIS REPORT

This report is divided into two parts. Part 1 (reference [3]) provides an introduction to the concepts, features and characteristics of the CCSDS File Delivery Protocol (CFDP). It is intended for an audience of persons unfamiliar with the CFDP or related protocols. The second part of this report (this document) is an implementers guide. It provides information to assist implementers in understanding the details of the protocol and in the selection of appropriate options, and contains suggestions and recommendations about implementation-specific subjects. This document also contains implementation reports from various member Agencies, reports on testing of the implementations and protocol, and the requirements upon which the CFDP is based.

1.4 CONVENTIONS AND DEFINITIONS

1.4.1 BIT NUMBERING CONVENTION AND NOMENCLATURE

In this document, the following convention is used to identify each bit in an N-bit field. The first bit in the field to be transmitted (i.e., the most left-justified when drawing a figure) is defined to be ‘Bit 0’; the following bit is defined to be ‘Bit 1’, and so on up to ‘Bit N-1’. When the field is used to express a binary value (such as a counter), the Most Significant Bit (MSB) shall be the first transmitted bit of the field, i.e., ‘Bit 0’, as shown in figure 1-1.

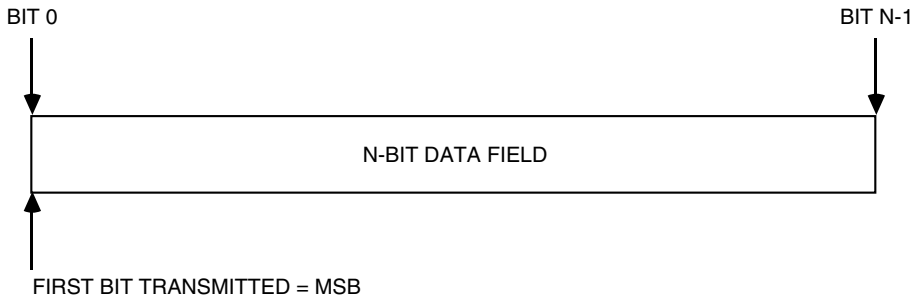


Figure 1-1: Bit Numbering Convention

In accordance with modern data communications practice, spacecraft data fields are often grouped into 8-bit 'words' which conform to the above convention. Throughout this Report, the nomenclature shown in figure 1-2 is used to describe this grouping.

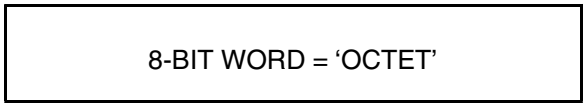


Figure 1-2: Octet Convention

By CCSDS convention, all 'spare' bits shall be permanently set to value 'zero'.

1.4.2 DEFINITIONS

Within the context of this document the following definitions apply:

A *file* is a bounded or unbounded named string of octets that resides on a storage medium.

A *filestore* is a system used to store files; CFDP defines a standard *virtual filestore* interface through which CFDP accesses a filestore and its contents.

A *CFDP protocol entity* (or *CFDP entity*) is a functioning instance of an implementation of the CFDP protocol, roughly analogous to an Internet protocol 'host'. Each CFDP entity has access to exactly one filestore. (It is recognized that the single [logical] filestore of a CFDP entity might encompass multiple physical storage partitions, but any specific reference to such a partition in identifying the location or destination of a file is expected to be encoded as part of the file's name [e.g., 'pathname']. Each entity also maintains a *Management Information Base (MIB),* which contains such information as default values for user communications requirements (e.g., for address mapping, and for communication timer settings).

The functional concatenation of a file and related *metadata* is termed a *File Delivery Unit* (FDU); in this context the term 'metadata' is used to refer to any data exchanged between CFDP protocol entities in addition to file content, typically either additional application data

(such as a 'message to user') or data that aid the recipient entity in effectively utilizing the file (such as file name).

NOTES

1 An FDU may consist of metadata only.

2 The term 'file' is frequently used in this specification as an abbreviation for 'file delivery unit'; only when the context clearly indicates that actual files are being discussed should the term 'file' not be read as 'file delivery unit.' For example, in the explanation of the record type parameter or the source and destination file name parameters of the CFDP Service Definition, the term 'file' should not be read as 'file delivery unit'.

The individual, bounded, self-identifying items of CFDP data transmitted between CFDP entities are termed *CFDP Protocol Data Units* (PDU), or *CFDP PDU*s. Unless otherwise noted, in this document the term 'PDU' always means 'CFDP PDU'. CFDP PDUs are of two general types: *File Data PDUs,* which convey the contents of the files being delivered, and *File Directive PDUs*, which convey only metadata and other non-file information that advances the operation of the protocol.

A *transaction* is the end-to-end transmission of a single FDU between two CFDP entities. A single transaction normally entails the transmission and reception of multiple PDUs. Each transaction is identified by a unique transaction ID; all elements of any single FDU, both file content and metadata, are tagged with the same CFDP transaction ID.

Any single end-to-end file transmission task has two associated entities: the *source* and the *destination*. The source is the entity that has the file at the beginning of the task. The destination is the entity that has a copy of the file when the task is completed.

Each end-to-end file transmission task comprises a point-to-point file copy operation. Any single point-to-point file copy operation has two associated entities: the *sender* and the *receiver*. The sender is the entity that has a copy of the file at the beginning of the operation. The receiver is the entity that has a copy of the file when the operation is completed. (In the current CFDP, the only sender of the file is the source and the only receiver is the destination. However, in more complex cases such as those discussed in annex B of this document and in annex D of reference [1], there are additional '*waypoint*' entities that receive and send copies of the file. The source is the first sender, and the destination is the last receiver. The terminology of both source/destination and sender/receiver pairs is retained, since the more complex case is for further study.)

The term *CFDP user* refers to the software task that causes the local entity to initiate a transaction, or the software task that is notified by the local entity of the progress or completion of a transaction. The CFDP user local to the source entity is referred to as the *source CFDP user*. The CFDP user local to the destination entity is referred to as the

destination CFDP user. The CFDP user may be operated by a human or by another software process. Unless otherwise noted, the term *user* always refers to the CFDP user.

A *message to user* (or *user message*) allows delivery of information related to a transaction to the destination user in synchronization with the transaction.

A *filestore request* is a request to the remote filestore for service (such as creating a directory, deleting a file, etc.) at the successful completion of a transaction.

Service primitives form the software interface between the CFDP user and its local entity. The user issues *request* service primitives to the local entity to request protocol services, and the local entity issues *indication* service primitives to the user to notify it of the occurrence of significant protocol events.

1.5 REFERENCES

The following documents are referenced in the text of this Report. At the time of publication, the editions indicated were valid. All documents are subject to revision, and users of this Report are encouraged to investigate the possibility of applying the most recent editions of the documents indicated below. The CCSDS Secretariat maintains a register of currently valid CCSDS Recommendations.

[1] *CCSDS File Delivery Protocol (CFDP)*. Draft Recommendation for Space Data System Standards, CCSDS 727.0-R-4. Red Book. Issue 5. Washington, D.C.: CCSDS, July 2001.

[2] *Procedures Manual for the Consultative Committee for Space Data Systems*. CCSDS A00.0-Y-7. Yellow Book. Issue 7. Washington, D.C.: CCSDS, November 1996.

[3] *CCSDS File Delivery Protocol—Part 1: Introduction and Overview*. Draft Report Concerning Space Data Systems Standards, CCSDS 720.1-G-0.8. Draft Green Book. Issue 0.8. Washington, D.C.: CCSDS, July 2001.

[4] *CCSDS File Delivery Protocol—Part 2: Implementers Guide*. Draft Report Concerning Space Data Systems Standards, CCSDS 720.2-G-0.9. Draft Green Book. Issue 0.9. Washington, D.C.: CCSDS, July 2001.

2 CFDP PROTOCOL DATA UNITS

2.1 GENERAL

This section presents the formats of the CFDP Protocol Data Units (PDU), as well as the relationships between the PDUs and the CFDP primitives. PDUs are exchanged between CFDP entities and, therefore, both their contents and their formats are defined. Primitives are not exchanged between protocol entities and, therefore, their contents are defined but their formats are not.

The information in this section is provided as an aid to visualizing and understanding the primitives and PDUs, and their relationships. In all cases more detail, and the protocol specifications and procedures, are found in reference [1]. As always, reference [1] is the defining document and in case of any disagreements between it and this Report, reference [1] is the authoritative document.

All PDUs consist of two components: the Fixed PDU Header and the PDU Data Field.

Two PDU types are defined: File Directive and File Data. The PDU type is signaled in the PDU Type field of the Fixed PDU Header, as shown in table 2-1 and subsection 2.2.

Table 2-1: PDU Type Code

Field	Values
PDU type	'0' - File Directive '1' - File Data

The format of the data field of File Data PDUs, which are the PDUs used to deliver the actual file data, is shown in 2.3.2.

The data field of File Directive PDUs consists of a Directive Code octet followed by a Directive Parameter field. The File Directive Codes are shown in table 2-2. The formats of each of the different file directive PDUs are shown in subsections 2.3 through 2.5.

Table 2-2: File Directive Codes

Directive Code (hexadecimal)	Action
00	Reserved
01	Reserved
02	Reserved
03	Reserved
04	EOF PDU
05	Finished PDU
06	ACK PDU
07	Metadata PDU
08	NAK PDU
09	Prompt (NAK) PDU
0C	Keep Alive PDU
0A, 0B, and 0D - FF	Reserved

The relationships between primitives and PDUs are shown in figure 2-1. The figure also shows the relationships of the primitives and PDUs to the operational process from initiation through termination. The MIB is shown on the diagram since its (minimum) contents are defined in the CFDP, and some of those contents are necessary to complete the Metadata PDU initiated by the Put Request. The format of each of the PDUs is presented in the remainder of this section.

In several cases, the Directive Parameter field of a File Directive includes a four-bit Condition Code. The Condition Code shall in each case indicate one of the conditions shown in table 2-3.

Table 2-3: Condition Codes

Condition Code (binary)	Condition
0000	No error
0001	Positive ACK limit reached
0010	Keep alive limit reached
0011	Invalid transmission mode
0100	Filestore rejection
0101	File checksum failure
0110	File size error
0111	NAK limit reached
1000	Inactivity detected
1001	Invalid file structure
1010 – 1101	(reserved)
1110	Suspend.request received
1111	Cancel.request received

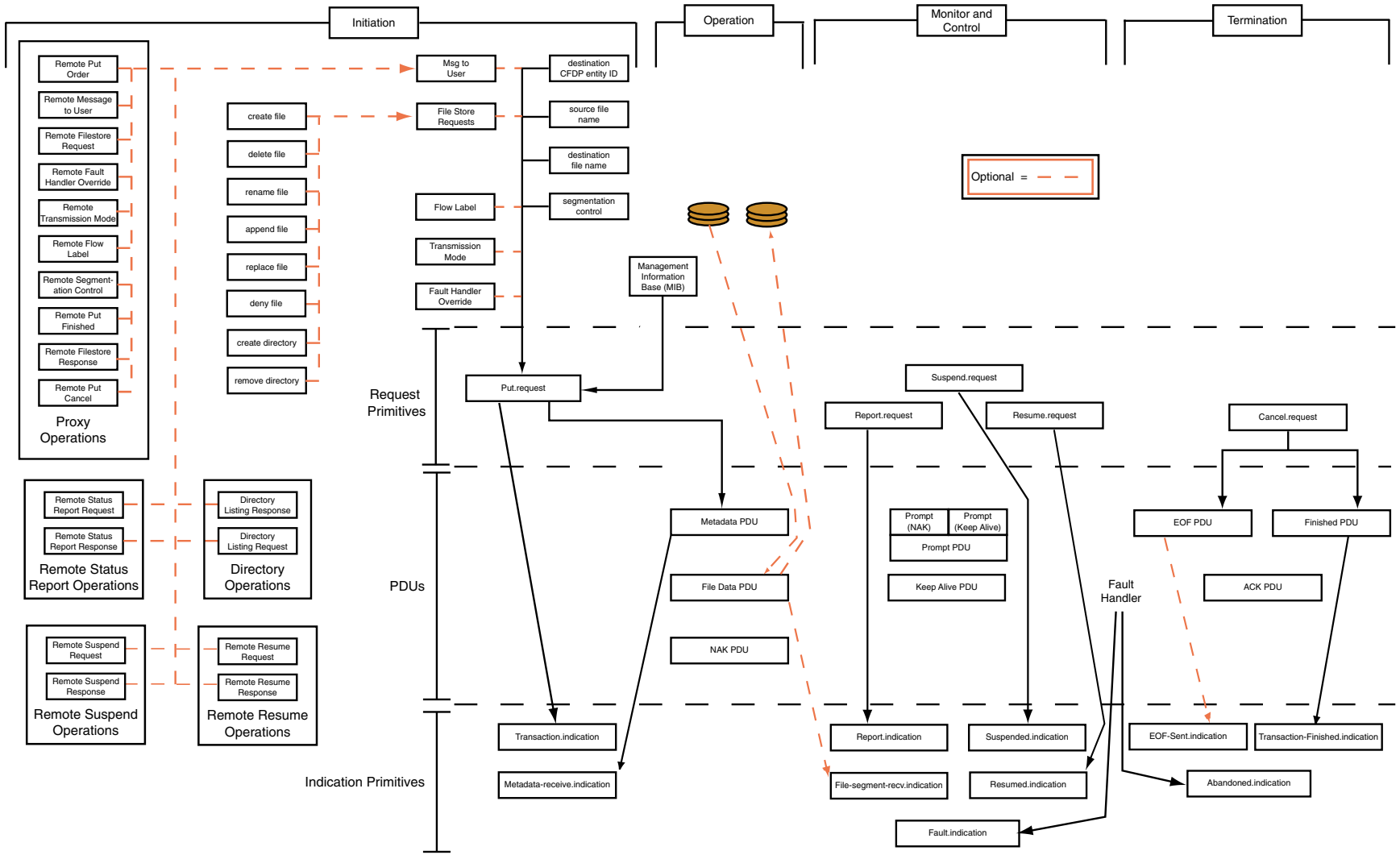


Figure 2-1: Operations View

2.2 FIXED PDU HEADER

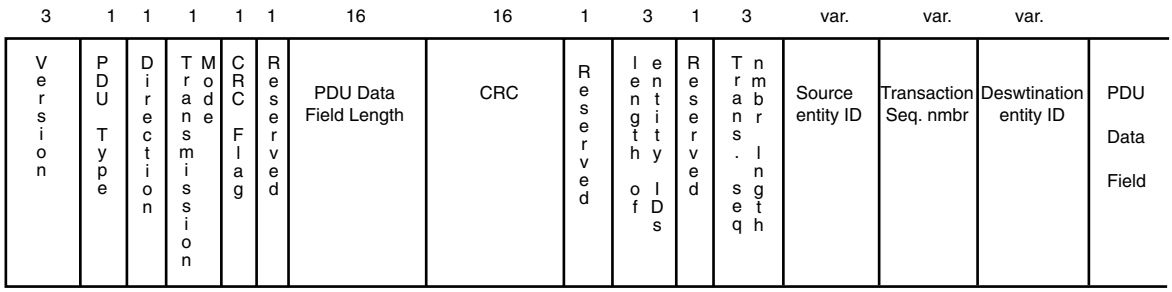
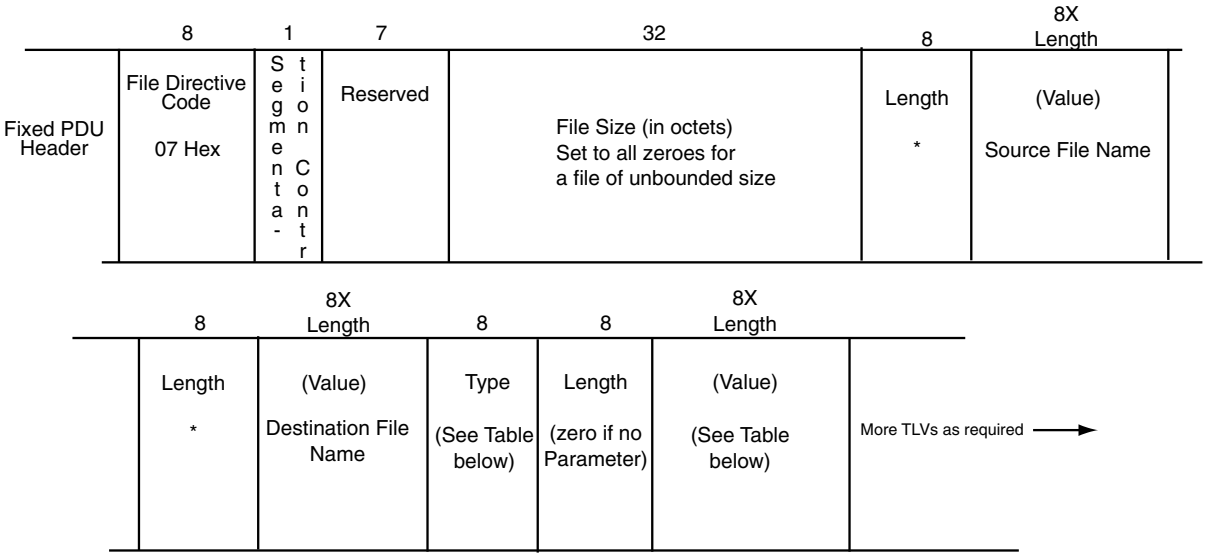


Table 2-4: Fixed PDU Header Fields

Field	Length (bits)	Values	Comment
Version	3	'000'	For the first version
PDU type	1	'0' — File Directive '1' — File Data	
Direction	1	'0' — toward file receiver '1' — toward file sender	Used to perform PDU forwarding
Transmission Mode	1	'0' — acknowledged '1' — unacknowledged	
CRC Flag	1	'0' — CRC not present '1' — CRC present	
Reserved for future use	1	set to '0'	
PDU Data field length	16		In octets
CRC	16		Optional
Reserved for future use	1	set to '0'	
Length of entity IDs	3		Number of octets in entity ID less one; i.e., '0' means that entity ID is one octet. Applies to all entity IDs in the PDU header.
Reserved for future use	1	set to '0'	
Length of Transaction sequence number	3		Number of octets in sequence number less one; i.e., '0' means that sequence number is one octet.
Source entity ID	variable		Uniquely identifies the entity that originated the transaction.
Transaction sequence number	variable		Uniquely identifies the transaction, among all transactions originated by this entity.
Destination entity ID	variable		Uniquely identifies the entity that is the final destination of the transaction's metadata and file data.

2.3 OPERATION PDUs

2.3.1 METADATA PDU



* LV Length field indicates zero length and LV value field omitted when there is no associated file, e.g. messages used for Proxy operations

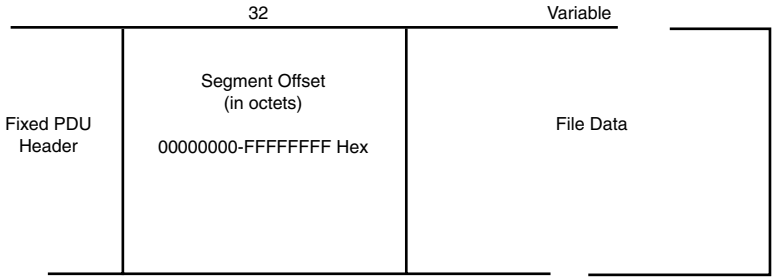
Table 2-5: Metadata Segmentation Control Field Contents

Segmentation Control
'0' - Record boundaries respected
'1' - Record boundaries not respected

Table 2-6: Metadata Type-Length-Value (TLV) Field Codes

Type Field Code	Contents of Value Field
00 Hex	Filestore Request
02 Hex	Message to User
04 Hex	Fault Handler Overrides
05 Hex	Flow Label

2.3.2 FILE DATA PDU



2.3.3 NEGATIVE ACKNOWLEDGMENT (NAK) PDU

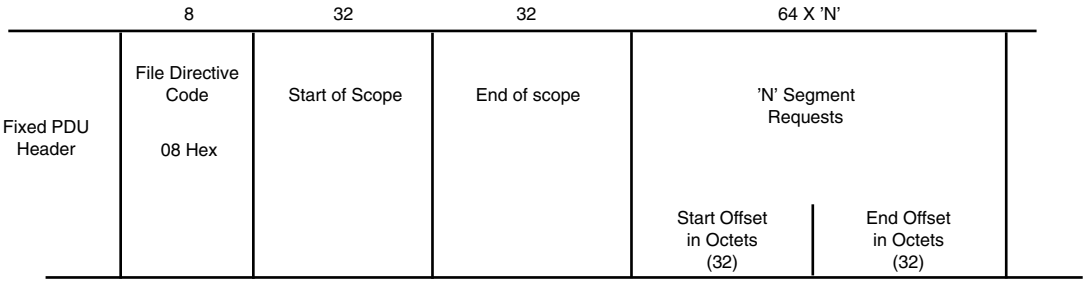


Table 2-7: Segment Request Form

Parameter	Length (bits)	Values	Comments
Start offset	32	Data — Offset of start of requested segment Metadata — 00000000 (hex)	In octets
End Offset	32	Data — Offset of first octet after end of requested segment Metadata — 00000000 (hex)	In octets

2.4 MONITOR AND CONTROL PDUs

2.4.1 PROMPT PDU

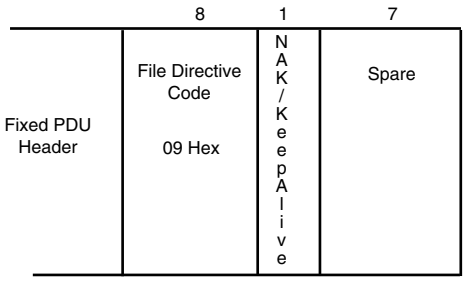
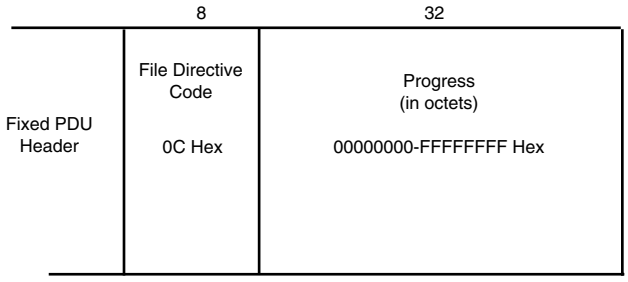


Table 2-8: Prompt PDU NAK/Keep Alive Field Contents

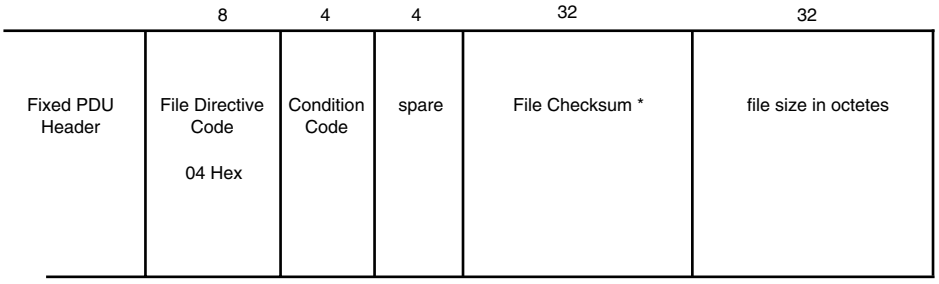
NAK/Keep Alive Code
'0' - NAK
'1' - Keep Alive

2.4.2 KEEP ALIVE PDU



2.5 TERMINATION PDUs

2.5.1 END OF FILE (EOF) PDU



* Modulo 232 octet-wide addition of all file segment data, aligned with reference to the start of file.
For method, see 4.1.5.1.1.8.
If the condition code is not zero then the checksum value shall be all zeroes

NOTES

1 File Checksum: Modulo 2^{32} word-wide addition (where 'word' is defined as 4 octets) of all file segment data transmitted by the sender (regardless of the condition code, i.e., even if the condition code is other than 'No error'), aligned with reference to the start of file.

2 File Size: Expressed in octets. This value shall be the total number of file data octets transmitted by the sender, regardless of the condition code (i.e., it shall be supplied even if the condition code is other than 'No error').

3 Unacknowledged-mode transactions always terminate on receipt of the EOF (No error) PDU; therefore, any Metadata or file data PDU received after the EOF (No error) PDU for the same transaction may be ignored.

2.5.2 FINISHED PDU

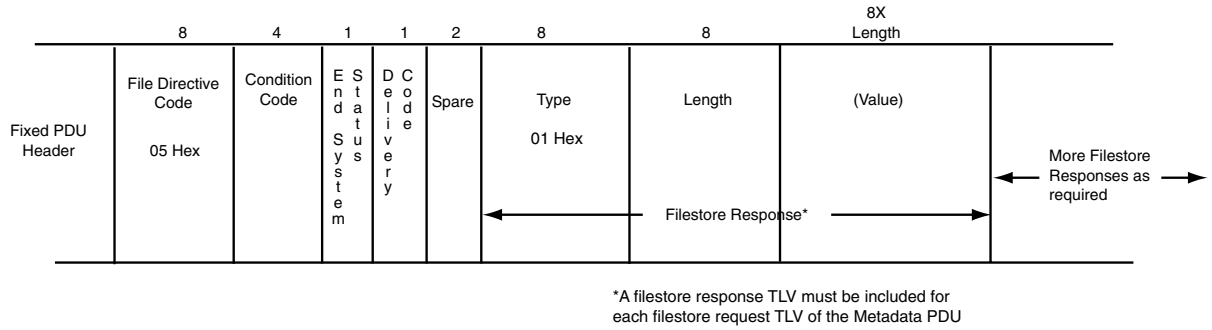
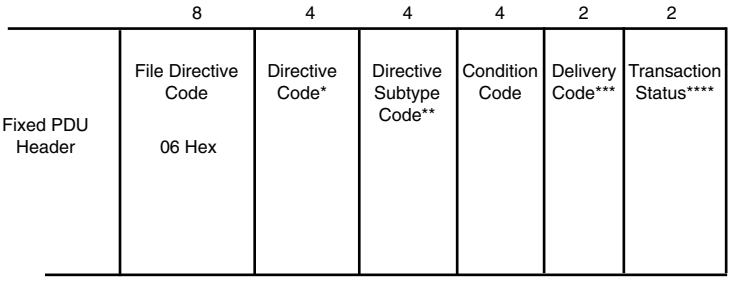


Table 2-9: Finished PDU Field Codes

Parameter	Values
Delivery Code	'0' - Data Complete '1' - Data Incomplete

2.5.3 POSITIVE ACKNOWLEDGMENT (ACK) PDU



* Directive code of the acknowledged PDU

** Values depend on file directive code. For Finished PDU: binary 0000 if generated by waypoint, binary 0001 if generated by end system. Binary 0000 for all other file directive codes.

***For ACK of EOF PDU: binary 10 if Data Complete, binary 11 if Data Incomplete. Binary 00 for ACKs of all other file directives.

****Status shall be undefined if the transaction is not active and transactions are not tracked after termination.

NOTE

Transaction Status parameter:

00 – Undefined: The transaction to which the acknowledged PDU belongs is not currently active at this entity, and the CFDP implementation **does not** retain transaction history. The transaction might be one that was formerly active and has been terminated, or it might be one that has never been active at this entity.

01 – Active: The transaction to which the acknowledged PDU belongs is currently active at this entity.

10 – Terminated: The transaction to which the acknowledged PDU belongs is not currently active at this entity; the CFDP implementation **does** retain transaction history, and the transaction is thereby known to be one that was formerly active and has been terminated.

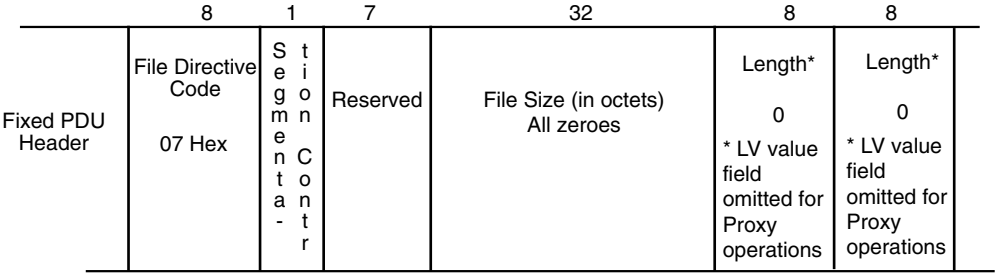
11 – Unrecognized: The transaction to which the acknowledged PDU belongs is not currently active at this entity; the CFDP implementation **does** retain transaction history, and the transaction is thereby known to be one that has never been active at this entity.

3 USER OPERATIONS MESSAGE FORMATS

3.1 USER OPERATIONS

3.1.1 METADATA PDU

User Operations Messages are contained in a metadata PDU, as pictured below:



3.1.2 RESERVED CFDP MESSAGE

Each individual User Operations Message in the metadata PDU is preceded by the Reserved Message Header field, pictured below. User Operations Message types are contained in table 3-1.

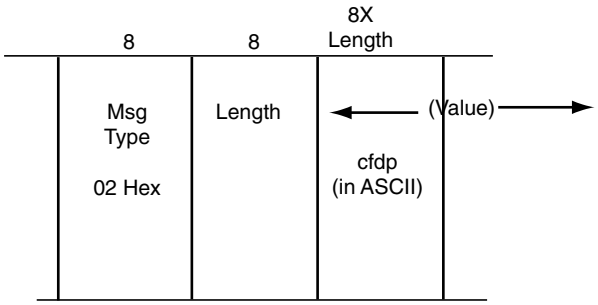
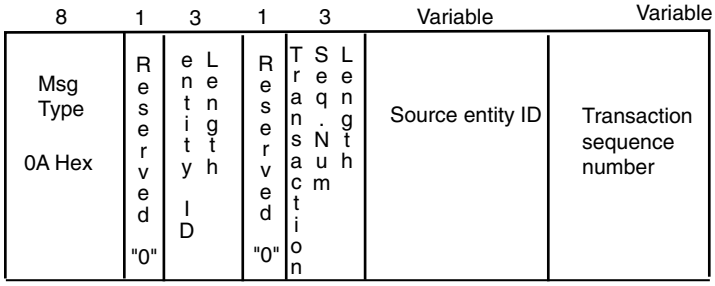


Table 3-1: User Operations Message Types

Msg Type (hex)	Interpretation
00	Remote Put Order
01	Remote Message to User
02	Remote Filestore Request
03	Remote Fault Handler Override
04	Remote Transmission Mode
05	Remote Flow Label
06	Remote Segmentation Control
07	Remote Put Finished
08	Remote Filestore Response
09	Remote Put Cancel
10	Directory Listing Request
11	Directory Listing Response
20	Remote Status Report Request
21	Remote Status Report Response
30	Remote Suspend Request
31	Remote Suspend Response
38	Remote Resume Request
39	Remote Resume Response

3.1.3 ORIGINATING TRANSACTION ID MESSAGE

The Originating Transaction ID message is common to all categories of User Operations messages, and its format, below, is the same when used in any of the categories.



3.2 PROXY OPERATIONS

3.2.1 REMOTE PUT ORDER

8	8	8X Length	8	8X Length	8	8X Length
Msg Type 00 Hex	Length	(Value) Destination entity ID	Length*	(Value) Source file name	Length*	(Value) Destination file name

* Length is zero if parameter is omitted

3.2.2 REMOTE MESSAGE TO USER

8	8	8X Length
Msg Type 01 Hex	Length	(Value)

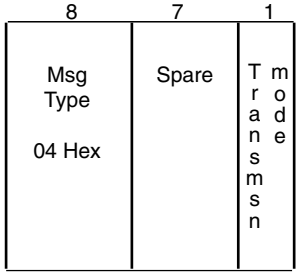
3.2.3 REMOTE FILESTORE REQUEST

8	8	8X Length
Msg Type 02 Hex	Length	(Value) (A single CFDP File Store Request)

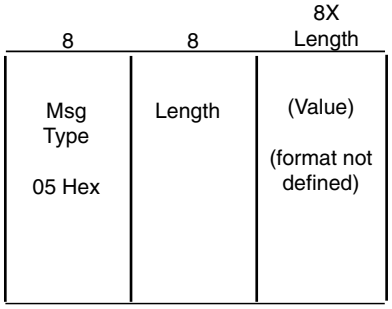
3.2.4 REMOTE FAULT HANDLER OVERRIDE

8	8	8X Length
Msg Type 08 Hex	Length	(Value) (A single CFDP File Store Ressponse)

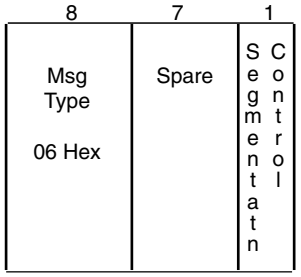
3.2.5 REMOTE TRANSMISSION MODE



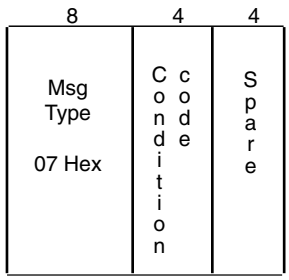
3.2.6 REMOTE FLOW LABEL



3.2.7 REMOTE SEGMENTATION CONTROL



3.2.8 REMOTE PUT FINISHED



3.2.9 REMOTE FILESTORE RESPONSE

8	8	8X Length
Msg Type 08 Hex	Length	(Value) (A single CFDP File Store Ressponse)

3.2.10 REMOTE PUT CANCEL

8
Msg Type 09 Hex

3.3 DIRECTORY OPERATIONS

3.3.1 DIRECTORY LISTING REQUEST

8	8	8X Length	8	8X Length
Msg Type 10 Hex	Length	(Value) Directory Name	Length	(Value) Directory File Name*

* The file name and path at the filestore local to the requesting CFDP user in which the responding CFDP user should put the directory listing

3.3.2 DIRECTORY LISTING RESPONSE

8	8	8	8X Length	8	8X Length
Msg Type 11 Hex	Listing Response Code 00-7F-Successful 80-FF-Unsuccess-ful	Length	(Value) Directory Name*	Length	(Value) Directory File Name**

*The name of the directory being listed, taken from the directory listing request

**The file name and path at the filestore local to the requesting CFDP in which the listing has been put, taken from the directory listing request

3.4 REMOTE STATUS REPORT OPERATIONS

3.4.1 REMOTE STATUS REPORT REQUEST

8	1	3	1	3	variable	variable	8	8X Length
Msg Type 20 Hex	Reserved	entity ID length	Reserved	Transact Seq Num length	Source entity ID	Transaction Sequence Number	Length	(Value) Report File Name

3.4.2 REMOTE STATUS REPORT RESPONSE

8	2	6	1	3	1	3	variable	variable	8	8X Length
Msg Type 21 Hex	Transaction Status	Reserved	Reserved	entity ID length	Reserved	Transact Seq Num length	Source entity ID	Transaction Sequence Number	Length	(Value) Report File Name

3.5 REMOTE SUSPEND OPERATIONS

3.5.1 REMOTE SUSPEND REQUEST

8	1	3	1	3	variable	variable
Msg Type 30 Hex	Reserved	entity ID length	Reserved	Transact Seq Num length	Source entity ID	Transaction Sequence Number

3.5.2 REMOTE SUSPEND RESPONSE

8	1	2	5	1	3	1	3	variable	variable
Msg Type 31 Hex	Susp Ind	Transaction Status	Reserved	Reserved	entity ID length	Reserved	Transact Seq Num length	Source entity ID	Transaction Sequence Number

3.6 REMOTE RESUME OPERATIONS

3.6.1 REMOTE RESUME REQUEST

8	1	3	1	3	variable	variable
Msg Type 38 Hex	Reserved	entity ID length	Reserved	Transact Seq Num length	Source entity ID	Transaction Sequence Number

3.6.2 REMOTE RESUME RESPONSE

8	1	2	5	1	3	1	3	variable	variable
Msg Type 39 Hex	Susp Ind	Transaction Status	Reserved	Reserved	entity ID length	Reserved	Transact Seq Num length	Source entity ID	Transaction Sequence Number

4 PROTOCOL OPTIONS, TIMERS, AND COUNTERS

4.1 GENERAL

This section contains subsections for implementation options, timers, and counters.

4.2 OPTIONS

Table 4-1: Options

Put Modes	Effect
UnACK	Selects Unreliable mode of operation.
NAK	Selects Reliable mode of operation.

Put NAK Modes	Effect
Immediate	NAKs are sent as soon as missing data is detected.
Deferred	NAK is sent when EOF is received.
Prompted	NAK is sent when a Prompt (NAK) is received.
Asynchronous	NAK is sent upon a local (implementation-specific) trigger at the receiving entity.

Put PDU CRC	Effect
True	Requires that a CRC be calculated and inserted into each File Data PDU.
False	No CRC is inserted in File Data PDUs.

Put File Types	Effect
Bounded	Sends a normal file, i.e., one in which the file is completely known before transmission.
Unbounded	Sends a file the length of which is not known when transmission is initiated (intended primarily for real-time data).

Segmentation Control (Record Boundaries Respected)	Effect
Yes	Causes each File Data PDU to begin at a record boundary.
No	Ignores record structure when building PDUs.

Put Primitives (Receiving End)	Effect
EOF-sent.ind	Indicates to User at source entity that the EOF for the identified transaction was sent.
Transaction-finished.ind	Mandatory at source entity, optional at destination entity.
File-segment-receive.ind	Indicates to the user at destination entity that a File Data PDU has been received.

Action on Detection of a Fault	Effect
Cancel	Cancels subject transaction.
Suspend	Suspends subject transaction.
Ignore	Ignores error (but sends Fault.indication to local user).
Abandon	Abandons transaction with no further action.

Action on Cancel At Receiving End	Effect
Discard data	Discards all data received in the transaction.
Forward incomplete	Forwards all data received to the local destination.

Put Report Modes (Sending End)	Effect
Prompted Rpt	Returns report on Prompt from local user.
Periodic	Returns report to local user at specified intervals.

Release of Retransmission Buffers	Effect
Incremental and Immediate	Releases local retransmission buffer as soon as sent.
In total When 'Finished' Received	Releases local retransmission buffer only when Finished PDU is received.

4.3 TIMERS

The following should be considered relative to the use of timers:

a) At the sender, the timer for a given EOF or Finished PDU should not be started until the moment that the PDU is delivered to the link layer for transmission. All outbound queuing delay for the PDU has already been incurred at that point.

b) At the receiver, acknowledgment PDUs should always be inserted at the *front* of the priority First-In-First-Out (FIFO) list to ensure that they are transmitted as soon as possible after reception of the PDUs to which they respond. (Acknowledgment PDUs are small and are sent infrequently, so the effect on the delivery of any emergency traffic is insignificant.)

c) To account for any additional delays introduced by loss of connectivity, the implementer must rely on external link state cues. Whenever loss of connectivity is signaled by a link state queue, the timers for all PDUs destined for the corresponding remote entity should be suspended; reacquiring the link to the entity should cause those timers to be resumed. By using this method, there is no need to try to estimate connectivity loss delays in advance, and there is no need for CFDP itself to be aware

of either the ephemerides or the tracking schedules of the local entity or of any remote entity.

Table 4-2: Timers

TIMER NAME	TYPE	TIMER LOCATION	STARTS ON	RESETS ON	TERMINATES ON	ACTIONS ON EXPIRY
NAK Retry Timer	Mandatory for all acknowledged modes	FDU Receiving entity	Issuance of a NAK	Issuance of a NAK	Reception of all requested data	Issue a new NAK for all unreceived data
ACK Retry Timer	Mandatory for all acknowledged modes	Entity issuing PDU to be acknowledged	Issuance of a PDU requiring positive acknowledgment	Re- issuance of the PDU	Reception of expected response	Re-issue the original PDU
Prompt (NAK) Timer	Implementation option	FDU Sending entity	Implementation-specific	Implementation-specific	Implementation-specific	Issue a Prompt (NAK) PDU
Async NAK Timer	Implementation option	FDU Receiving entity	Implementation-specific	Implementation-specific	Implementation-specific	Issue a single NAK for all unreceived data
Keep Alive Timer	Optional in all acknowledged modes	FDU Receiving entity	Implementation-specific	Implementation-specific	Implementation-specific	Issue a Keep Alive PDU
Prompt (Keep Alive) Timer	Optional in all acknowledged modes	FDU Sending entity	Implementation-specific	Implementation-specific	Implementation-specific	Issue a Prompt (Keep Alive) PDU
Inactivity Timer	Mandatory except sending entity in unacknowledged mode	Each Source and Destination entity	Reception of any PDU	Reception of any PDU	Implementation-specific	Issue an Inactivity.indication

4.4 COUNTERS

Table 4-3: Counters

COUNTER NAME	TYPE	COUNTER LOCATION	COUNTER LIMIT	ACTION ON REACHING LIMIT
NAK Timer Expiration Limit	Mandatory for all acknowledged modes	FDU Receiving entity	Implementation-specific	Invoke Fault procedures
ACK Timer Expiration Limit	Mandatory for all acknowledged modes	Entity issuing PDU to be acknowledged	Implementation-specific	Invoke Fault procedures
Keep Alive Discrepancy Limit	Optional		Implementation-specific	Invoke Fault procedures

.

5 STATE TABLES NOTES

5.1 GENERAL

This section contains adjunct material to the State Transition Diagrams and State Tables contained in reference [1].

5.2 CFDP KERNEL

The State Tables contained in reference [1] assume that some other part of the CFDP implementation:

a) starts up all required State Machines;

b) establishes the lower communications layer connection(s);

c) receives all incoming PDUs via the lower communications layer and delivers them to the appropriate State Machine(s) by triggering the appropriate event(s);

d) receives all User Requests and delivers them to the appropriate State Machine(s) by triggering the appropriate event(s).

The standard does not specify how these needs are met. One possible solution is to implement a CFDP Kernel. The following logic is a summary of one possible kernel implementation. The logic shows the action to be taken by the kernel in response to all possible events:

/* A new Sender State Machine is created for each User Put Request */

E11 - Received a Put Request

If ("the Put Request is for Unacknowledged Mode)

Establish lower-communications-layer path to/towards the destination.

Start a new Unacknowledged Mode Sender State Machine and trigger E11.

Else /* Acknowledged Mode */

Establish lower-communications-layer path to/towards the destination.

Start a new Acknowledged Mode Sender State Machine and trigger E11.

/* A new Receiver State Machine is created for each MD/FD/EOF PDU containing a new transaction-id. MD/FD/EOF PDUs are delivered to the appropriate State Machine. */

```
E12 - Received a Metadata PDU

E13 - Received a File-data PDU

E14 - Received an EOF PDU

  If ("this is a PDU from the Sender to the Receiver")

    If ("no State Machine is assigned to the referenced transaction")

      /* Start state machine(s) to handle this new transaction */

      {

If ("the PDU Transmission Mode is 'Unacknowledged'")

          Start a new Unacknowledged Mode Receiver State Machine.

        Else /* Acknowledged Mode */

          Start a new Acknowledged Mode Receiver State Machine.

      }

    /* Any required state machine(s) are running; trigger the appropriate event(s) */

    Trigger the same event for the Receiver State Machine (E12, E13, or E14)

  Else /* PDU from Receiver to Sender */

    /* The protocol never requires sending MD, FD, or EOF back to the Sender */

    Throw it away.

E15 Received an Ack-EOF PDU

E17 Received an Ack-Finished PDU

E18 Received a NAK PDU

  If ("this is a PDU from the Sender to the Receiver")

    If ("a Receiver State Machine is assigned to the referenced transaction")

      Trigger the same event in that Receiver State Machine (E15, E17, or E18).

  Else /* from Receiver to Sender */

    If ("a Sender State Machine is assigned to the referenced transaction")
```

Trigger the same event in that Sender State Machine (E15, E17, or E18).

E16 Received a Finished-final PDU

E26 Received an EOF-Cancel PDU

E28 Received a Finished-Cancel PDU

If ("this is a PDU from the Sender to the Receiver")

If ("a Receiver State Machine is assigned to the referenced transaction")

{

Trigger the same event in that Receiver State Machine.

}

Else /* from Receiver to Sender */

If ("a Sender State Machine is assigned to the referenced transaction")

{

Trigger the same event in that Sender State Machine (E16a, E22, E23, E25, or E28).

}

/* Most User Requests are delivered to all State Machines assigned to their transaction; this ensures that Cancel reaches all entities involved in the transaction. */

E29 Received a Cancel Request

E31 Received a Suspend Request

E34 Received a Resume Request

E36 Report Request

If ("a Sender State Machine is assigned to the referenced transaction")

Trigger the same event in that Sender State Machine (E29, E31, E34, or E36).

If ("a Receiver State Machine is assigned to the referenced transaction")

Trigger the same event in that Receiver State Machine (E29, E31, E34, or E36).

6 IMPLEMENTATION CONSIDERATIONS

6.1 GENERAL

The CFDP protocol was designed to provide file delivery services in a wide category of space missions which were derived from a series of representative, but generic, scenarios.

In the context of a specific mission, many considerations can impact on the way that CFDP services will be requested and solicited. For example:

a) Mission analysis;

b) System requirements (reliable/unreliable transfers, autonomy, transfer initiative management, …);

c) Spacecraft orbit and visibility (Low Earth Orbit [LEO], Geosynchronous Earth Orbit [GEO], Geosynchronous Transfer Orbit [GTO], Deep Space, …);

d) Onboard data handling capabilities;

e) Ground stations density (disjoint/overlapping passes, …);

f) Ground segment connectivity (bandwidth limitation, …);

g) Ground segment topology and interfaces (functional distribution, reusability of existing components, compatibility issues, …);

h) Operational requirements (pass management, ground station availability, …);

i) . . . .

Such considerations may lead to the selection of specific classes or subsets of the CFDP (e.g., reliable or unreliable modes of data transmission). In order that the protocol may successfully operate in any particular mission environment, it must be complimented by implementation-specific information and enabling mechanisms.

6.2 TRANSFERRING SUPPORTING INFORMATION

During the CFDP design phase, considerable effort was deployed to avoid an exponential expansion of the number of optional parameters carried by CFDP PDUs. To reduce complexity, CFDP is intentionally restricted to a minimum set of primitives sufficient to achieve its primary objective of transferring files.

In situations where it is necessary to convey CFDP-related information to a remote system, the information is propagated outside of the CFDP protocol.

Basically, three alternative ‘bypass’ solutions are suggested:

a) CFDP may be used to transfer a 'message to user' using a metadata PDU for an FDU that does not contain file data. The message will be passed to the CFDP user and from there it may be conveyed to a local application using implementation-specific mechanisms. This 'user to user' pass-through interface can be used to deliver a mission-specific directive or option. For example: 'suspend transaction number X in 6 minutes then auto resume this transaction in 7 hours and 35 minutes' is the kind of macro directive *not* supported by CFDP, but which can be carried by CFDP to an appropriate application via the CFDP 'message to user'.

b) CFDP may be used to transfer a file with an associated message to user. For example, 'here is a file containing pass schedules for next 10 days'.

c) CFDP is not the only way to communicate with the remote system, and any alternative interface (Telecommand [TC] or Telemetry [TM] packet) can be used to carry unsupported CFDP features. For example, 'this packet means that remote CFDP is momentarily off, due to an onboard reconfiguration'.

Bypass and proprietary solutions should only be used when basic CFDP services are not able to provide the required function.

6.3 EXAMPLE FILE CHECKSUM CALCULATION

6.3.1 SPECIFICATIONS

As specified in reference [1]:

The checksum shall be 32 bits in length and calculated by the following method:

1) it shall initially be set to all 'zeroes';

2) it shall be calculated by modulo 2^{32} addition of all 4-octet words, aligned from the start of the file;

3) each 4-octet word shall be constructed by copying into the first (high-order) octet of the word, some octet of file data whose offset within the file is an integral multiple of 4, and copying the next three octets of file data into the next three octets of the word;

4) the results of the addition shall be carried into each available octet of the checksum unless the addition overflows the checksum length, in which case carry shall be discarded.

6.3.2 EXAMPLE

An example of creating the checksum, developed by the National Space Development Agency (NASDA) of Japan, comprises the remainder of this subsection.

The checksum is calculated by modulo 2^{32} addition of 4-octet integers. The integers are constructed from 4-octet sets aligned from the start of the file. Each set is converted to an

integer by placing its first octet in the leftmost octet of the integer, and so on, up to the fourth octet which is placed in the rightmost octet. The integer is then added to the 4-octet running total, ignoring addition overflow.

Octets may omitted, either due to file segments arriving out of order or the file size being an inexact multiple of 4, i.e., 32 bits. Missing octets may be substituted with zeroes for the purposes of checksum calculation, as addition is commutative.

Worked example:

a. Consider a 10-byte file

0x8a	0x1b	0x37	0x44	0x78	0x91	0xab	0x03	0x46	0x12

b. The checksum calculation is:

```
      0x8a1b3744   Bytes 0-3

+     0x7891ab03   Bytes 4-7
      ----------
      0x102ace247

&     0xffffffff   Modulo 2^32, clear carry flag
      ----------
      0x02ace247

+     0x46120000   Bytes 8-9, padded with trailing zeroes
      ----------
      0x48bee247   Final checksum, carry flag not set
```

6.4 JPL NOTES ON CFDP IMPLEMENTATION

6.4.1 OVERVIEW

(Contributed by Scott Burleigh, NASA/JPL)

The Jet Propulsion Laboratory's (JPL) implementation of CFDP has been aimed at reducing the need for active management of the protocol to the lowest level possible, in the expectation that maximizing protocol agent autonomy will help minimize the cost of operating complex deep space missions (the Mars program, for example). Here is a

discussion of several design approaches embodied in that implementation which other implementers might (or might not) find useful.

6.4.2 DEFERRED TRANSMISSION

Deferred transmission can offer a degree of convenience to applications: it simplifies applications by relieving them of the need to know when communication links are active.

Deferred transmission makes CFDP responsible for scheduling file delivery to various other CFDP entities. Two implementation measures support this:

a) First, the function of responding to application requests for file delivery is partitioned from the function of handing data to the link layer for transmission; the former is handled by the *fdpd* (FDP daemon) task, the latter by a separate *fdpo* (FDP output) task (fdpd is always running, but fdpo runs only while the communication link to a specific CFDP entity is active). In response to application requests, fdpd constructs CFDP PDUs and enqueues them in **persistent FIFOs** (linked lists) of data destined for the designated entities; separately, fdpo dequeues PDUs from those FIFOs and passes them on to the underlying communication system for immediate radiation. The FIFOs grow while links are inactive, and shrink while they are active, but this is transparent to applications.

b) Second, the implementation fully supports the 'link state change' procedures by starting and stopping fdpo. CFDP itself is just a communication protocol, not an operating system; in order for the host of the CFDP entity (spacecraft, ground station, whatever) to be able to use CFDP for communication, the host itself must establish the communication links that CFDP will use. Some mechanism—e.g., scheduled tracking passes, beacon response, or some combination of both—must therefore exist for commanding the host to establish and break those links. This implies that knowledge of link state already exists outside of CFDP, so delivery of that knowledge to CFDP can be used to drive the starting and stopping of fdpo tasks. In the case of a single entity that can communicate with multiple remote entities, those external *link state cues* also tell fdpo which entity is currently 'in view' and, therefore, from which FIFOs to dequeue PDUs .

By relying on link state cues to control the operation of fdpo, we can accommodate occultation and other interruptions in connectivity simply and efficiently: when the link is lost, CFDP simply stops transmission and reception of data between the two endpoints of the link. This implementation of deferred transmission incurs far less overhead than using the Remote Suspend and Resume user operations to control suspension and resumption of communication:

a) Suspend and Resume operations entail protocol activity, requiring a cooperative interchange of data between entities. Deferred transmission is entirely local; no PDUs are issued or received to affect it.

b) Because deferred transmission is an entirely local mechanism, it is unaffected by delay due to the distance between the participating entities. Moreover, there is no chance of incomplete remote suspension/resumption due to loss of a PDU.

c) Remote Suspend and Resume are transaction-specific. This means that suspending all transmission between any pair of entities would require the reliable transmission of PDUs for every transaction currently in progress between them, as would resumption of transmission. In contrast, the deferred transmission mechanism is atomic and comprehensive.

6.4.3 FLOW LABELS

Flow label processing is identified in reference [1], but is left undefined. The JPL implementation of CFDP incorporates a flow label algorithm that is intended to provide highly flexible bandwidth allocation without requiring active management.

A JPL flow label is an integer in the range 0 through N inclusive, where N is some small value. In testing to date we have used N = 3, with 0 as the default flow label, for transactions that omit the flow label TLV from transaction metadata.

For each remote CFDP entity, fdpd enqueues the PDUs of each file destined for that entity onto one of N+1 FIFOs, depending on the flow label associated with the transaction. FIFO 'N' is designated the 'priority' queue for that entity Each of the other queues is assigned a 'service level', a number that indicates that queue's allocation of total transmission bandwidth in the absence of priority traffic.

Fdpo loops endlessly through the following algorithm to obtain from these N+1 FIFOs the PDUs it sends to the remote entity that is currently in view:

a) If there are any PDUs currently in the priority FIFO, remove the first PDU from that FIFO and transmit it.

b) Otherwise, if any of the non-priority FIFOs are non-empty:

    1) Compute 'service provided' for each non-empty non-priority FIFO. For a given FIFO, service provided is calculated as the FIFO's service total (the total number of bytes of data dequeued so far from this FIFO) divided by the service level assigned to the FIFO.

    2) Remove the first PDU from the FIFO for which the least service has been provided, transmit it, and add its length to that FIFO's service total.

c) Otherwise, wait until fdpd signals that PDUs have been placed in one or more of the FIFOs.

The service levels assigned to non-priority FIFOs can be any numeric values, but the service level assignment scheme we have used in testing enables a small optimization. If the service level assigned to FIFO n (where 0 <= n < N) is 2**n, then you can compute service provided

for any FIFO by simply shifting its service total n bits to the right. If N = 3, the FIFOs are configured as follows:

FIFO number	Service level
0	2**0 = 1
1	2**1 = 2
2	2**2 = 4
3	(priority FIFO, service level n/a)

Assigning a given file the flow label N causes it to be appended to the priority FIFO, so that it is transmitted after all previously enqueued priority transmissions (if any), but before all non-priority transmissions. Assigning a given file a flow label less than N causes it to be appended to the corresponding FIFO; it will be transmitted after all previously enqueued transmissions with the same flow label, but possibly before previously enqueued transmissions with different flow labels, depending on the lengths of the various FIFOs and the service levels assigned to them. For example, if all non-priority traffic is assigned either flow label 0 (with service level 1) or flow label 2 (with service level 4), and FIFOs 0 and 2 are both kept non-empty at all times, then transmissions assigned to flow 2 will be delivered four times as rapidly as those assigned to flow 0; flow 2 will occupy 80% of the transmission bandwidth, while flow 0 occupies the remaining 20%.

The effect of this scheme is to apportion transmission resources automatically to various classes of traffic, without ever starving any class of traffic altogether, while still enabling an emergency transmission to take temporary precedence over all other traffic when necessary. No management is necessary, aside from the assignment of service levels to flows.

NOTE – When an unused FIFO begins to be used, the algorithm described above may enable it to monopolize the transmission link for some time. (Its service total is initially zero, so its computed service provided may remain less than that of all other flows for a while, even if has a lower service level.) For this reason, an additional computation is performed each time a PDU is dequeued from a non-priority channel: if the difference between lowest and highest calculated values of service provided is greater than some constant K times the current data transmission rate (in bytes per second), then the service totals of all FIFOs are reset to zero to resynchronize the algorithm automatically. K, a management parameter, represents the maximum number of seconds the mission operator is willing to risk letting one flow monopolize the transmission link.

6.4.4 TIMERS

Successful transmission of a PDU can be signified by an acknowledgment, but the only reliable way to detect a possible failure in transmission is to wait for a timeout period to expire prior to acknowledgment. Computation of these timeout periods in CFDP is complicated by the fact that connectivity is discontinuous; reception of an acknowledgment may be arbitrarily delayed, not only by planetary occultation but also by resource scheduling decisions at both ends of the link. The effect of using an inaccurate timeout period to control retransmission can be either unnecessary delay in data delivery (if the timeout period is too long), or unnecessary retransmission traffic (if the timeout period is too short).

The JPL implementation of CFDP uses the following mechanism to detect timeout expiration for EOF and Finished PDUs:

a) The total time consumed in a 'round trip' (transmission and reception of the original PDU, followed by transmission and reception of the acknowledgment) has the following components:

   1) Protocol processing time at sender and receiver.

   2) Inbound queuing: delay at the receiver while the original PDU is in a reception queue, and delay at the sender while the acknowledgment is in a reception queue.

   3) Outbound queuing: delay at the sender while the original PDU is in a FIFO waiting for transmission, and delay at the receiver while the acknowledgment is in a FIFO waiting for transmission.

   4) Round-trip light time: propagation delay at the speed of light, in both directions.

   5) Delay due to loss of connectivity.

b) Processing time at each end is assumed to be negligible.

c) Inbound queuing delay is also assumed to be negligible, because processing speeds are high compared to data transmission rates, even on small spacecraft.

d) Two mechanisms are used to make outbound queuing delay negligible:

   1) At the sender, the timer for a given EOF or Finished PDU is not started until the moment that fdpo delivers the PDU to the link layer for transmission. All outbound queuing delay for the PDU has already been incurred at that point.

   2) At the receiver, acknowledgment PDUs are always inserted at the *front* of the priority FIFO to ensure that they are transmitted as soon as possible after reception of the PDUs to which they respond. (Acknowledgment PDUs are small and are sent infrequently, so the effect on the delivery of emergency traffic is insignificant.)

e) We assume that one-way light time to the nearest second can always be known (e.g., provided by the MIB). So the initial value for each timer is simply twice the one-way light time plus 1 second of margin to account for processing and queuing delays.

f) This leaves only one unknown, the additional round trip time introduced by loss of connectivity. To account for this, we again rely on external link state cues. Whenever loss of connectivity is signaled by a link state queue, we not only stop fdpo, but also suspend the timers for all PDUs destined for the corresponding remote entity; reacquiring link to the entity causes those timers to be resumed. There is no need to try to estimate connectivity loss delays in advance, nor is there is a need for CFDP itself to be aware of either the ephemerides or the tracking schedules of the local entity, or of any remote entity.

In testing performed to date, this mechanism seems to trigger timeout-driven retransmission without imposing either excessive retransmission traffic or excessive file delivery delay.

6.4.5 IGNORING LATE DATA

Unacknowledged-mode transactions always terminate on receipt of the EOF (No error) PDU. Therefore any Metadata or file data PDU received after the EOF (No error) PDU for the same transaction may be ignored.

6.4.6 PDU QUEUING WITHIN THE CFDP ENTITY

Under some circumstances, CFDP PDUs should be physically transmitted (radiated) in an order that differs from the order in which they were generated.

Operational considerations or other user constraints may require that access to transmission bandwidth be allocated among multiple ‘flows’ according to a user-visible management algorithm. Typically, it may be necessary to prevent the transmission of a single large but non-critical file from delaying the delivery of small but critical files whose transmission is requested later. The CFDP ‘flow label’ mechanism is intended to address this sort of requirement. The various ‘flows’ are typically implemented as logically distinct transmission channels within CFDP that must be multiplexed on output.

Additionally, though, some PDUs that serve only CFDP internal control purposes may need to be radiated on an urgent basis, possibly ahead of a large number of file data PDUs that are currently queued for transmission. A single CFDP service request or protocol procedure may result in the transmission of multiple PDUs. Since any single transmission medium can only send one value at a time, a CFDP implementation must provide some mechanism for imposing a rational order of transmission on those PDUs. Typically *queues* (or *FIFOs)* are the basis for this mechanism. However, PDU queuing must be done carefully in order to avert various kinds of trouble. In particular, if a single queue is used and new PDUs are always added to the back of this queue, then:

a) The File Data PDUs for an urgently needed file can never be transmitted until the previously queued PDUs of less important files, bound for the same destination entity, have been transmitted.

b) An ACK PDU will never be transmitted until all previously queued PDUs have been transmitted. This makes the arrival time of the ACK heavily dependent on the size of the backlog of PDUs pending transmission at the ACK's sending entity. Since this size is difficult or impossible to estimate accurately, the sender of the PDU to which the ACK is responding cannot accurately anticipate the ACK's arrival time; it therefore cannot know with any accuracy when to presume data delivery failure and retransmit the PDU.

One alternative approach is to use a single queue but manage it intensively, inserting new PDUs not just at the back but at various points throughout the queue, and possibly rearranging items within the queue as necessary.

Another approach, which seems structurally more complex but may be procedurally simpler, is to use multiple queues and merge them at the point of access to the Unitdata Transfer (UT) layer. A possible implementation is discussed in 6.4.3.

A further note on the effect of queuing on ACK arrival time: selection of accurate retransmission timer intervals in CFDP is difficult, but it need not be impossible. Nearly all of the uncertainty in computing these values can be removed if the CFDP implementation observes these principles (refer to 6.4.4 for a fuller discussion):

a) A positive acknowledgment timer should not be started until the affected PDU can be assumed to have been physically radiated. A service indication from the UT layer may be required for this purpose.

b) Positive acknowledgment timers should be temporarily stopped during any time interval in which the responding entity is unable to transmit (i.e., between tracking passes) and restarted when the responding entity's ability to transmit is restored (i.e., the next tracking pass starts). This activity is entailed in the 'link state change' procedures.

c) ACKs should be delivered to the UT layer immediately, as soon as they are created. This might mean inserting them at the front (rather than the back) of the single outbound PDU queue, or alternatively inserting them at the back of a separate, top-priority queue reserved for ACK transmission.

6.4.7 ADDITIONAL COMMUNICATIONS CHANNEL

The separate queue for ACK transmission alluded to in 6.4.6 might also be considered an 'additional communications channel', a mechanism for immediate transmission of urgent protocol control information.

It has been speculated that such a mechanism might be used for transmission of several types of file directive PDUs. ACK PDUs are clearly urgent enough to warrant top-priority transmission: significant delay in transmitting an ACK can result in premature timer

expiration and unnecessary retransmission, consuming scarce bandwidth. It is not yet clear that any other file directive PDUs are similarly critical, so no consensus on this topic has been reached within CCSDS Panel 1F.

6.4.8 TRANSACTION INDICATIONS

The Transaction.indication primitive that is issued to the user application upon initiation of a transaction indicates the ID assigned to the new transaction. However, CFDP is not constrained to block the submission of a Put.request primitive until a Transaction.indication has been issued in response to the prior Put.request; nothing in the standard prevents the submission of multiple Put.requests in quick succession without intervening reception of any resulting Transaction.indications. In order for the user application to be able to associate a transaction ID with the corresponding Put.request (and, implicitly, with the corresponding file), an implementation-specific mechanism must be supplied.

One option is flow control, the single-threading of Put.request activity described in 6.4.3. While the CFDP standard does not require this behavior, neither does it prohibit it.

Another option would be an implementation-specific transaction tag system, such as might be provided in an application programming interface. For example, the function used to submit a Put.request might return a ‘request ID’ number, which could subsequently be inserted into the data object that is sent to the user application when the resulting Transaction.indication is produced; the user application could link the Transaction.indication to the corresponding Put.request by request ID.

6.5 SIMPLE ANALYSIS OF NAK RETRANSMISSION

(Contributed by Rob Smith, Defence Evaluation and Research Agency [DERA]/British National Space Centre [BNSC].)

The performance for CFDP can be gauged by making a few simple approximations using the method outlined in this subsection. The most important measure is the probability of a PDU being received.

It has been assumed that the link has a long delay, whereby the data rate is high relative to the link delay, i.e., all data is transmitted and then, at some later time, all data is received. In this case, there is no time overlap between transmission and reception, which is not unreasonable, as data rates will increase and the speed of light will not.

The probability of PDU loss, q_{pl}, is dependent on the number of bytes in the PDU, n_p, and the probability of a bit error, p_{be}.

This confirms that risk of PDU loss increases with PDU length.

In a single transaction, most of the traffic consists of File Data PDUs, which are typically significantly larger than other PDUs involved. The majority of non-File Data PDUs are small, i.e., 20-200 bytes, and so are less prone to corruption. The only exception is the NAK PDU, which may be large if there is a lot of data corruption, and it is the trigger for File Data PDU retransmission.

Hence, the transaction simplifies to:

a) Send all File Data PDUs.

b) Return NAK PDU.

How big should File Data PDUs be? If they are too big, they are easy prey to bit errors, meaning the whole PDU must be resent. If they are too small, then their headers become an unacceptably large overhead. In this example, 1024 bytes has been taken as a reasonable compromise.

How big are NAK PDUs? Their size is dependent on the File Data PDU length, n_{fd}, probability of bit error, p_{be}, and the file size, n_{fl}.

The number of File Data PDUs, N_{fd}, is:

The probability that a File Data PDU is lost, q_{fd}, is:

So an estimate of the number of NAKs, n_n, is:

And the probability of a NAK PDU loss, q_n, is:

How likely is a bit error? This depends on the mission and its environment. For the purposes of this example, a typical link is assumed to lose 1 bit in 10^{10} ($p_{be}=10^{-10}$), and a poorly-designed link will drop 1 bit in 10^{6} ($p_{be}=10^{-6}$). These figures are based on current space communications links with and without error correction.

Consider a 1Gb file ($N_{fd} = 10^{6}$):

p_{be}	10^{-10}	10^{-8}	10^{-6}	10^{-5}
q_{fe}	$8x10^{-7}$	$8x10^{-5}$	$8x10^{-3}$	8x10-2
n_n	~1	80	8000	80000
q_n	$8x10^{-10}$	$6x10^{-6}$	$6x10^{-2}$ (1:16)	0.998 (~1)

Bit error probabilities of 10^{-8} and 10^{-5} have been added for context, as the trends are far from linear, especially around 10^{-6}. As the link quality decreases, the number of NAKs rises sharply, and the probability of NAKs failing becomes almost certain (~1).

The example presented here provides a rough guide to performance for a typical transaction. However, the analysis method has also been outlined to allow users to evaluate CFDP performance with their own mission parameters.

7 IMPLEMENTATION REPORTS

NOTE – The implementation reports in this section describe implementations created in the process of developing the CFDP specifications. The next issue of this Report will include reports on implementations conforming to the CFDP specifications contained in the next approved version of reference [1].

7.1 CNES CFDP IMPLEMENTATION REPORT

7.1.1 INTRODUCTION

Centre National d'Etudes Spatiales (CNES) has selected CFDP as a candidate protocol for future space missions. In support of this selection, CNES has performed detailed analysis of the CFDP, including data flow chart generation, the creation of state transition diagrams, and identification of timeout values. Further, utilizing the analysis and synthesis results, an implementation of the CFDP has been coded and the implementation has been thoroughly tested. In order to perform such testing CNES developed the space link simulator (LinkSim), and used the simulator in the testing and validation of the Core Procedures of the CFDP protocol.

7.1.2 CFDP VALIDATION PLAN

According to the CNES CFDP validation plan, the implementation validation will proceed through the following steps:

a) operational test and validation of the implemented protocol;

b) protocol test and validation in a simulated space mission;

c) validation by interoperation with the National Aeronautics and Space Administration (NASA), the European Space Agency (ESA) and, eventually, with other national space Agencies;

d) flight test and validation using a test spacecraft.

Currently, CNES has completed the first two steps, and the implementation of the CFDP has been proven to work as specified. In each set of tests, all user commands were executed.

The third step of validation, interagency operations, awaits implementation by other agencies in addition to NASA and ESA.

To accomplish the fourth step, CNES will implement CFDP on three different flight tests:

a) the Stratospheric Balloon system;

b) the Multi-mission Microsatellite Project, whose first launch is expected in mid-2001;

c) the Onboard Autonomy Evaluation Project, in which the CFDP is proposed as a means of space link connection automation.

These projects face more sophisticated space-ground connection geometry than in the past, and thus their need for a reliable file transfer or telemetry transfer protocol is increased.

Unfortunately, the first launch of the above projects (either a balloon or a spacecraft) is not expected before 2000. For the time being, the CFDP implementation will be enhanced and ported to the target processor.

7.1.3 IMPLEMENTATION

7.1.3.1 Implementation Objectives

The CNES implementation of the CFDP fulfills the following objectives:

a) verification of the new specification;

b) demonstration of the protocol within CNES as a candidate for its future space missions;

c) active participation of CNES in CCSDS to support and influence the emerging international standard.

7.1.3.2 Implementation Scope

The CNES implementation of CFDP has two phases:

a) implementation of the Core Procedures of the CFDP, plus some extended features of transparent waypoint (relay) functionality;

b) complementary addition of the Extended Procedures, which support the store-and-forward function.

The store-and-forward waypoint functionality has been excluded in the first implementation phase because there is no immediate need in CNES to have that functionality, and also because the direct link connectivity (Core CFDP) must be verified before going on to the Extended Procedures phase.

As the first implementation phase, the current implementation provides the following file transfer capabilities (see figure 7-1):

a) direct link (point-to-point) transactions;

b) one or more transparent hops (via ‘relays’ as opposed to CFDP ‘waypoints’);

c) proxy initiated transaction (both proxy put and proxy get);

d) one or more transparent hops between proxy initiator and file sender (or file receiver);

e) any combination of cases (a) through (d).

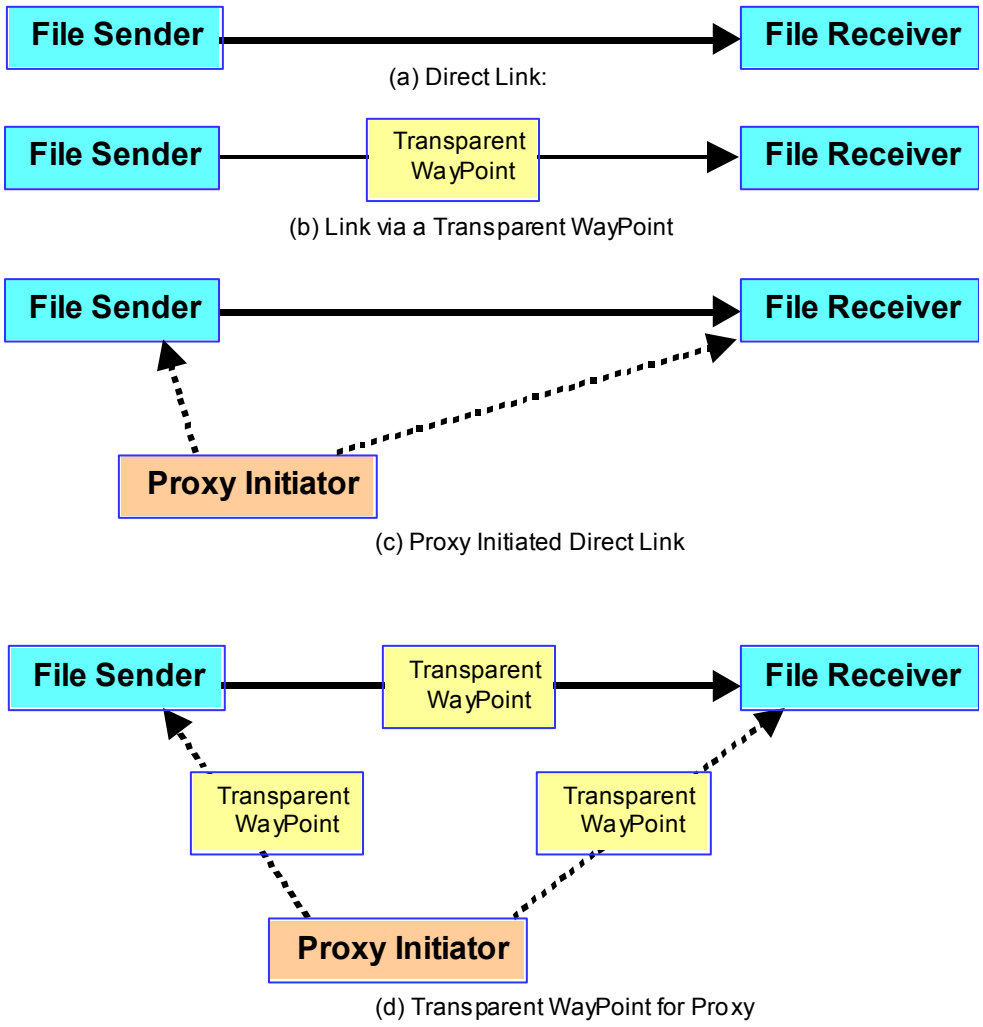


Figure 7-1: The Functionality of the CFDP Implementation

Although the current CNES CFDP implementation does not provide store-and-forward waypoint functionality, it can be simulated by adding an application program. The application program resides on the waypoint and manipulates the incoming file transfer data to forward the file to the final destination (file receiver) when the link to the file receiver is available. Figure 7-2 illustrates this application aided store-and-forward file transfer. In this case, two separate CFDP Core Procedures connections exist.

Figure 7-2: Store-and-Forward File Transfer by an Application Simulation

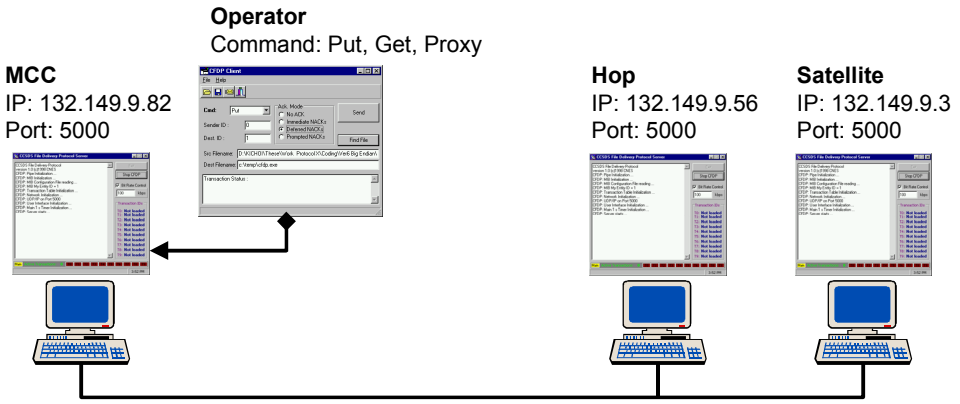
Figure 7-3 shows the test bed used for the CFDP protocol validation. Figure 7-3 (a) shows the configuration used for operational test and validation of the implemented protocol, and Figure 7-3 (b) shows the introduction of the space link simulator (LinkSim) between the two file transfer entities. Note that:

a) the ground mission control center and a spacecraft are simulated by two separate computers;

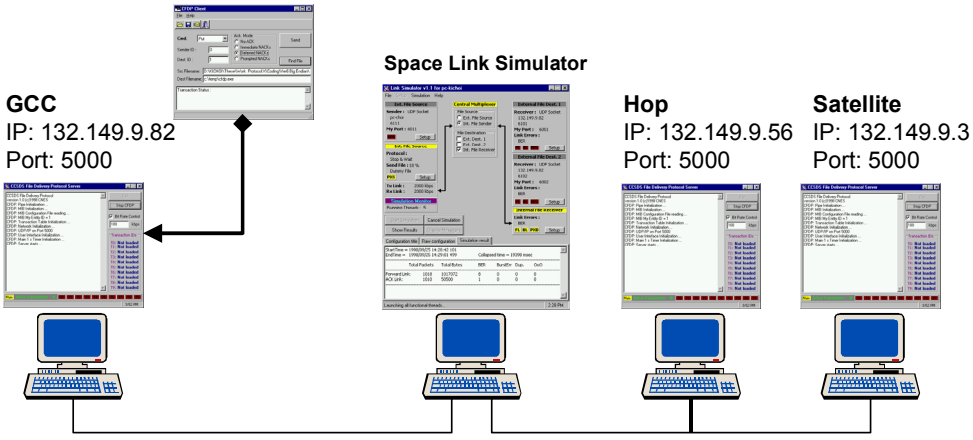
b) when required, the LinkSim is inserted into a separate computer between the mission control center and the spacecraft;

c) all computers are IBM PC compatibles running Windows NT as their operating system;

d) the programming language is C++.



(a) Direct link CFDP test and validation



(b) Simulated space link CFDP test and validation

Figure 7-3: CFDP Implementation Test Configurations

The LinkSim program introduces the following perturbations on the link:

a) packet losses according to a specified bit error rate (for example, BER = 10^{-7});

b) packet losses according to a specified burst of errors rate;

c) packets out of order according to a specified error rate;

d) link delay according to a specified link distance;

e) specified visibility (connectivity) time(s) for the link (file transfer is possible only during the visible period).

In this simulated space link environment, the CFDP has overcome all the introduced perturbations and anomalies and successfully completed the file transfers. In addition, when visibility (connectivity) is lost during a transaction, the CFDP implementation detects the event through its timers and waits until it again detects visibility (i.e., connectivity is re-established) and then resumes the suspended transaction.

The following operational scenarios were simulated on the test bed:

a) Mission Control Center (MCC) initiated spacecraft telemetry file downloading (CFDP Service Class 3, Scenario 1C);

b) spacecraft initiated onboard telemetry file downloading (CFDP Service Class 2, Scenario 1B);

c) MCC initiated telecommand file uploading (CFDP Service Class 4, Scenario 1D);

d) MCC initiated file transfer between two spacecraft or between one spacecraft and one ground terminal (CFDP Service Class 5, Scenario 2A).

This CFDP implementation has shown that the CFDP specification for the Core Procedures is adequate, and that the implementation operates correctly on a simulated space link.

7.1.3.3 Coding

The protocol has been implemented as a Windows NT application, named ‘CFDP server’. The CFDP server provides all the necessary functions implemented for the CFDP, and also provides the client interface via the Windows Pipe mechanism.

For user communication with the CFDP server, a ‘CFDP client’ was implemented, which issues the CFDP user's commands to the CFDP server.

As shown in figure 7-4, the connection between the CFDP server and client (user) is realized using Windows Pipe. The underlying network is User Datagram Protocol (UDP)/Internet Protocol (IP) of Windows NT.

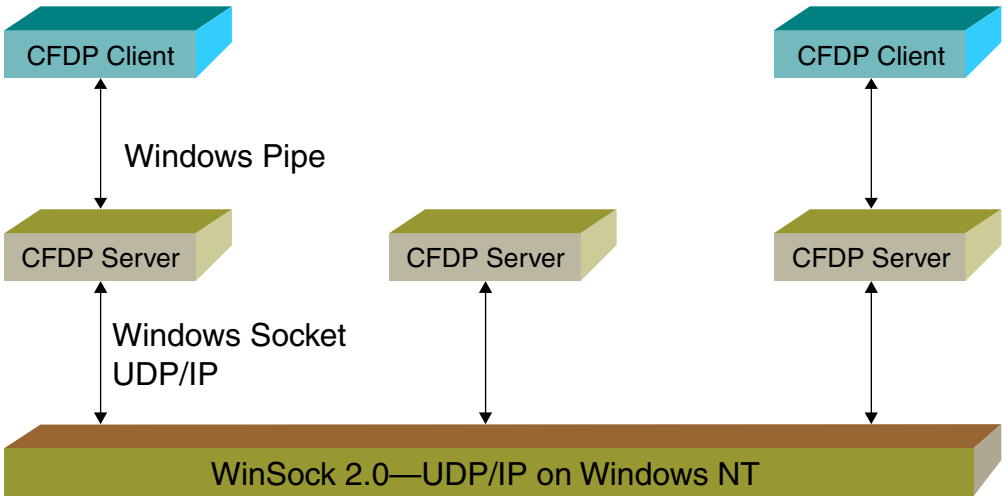
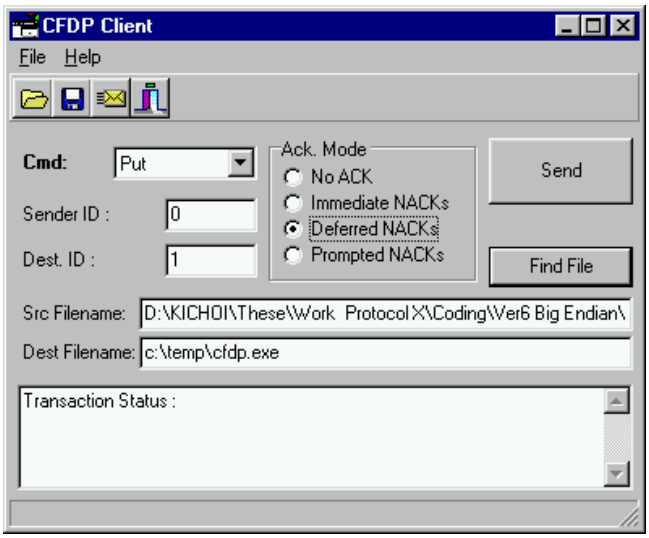


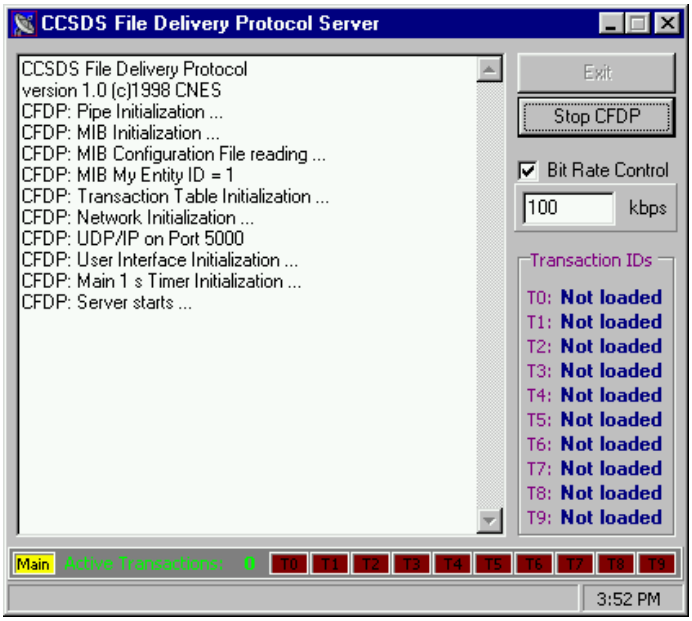
Figure 7-4: CFDP Program Organization

The CFDP server program incorporates a bit rate control mechanism in its UDP packet sending routine. This mechanism, when enabled, limits the maximum UDP packet flow on the network and allows determination of bandwidth efficiency through measurement of the total bandwidth and transaction time.

Figure 7-5 shows the CFDP server and CFDP client displays on Windows NT.



(a) CFDP Server



(b) CFDP Client

Figure 7-5: CFDP Server and Client Implementation on Windows NT

7.1.3.4 Timeout Mechanisms

In the CNES implementation of the CFDP, four different timeout values and one timeout counter were identified as necessary. These timers and counter are defined below. The relationship (if any) of each of these timers and counter to the timers and counters defined in reference [1] is noted in parenthesis at the end of each description.

a) RT_Timer: *Round Trip Timer*: Defined as a global variable, the timer can be changed for each transaction or, by an additional mechanism (such as TCP's RTT calculation), the value of RT_Timer can be changed dynamically. This timeout value is used to detect packet loss. *(NAK Retry Timer and Positive ACK Retry Timer.)*

b) Retry_Counter: *Retry Limit Exceed Counter*: Each time the RT_Timer expires, it increases the Retry_Counter and resends the previous PDU. The Retry_Counter has a certain predefined limit, and when it passes this limit, it alerts the CFDP protocol machine that a Protocol Error has occurred. *(NAK Retry Counter and ACK Retry Counter.)*

c) NoData_Timer: *No Data Timer*: This timeout value is used for two purposes, one at the file sender and one at the file receiver.

   1) File Receiver: Enabled in file receiving or data receiving cases. NoData_Timer is reset for each PDU received from the file sender. When NoData_Timer expires, the file receiver considers that the connection is lost. Normally the file receiver then puts the transaction into the suspended state. *(Implementation specific method.)*

   2) File Sender: When used in the file transmitting side, it can be enabled for the Prompted NAK mode, so that at each timeout of NoData_Timer, the sending entity issues a Prompt PDU. At the reception of the Prompt PDU, the receiving entity generates either a Keep_Alive PDU or, if a file gap has been detected, a NAK PDU. *(Prompt (NAK) Timer.)*

d) KeepAlive_Timer: *Keep Alive Timer*: Used to maintain the transaction activity at both the file sender and the file receiver.

   1) File Receiver: At the file receiver, on each timeout of the KeepAlive Timer a KeepAlive PDU is sent, or, if a packet loss has been detected, a NAK. *(KeepAlive Timer and/or an implementation of the Async NAK Timer.)*

   2) File Sender: In the suspended state, when triggered by this timer, the file sender sends a Metadata PDU. When the file receiver—in a suspended state—receives this Metadata PDU, it sends back a Resume PDU to wake up the file sender. *(Implementation-specific method.)*

e) Trx_Timer: *Transaction Timer*: Initiated at the start of a transaction, when it expires the transaction is terminated, whatever its current state. This avoids any blocking conditions which might persist indefinitely because of an undefined protocol error. *(Part of an implementation specific Protocol Error Procedure.)*

7.1.4 CFDP PERFORMANCE

7.1.4.1 Theoretical Evaluation of the CFDP Retransmission Strategies

The performance of the CFDP can be described by the throughput of the data transfer for a given constant bit rate communication link. Unfortunately the dynamic routing capability and the suspend/resume scheme make it impossible, or at least very difficult, to formulate the overall link throughput. In addition, no other protocols can be compared with CFDP in such an environment, as they do not support dynamic routing or suspend/resume capability. However, a useful throughput comparison can be made with other protocols by using the direct line-of-sight CFDP throughput (no intermediate waypoint).

The CFDP throughput can be evaluated for three different cases:

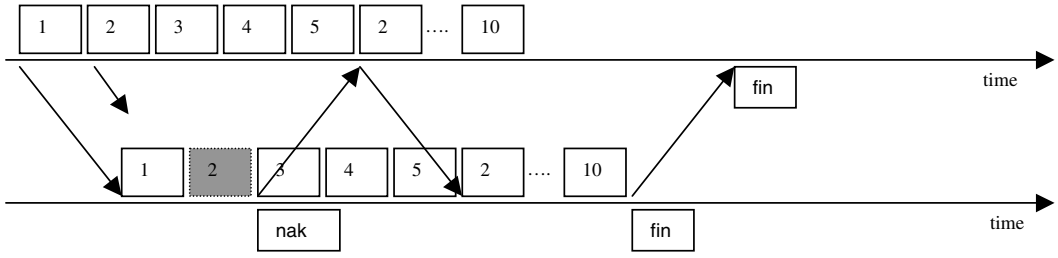
a) No Acknowledgements (simplex);

b) Immediate Negative Acknowledgments (NAK); and

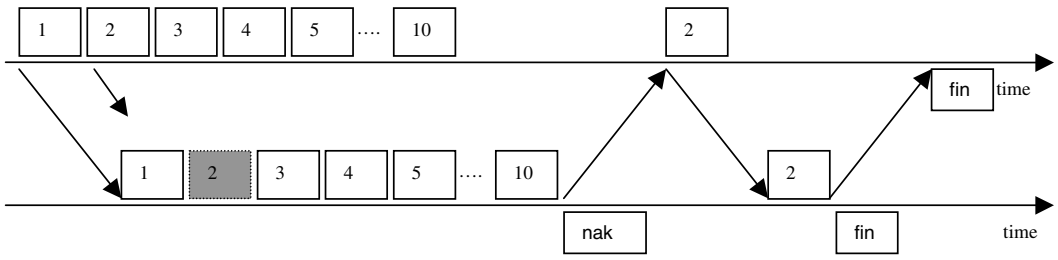
c) Deferred NAKs.

The two other cases are considered for evaluation of their theoretical throughput. Figure 7-6 shows an example schema for these two retransmission strategies.

In the No Acknowledgement (simplex link) case, the CFDP does not assure the integrity of the file at the other end, and the transfer time does not vary according to the different configurations and environmental conditions.



(a) CFDP **Immediate** Negative Acknowledgements



(b) CFDP **Deferred** Negative Acknowledgements

Figure 7-6: CFDP Example Data Flow Charts for Different NAK Strategies

For the Immediate NAK case, the total number of packets N_{total} which are transmitted by the file sender is given as

$$N_{total} = N + N \times PER + N \times PER^2 + N \times PER^3 + \ldots \quad (1)$$

$$= \mathrm{N} / (1 - \mathrm{PER})$$

where

PER : Packet Error Rate induced from the link Bit Error Rate (BER) = $1 - (1 - \mathrm{BER})^{8L}$

N : Total number of packets = $\frac{FS}{L - H}$

FS : File Size (in bytes)

H : Packet Header Length (in bytes)

L : Packet Length (in bytes)

N_{total} takes into account the repeating retransmissions of the packets.

To send this number of packets to the file receiver, the maximum value of the total transmission time $T_{Tx,max}$ of a file is calculated as

$$T_{Tx,max} = N_{total} \times T_{packet} + RTT = \frac{N \times T_{packet}}{(1 - PER)} + RTT \quad (2)$$

where

$T_{Tx,max}$: Maximum value of the total transmission time

T_{packet} : Packet time for a packet to arrive at the receiver = *L* / *LBR*

LBR : Link Bit Rate

RTT : Round Trip Time between file file sender and the file receiver (sec)

The term *RTT* is inserted in Equation *(2)*, because if the last packet of the file transfer is lost, *RTT* seconds will be taken for the receiver to send a NAK packet and receive back the corresponding packet, which is the last one of the whole transfer. In this equation, the processing time of both file sender and file receiver's CPU is ignored.

Consequently, the total transmission time of a file transfer T_{Tx} would be smaller than $T_{Tx,max}$ but greater than ($N_{total} \times T_{packet}$).

On the other hand, for the Deferred NAK case, the file receiver waits for the end of file transfer from the file sender before it issues the set of NAKs for all the lost packets during the transfer. In this case, the total transmission time T_{Tx} becomes

$$T_{Tx} = (N \times T_{packet}) + (RTT + N \times T_{packet} \times PER) + (RTT + N \times T_{packet} \times PER^2) + (RTT + \ldots \quad (3)$$

In Equation *(3)*, the first term in () represents the first time transfer of the given file data, and if there are errors, a round trip time (*RTT*) will be passed before starting to receive another set of the missing packets. Recursively calculated, the resulting equation becomes

$$T_{Tx} = \frac{N \text{ x } T_{packet}}{(1 - PER)} + k\,RTT \quad (4)$$

where

k : number of retransmissions, determined from N and PER

The number of retransmissions k varies according to N and PER. Repeating the transmissions for the erroneous packets will decrease the number of packets by PER, so k^{th} retransmission consists of $(N \times PER^k)$ packets. If this value $(N \times PER^m)$ is smaller than 1, then there will be no more packets to send. From N and PER, k is obtained.

$$k = -\frac{\log N}{\log PER} \quad (5)$$

For example, to send 10,000 packets with PER equal to 0.01, the retransmission time k is equal to 2. Of course, that is a theoretical estimation.

In summary, the total CFDP transmission time for the two different retransmission strategies can be described as

a) Immediate NAK case: $$T_{Tx} = \frac{N \text{ x } T_{packet}}{(1 - PER)} + RTT + T_o$$

b) Deferred NAK case: $$T_{Tx} = \frac{N \text{ x } T_{packet}}{(1 - PER)} - \frac{\log N}{\log PER} RTT + T_o \quad (6)$$

where T_o is the overall link overhead which consists of the initial connection establishment time and the final closing time. In the CFDP, a Put request PDU does not need to await the response from the file receiver, and the file sender starts immediately to send the file data. On the other hand, for the Get request PDU, which is issued by the file receiver, there is a wait of a single link time (half the *RTT*) before the file sender starts to send data packets. At the end of file transfer, both file sender and file receiver can issue the [Finished] PDU, and in the normal case, it is the file receiver which issues this [Finished] PDU. In this case, to close the connection ([Finished] PDU and then [ACK(Finished)] PDU), a round trip time (*RTT*) is needed for both ends.

The theoretical throughput of the CFDP is obtained in both cases:

Throughput of CFDP = File Size / T_{Tx}

7.1.4.2 Direct Link Performance Measurements

To measure the direct link throughput of the CFDP, different configuration parameters and link environments have been used, as described in table 7-1. Several sets of tests have been performed:

a) Test 1: Fixed File Size (1 MB), **Different Packet Size**, No Link Delay, No Link Errors;

b) Test 2: **Different File Size**, Fixed Packet Size (1024 bytes), No Link Delay, No Link Errors;

c) Test 3: Fixed File Size (1 MB), Fixed Packet Size (1024 bytes), **Different Link Delay**, No Link Errors;

d) Test 4: Fixed File Size (1 MB), Fixed Packet Size (1024 bytes), **Different Link Delay**, **Fixed BER** (1e-5), No Link Diversity, No Burst Errors;

e) Test 5: Fixed File Size (1 MB), Fixed Packet Size (1024 bytes), Fixed Link Delay (50 ms), **Different BERs**, No Link Diversity, No Burst Errors;

f) Test 6: Fixed File Size (1 MB), Fixed Packet Size (1024 bytes), Fixed Link Delay (50 ms), Fixed BERs (1e-5), **Link Diversity** (Packet Duplication = 1e-2, Packet Out-of-Order = 1e-2), No Burst Errors;

g) Test 7: Fixed File Size (1 MB), Fixed Packet Size (1024 bytes), Fixed Link Delay (50 ms), Fixed BERs (1e-5), Link Diversity (Packet Duplication = 1e-2, Packet Out-of-Order = 1e-2), **Burst Error** (BEOP = 0.001, Burst mean duration = 0.1 sec.).

Table 7-1: CFDP Performance Measurement Configuration Parameters

Variable Parameters	Unit	Parameter Values (default values in bold)					
Target File Size	Kbytes	10	100	**1000**	10000		
Transmission Packet Size	bytes	128	256	512	**1024**		
Link Delay	ms	**50**	100	200	400		
Bit Error Rate (BER)		0	1e-9	1e-8	1e-7	1e-6	**1e-5**
Packet duplication rate		0	**1e-2**				
Packet out-of-order rate		0	**1e-2**				
BEOP		0	**0.001**				
Burst Error Mean duration	sec	0	**0.1**				
Transmission bit rate	kbps	**100**					

Each of these tests also had 3 different modes of retransmission strategies: No Acknowledgements (Simplex), Immediate NAKs , and Deferred NAKs.

For these tests, the test files had the following sizes:

a) Kbytes File: 11,078 bytes;

b) Kbytes File : 100,618 bytes;

c) MB File: 1,001,078 bytes;

d) MB file: 9,986,678 bytes.

In all tests, each packet had 9 bytes of overhead for its header. Refer to tables 7-2 through 7-8 for summary information from the CFDP performance tests.

Table 7-2: CFDP Performance Test 1: Packet Length Variation

Test 1. Different Packet Size				
	Retransmission strategy	No ACKs	Imm. NAKs	Deferred NAKs
File	File size (bytes)	1,001,078	1,001,078	1,001,078
Constants				
CFDP	RT Timeout (sec)	0.1	0.1	0.1
parameters	Retry Timeout	10	10	10
	No Data Timeout (sec)	10	10	10
	Keep Alive Timeout (sec)	10	10	10
	Forward Link bit rate (kbps)	100	100	100
	Backward Link bit rate (kbps)	100	100	100
LinkSim	Link delay (ms)	0	0	0
parameters	Bit Error Rate	0	0	0
	Packet duplication rate	0	0	0
	Packet out-of-order rate	0	0	0
	Burst mean arrival time	0	0	0
	Burst mean duration	0	0	0
Variables	**Packet Length (bytes)**			
Resulting	128	**107.184**	**107.765**	**108.176**
Transfer	256	**97.941**	**97.991**	**97.951**
Time (sec)	512	**87.976**	**88.057**	**88.007**
	1024	**83.240**	**83.249**	**83.250**

Table 7-3: CFDP Performance Test 2: File Size Variation

Test 2. Different File Size to transfer				
	Retransmission strategy	No ACKs	Imm. NAKs	Deferred NAKs
Constants				
CFDP	Packet Length (bytes)	1024	1024	1024
parameters	RT Timeout (sec)	0.1	0.1	0.1
	Retry Timeout	10	10	10
	No Data Timeout (sec)	10	10	10
	Keep Alive Timeout (sec)	10	10	10
	Forward Link bit rate (kbps)	100	100	100
	Backward Link bit rate (kbps)	100	100	100
LinkSim	Link delay (ms)	0	0	0
parameters	Bit Error Rate	0	0	0
	Packet duplication rate	0	0	0
	Packet out-of-order rate	0	0	0
	Burst mean arrival time	0	0	0
	Burst mean duration	0	0	0
Variables	**File size to transfer (bytes)**			
Resulting	11,078	**0.941**	**0.931**	**0.941**
Transfer	100,618	**8.382**	**8.382**	**8.382**
Time (sec)	1,001,078	**83.240**	**83.240**	**83.189**
	9,986,678	**832.016**	**833.288**	**834.390**

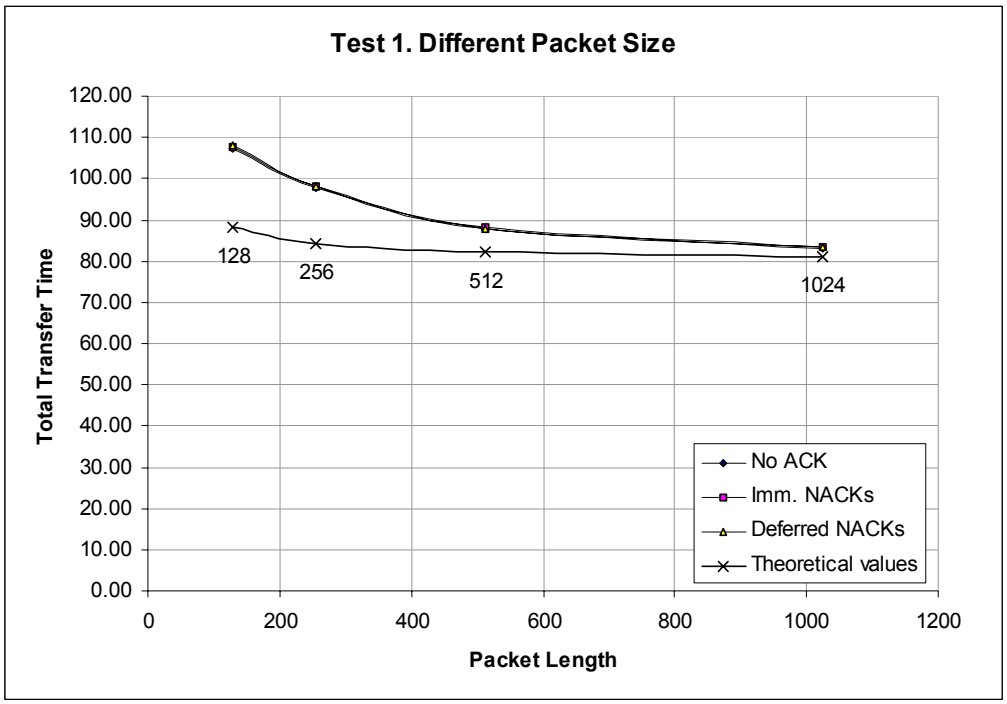


Figure 7-7: CFDP Performance Test 1: Packet Length Variation

Table 7-4: CFDP Performance Test 3: Link Delay Variation

Test 3. Different Link Delay				
	Retransmission strategy	No ACKs	Imm. NAKs	Deferred NAKs
File	File size (bytes)	1,001,078	1,001,078	1,001,078
Constants				
CFDP	Packet Length (bytes)	1024	1024	1024
parameters	RT Timeout (sec)	0.1	0.1	0.1
	Retry Timeout	10	10	10
	No Data Timeout (sec)	10	10	10
	Keep Alive Timeout (sec)	10	10	10
	Forward Link bit rate (kbps)	100	100	100
	Backward Link bit rate (kbps)	100	100	100
LinkSim	Bit Error Rate	0	0	0
parameters	Packet duplication rate	0	0	0
	Packet out-of-order rate	0	0	0
	Burst mean arrival time	0	0	0
	Burst mean duration	0	0	0
Variables	**Link Delay (ms)**			
Resulting	50	**83.230**	**83.240**	**83.329**
Transfer	100	**83.230**	**83.450**	**83.440**
Time (sec)	200	**83.239**	**83.480**	**83.440**
	400	**83.230**	**83.450**	**83.450**

Table 7-5: CFDP Performance Test 4: Link Delay Variation with Non-Zero BER

Test 4. Different Link Delay with non-zero BER				
	Retransmission strategy	No ACKs	Imm. NAKs	Deferred NAKs
File	File size (bytes)	1,001,078	1,001,078	1,001,078
Constants				
CFDP parameters	Packet Length (bytes)	1024	1024	1024
	RT Timeout (sec)	0.1	0.1	0.1
	Retry Timeout	10	10	10
	No Data Timeout (sec)	10	10	10
	Keep Alive Timeout (sec)	10	10	10
	Forward Link bit rate (kbps)	100	100	100
	Backward Link bit rate (kbps)	100	100	100
LinkSim parameters	Bit Error Rate	**1e-5**	**1e-5**	**1e-5**
	Packet duplication rate	0	0	0
	Packet out-of-order rate	0	0	0
	Burst mean arrival time	0	0	0
	Burst mean duration	0	0	0
Variables	**Link Delay (ms)**			
Resulting Transfer Time (sec)	50	**83.230**	**95.968**	**93.604**
	100	**83.230**	**96.178**	**99.403**
	200	**83.239**	**94.095**	**98.642**
	400	**83.230**	**95.637**	**96.729**

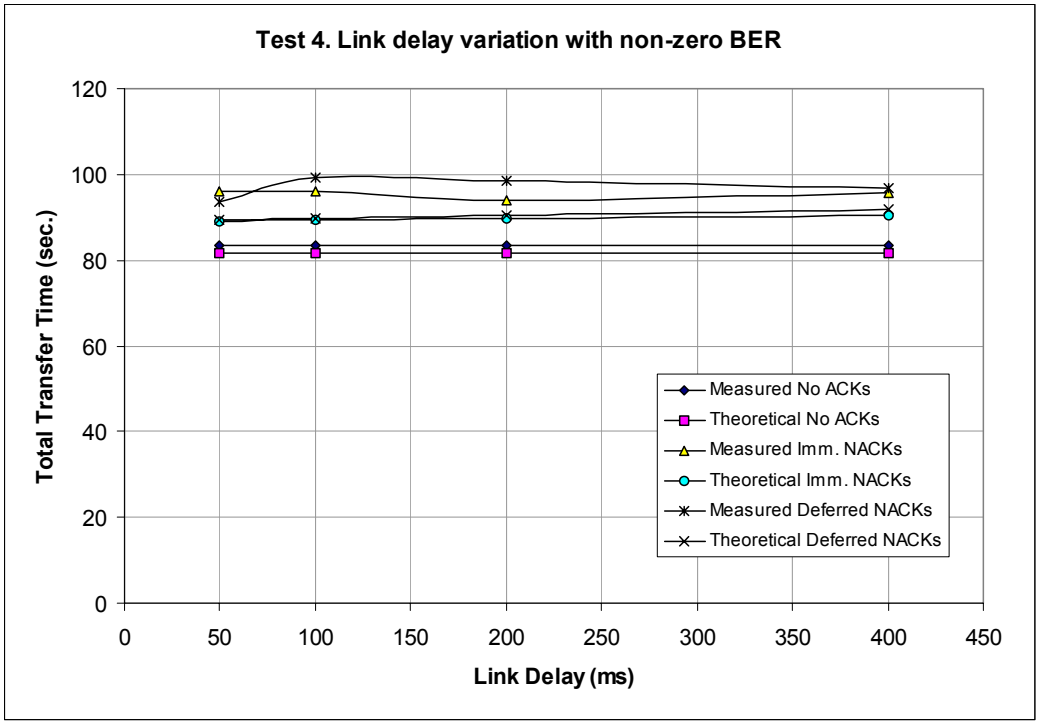


Figure 7-8: CFDP Performance Test 4: Link Delay Variation with Non-Zero BER

Table 7-6: CFDP Performance Test 5: BER Variation

Test 5. Different Bit Error Rates				
	Retransmission strategy	No ACKs	Imm. NAKs	Deferred NAKs
File	File size (bytes)	1,001,078	1,001,078	1,001,078
Constants				
CFDP	Packet Length (bytes)	1024	1024	1024
parameters	RT Timeout (sec)	0.1	0.1	0.1
	Retry Timeout	10	10	10
	No Data Timeout (sec)	10	10	10
	Keep Alive Timeout (sec)	10	10	10
	Forward Link bit rate (kbps)	100	100	100
	Backward Link bit rate (kbps)	100	100	100
LinkSim	Link Delay (ms)	50	50	50
parameters	Packet duplication rate	0	0	0
	Packet out-of-order rate	0	0	0
	Burst mean arrival time	0	0	0
	Burst mean duration	0	0	0
Variables	**Bit Error Rate**			
Resulting	1e-9		**83.350 / 0**	**83.330 / 0**
Transfer	1e-8		**83.329 / 0**	**83.701 / 2**
Time (sec) /	1e-7		**83.500 / 1**	**83.520 / 1**
Lost packets	1e-6		**84.532 / 5**	**85.503 / 7**
due to BER	1e-5		**98.151 / 107**	**93.745 / 74**

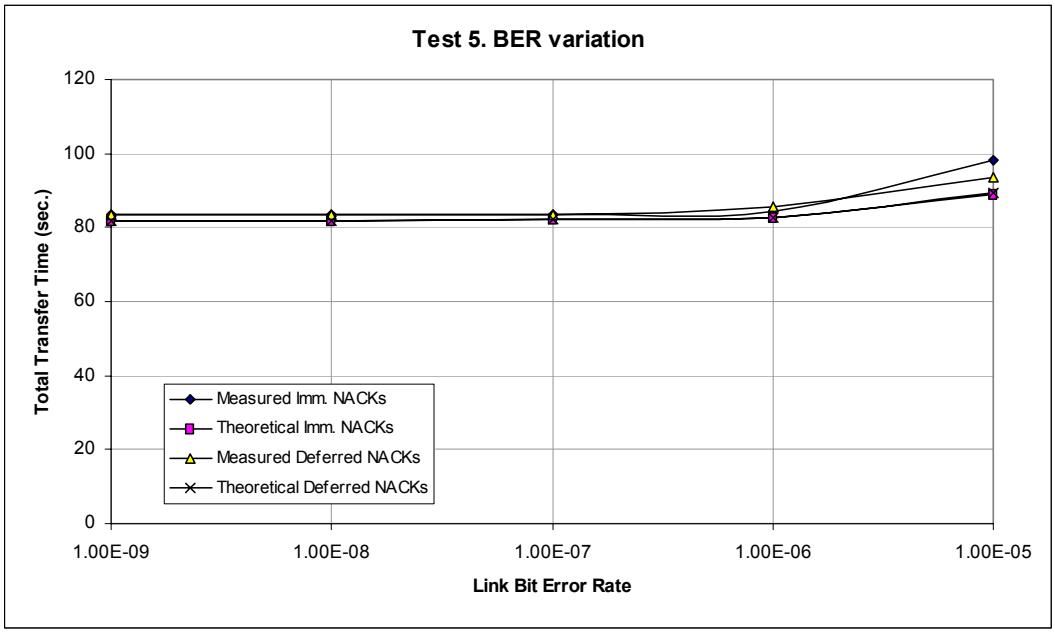


Figure 7-9: CFDP Performance Test 5: BER Variation

Table 7-7: CFDP Performance Test 6: Link Diversity Introduction

Test 6. Link Diversity Introduction				
	Retransmission strategy	No ACKs	Imm. NAKs	Deferred NAKs
File	File size (bytes)	1,001,078	1,001,078	1,001,078
Constants				
CFDP parameters	Packet Length (bytes)	1024	1024	1024
	RT Timeout (sec)	0.1	0.1	0.1
	Retry Timeout	10	10	10
	No Data Timeout (sec)	10	10	10
	Keep Alive Timeout (sec)	10	10	10
	Forward Link bit rate (kbps)	100	100	100
	Backward Link bit rate (kbps)	100	100	100
LinkSim parameters	Link Delay (ms)	50	50	50
	Bit Error Rate	1e-5	1e-5	1e-5
	Burst mean arrival time	0	0	0
	Burst mean duration	0	0	0
Variables	**Link Diversity**			
Resulting Transfer Time (sec)	Dup. = 1e-3 / OoO = 0	T_{Tx}	**96.099**	**90.560**
		Forward	**90 / 0 / 12 / 0**	**92 / 0 / 14 / 0**
		Backward	**1 / 0 / 0 / 0**	**0 / 0 / 0 / 0**
	Dup. = 0 / OoO = 1e-3	T_{Tx}	**95.287**	**93.774**
		Forward	**80 / 0 / 0 / 9**	**80 / 0 / 0 / 9**
		Backward	**0 / 0 / 0 / 2**	**0 / 0 / 0 / 1**
	Dup. = 1e-3 / OoO = 1e-3	T_{Tx}	**96.429**	**94.136**
		Forward	**84 / 0 / 10 / 10**	**86 / 0 / 13 / 12**
		Backward	**0 / 0 / 0 / 4**	**0 / 0 / 0 / 1**

Table 7-8: CFDP Performance Test 7: Burst Error Introduction

Test 7. Burst Error Introduction				
	Retransmission strategy	No ACKs	Imm. NAKs	Deferred NAKs
File	File size (bytes)	1,001,078	1,001,078	1,001,078
Constants				
CFDP	Packet Length (bytes)	1024	1024	1024
parameters	RT Timeout (sec)	0.1	0.1	0.1
	Retry Timeout	10	10	10
	No Data Timeout (sec)	10	10	10
	Keep Alive Timeout (sec)	10	10	10
	Forward Link bit rate (kbps)	100	100	100
	Backward Link bit rate (kbps)	100	100	100
LinkSim	Link Delay (ms)	50	50	50
parameters	Bit Error Rate	1e-5	1e-5	1e-5
	Packet duplication rate	1e-2	1e-2	1e-2
	Packet out-of-order rate	1e-2	1e-2	1e-2
Variables	**Burst Error**			
Resulting	BEOP = 0.001		**96.068**	**96.379**
Transfer	Mean Duration = 0.1 sec		**95 / 1/ / 6 / 11**	**121 / 1 / 7 / 8**
Time (sec)	Mean Arrival Time = 0.01 /sec		**0 / 1 / 4 / 3**	**0 / 1 / 2 / 3**

In Test 1 there are no link errors, and all packets are received correctly, so there is no difference between the acknowledged mode and the non-acknowledged mode except for the final confirmation phase.

Table 7-2 shows the measured results for Test 1, and figure 7-7 displays these results graphically together with the theoretical limit of the non-acknowledged mode. To provide higher credibility, five measurements are averaged to give each value.

As the packet length becomes smaller, the difference between the theoretical value and the measured results becomes larger. This is probably caused by the protocol processing overhead in the CFDP program, because using smaller packets increases the number of packets to be processed, and thus the number of file accesses and other mechanisms needed for the transport function also increases.

For the largest packet size (=1024 bytes for the data field), the theoretical value limits the total transmission time at 80.79 seconds. The measured values are more or less the same, around 83.35 seconds, so that the difference is 2.56 seconds. The throughput in theory can reach up to 99.13 kbps in the 100 kbps channel, equal to 99.13 % of the bandwidth usage. The measured throughput becomes 96.08 kbps (96.08 % bandwidth usage), with a 3 % overhead on the system.

The Test 2 results show that, without link errors, the total transfer time increases proportionally with the file size.

Tests 3 and 4 show that link delay without link errors has little effect on the throughput but, once link errors are introduced, the total transfer time increases significantly due to the number of retransmission packets. In Test 4, the 'No ACKs' transmission did not finish the file transfer due to the errors on the link (table 7-5 and figure 7-8). With 10^{-5} of BER, the Packet Error Rate (PER) becomes 0.08 (8 packets per 100 are lost due to the BER).

7.1.5 CONCLUSION

CFDP, by its design, would improve the connectivity of a space network. It permits the application layer to be automated and facilitates operations.

The CNES implementation of CFDP is not a final version, but a pioneering implementation. It has demonstrated the advantages and appealing features of this protocol.

7.2 ESA CFDP IMPLEMENTATION REPORT

7.2.1 INTRODUCTION

The implementation report presented in this subsection describes the validation status (within the European Space Research & Technology Centre [ESTEC]) of the CCSDS File Delivery Protocol (CFDP) through a software implementation and a document review of the CFDP Red Book (reference [1]) and its accompanying Green Books (references [3] and [4]).

This report provides information on:

a) CFDP implementation status;

b) Implementation environment and software architecture;

c) CFDP performance test results;

d) Specification document (reference [1]) validation report.

7.2.2 IMPLEMENTATION STATUS

The CFDP software coverage so far entails the implementation of the entire *Core* file delivery capability, in both Reliable and Unreliable service types, as well as the implementation of part of the Extended procedures. That requires the capability to perform a single *Point-to-Point* file copy operation between two CFDP entities, a *Proxy* file copy operation via one *waypoint* (Get Request), and a file transfer via *Store and Forward waypoints* between the file *sender* and the file *receiver*.

According to reference [1], the implemented classes are as follows:

a) Class 1: Unreliable Single *Point-to-Point* File Transfer;

b) Class 2 Reliable Single *Point-to-Point* File Transfer;

c) Class 3: Unreliable *End-toEnd* File Transfer via One Waypoint Entity;

d) Class 4: Reliable *End-toEnd* File Transfer via One Waypoint Entity;

e) Class 5: Reliable File Transfer by a Proxy Entity.

The Extended protocol classes (3 and 4) are basically functioning but not completely implemented and tested (i.e., Cancel, Suspend and Resume propagation procedures are missing).

7.2.3 CFDP IMPLEMENTATION ENVIRONMENT AND CODING

The test bed used for the CFDP implementation in ESTEC is an application developed under the *Delphi 4* Integrated Development Environment (IDE) using the *Object Pascal* programming language.

All computers used were International Business Machines (IBM) Personal Computer (PC) compatibles running Windows 95/NT OS.

The implementation can be subdivided in two different phases:

a) *Core* procedure development (currently implemented);

b) *Extended* procedures development (to be completed).

Two separate computers simulate the *ground statio*n and a *virtual satellite*.

From now on, the terms 'Client' and 'Server' will be used as they apply to the *Distributed Common Object Model* (DCOM) technology concept explained later in this subsection.

So far, the ESTEC prototype implements the CFDP Entity (Server) as a *Delphi Component*.

This component can be easily dragged and dropped inside a Graphical User Interface (GUI). Then, such a GUI (Client) will 'stimulate' the linked CFDP component with protocol *Requests* and will receive protocol *Indications* from it. It is clear that a *Delphi component* can be used only inside a *Delphi IDE*. For this reason, it is anticipated that in the future an *Active X* version of this component will be distributed.

The Active X standard is built on top of COM technology (COM-based) and allows the component to be 'Language Independent'. That is, an Active X component can potentially be used in any kind of development environment (e.g., Visual C++, Visual Basic, etc.).

COM technology defines an API and a binary standard for communication between objects (i.e., a CFDP Entity) that is independent of any particular programming language or (in theory) platform. A COM object consists of one or more 'interfaces', which are essentially

tables of functions associated with that object. Such an object can be implemented from any EXE or DLL.

In this way, a COM mechanism handles all the intricacies of calling functions across process and even machine boundaries. This makes it possible to access an object (CFDP Entity) located on machine A from an application (user software) running on machine B.

This *intermachine* communication method is called DCOM.

Therefore, the DCOM technology discussed in this subsection is strictly used to point out the possibility of using the CFDP component across machine boundaries (see figure 7-13).

7.2.4 OBJECT-ORIENTED PROGRAMMING (OOP) INTRODUCTION

The goal of this introduction to Object-Oriented Programming (OOP) is to provide the fundamental principle on which Delphi's Object Pascal Language is based.

OOP is a programming paradigm that uses discrete objects (an instance of a Class), containing both data and code, as application building blocks. Although the OOP paradigm does not necessarily lend itself to easier-to-write code, the result of using OOP traditionally has been easy-to-maintain code. Keeping an object's data and code together simplifies the process of hunting down bugs, fixing them with minimal effect on other objects, and improving the program one part at time. Traditionally, an OOP language contains implementations of at least three OOP concepts:

a) *Encapsulation*: Deals with combining related data fields and hiding the implementation details. The advantages of encapsulation include modularity and isolation of code from other code.

b) *Inheritance*: The capability to create new objects that maintain the properties and behavior of ancestor objects. This concept enables the creation of object hierarchies such as VCL (first creating generic objects and then creating more specific descendants of those objects that have more narrow functionality). The advantage of inheritance is the sharing of common code.

c) *Polymorphism*: Literally, polymorphism means 'many shapes'. Calls to methods of an object variable will call code appropriate to whatever instance is actually in the variable.

An *Object* is made of:

a) *Fields*: Are data variables contained within objects.

b) *Methods*: The name of procedures and functions belonging to an object. Methods are those things that give an object behavior, rather than just data.

c) *Properties*: A property is an entity that acts as an accessory to the data and code contained within an object. Properties insulate the end user from the implementation details of an object.

7.2.5 CFDP *vs*. OPEN SYSTEMS INTERCONNECTION (OSI) MODEL LAYERING CONTEXT

CFDP can be viewed as a high-level protocol service designed to take advantage of the lower-level protocols, relying on a minimal underlying data communication service (i.e., UDP/IP, CCSDS, etc.).

CFDP correspondent layer is the *Application Service*. It includes portions of the OSI *Session Layer*, *Presentation Layer* and the *Application Layer*, as well as extending into the space above the OSI stack itself traditionally considered to be system application space. Therefore, CFDP can work over several different Packet Transfer Layers, including UDP/IP and CCSDS Standards.

The ESTEC version so far works only over the UDP/IP underlying protocol (see figure 7-10).

UDP was designed to provide a low network-overhead mechanism for transmitting data over the lower layers. Although it still provides packet handling and sequencing services, UDP lacks a number of TCP's more powerful connection-oriented services, such as acknowledgment, flow control and packet reordering (provided by CFDP). The main services offered by UDP can be summarised as follow:

a) Segmenting of data streams (CFDP PDUs) into packets;

b) Reconstruction of data streams from packets;

c) Socket services (Winsock 2.0 creation and manipulation) for providing multiple connections to ports on remote hosts.

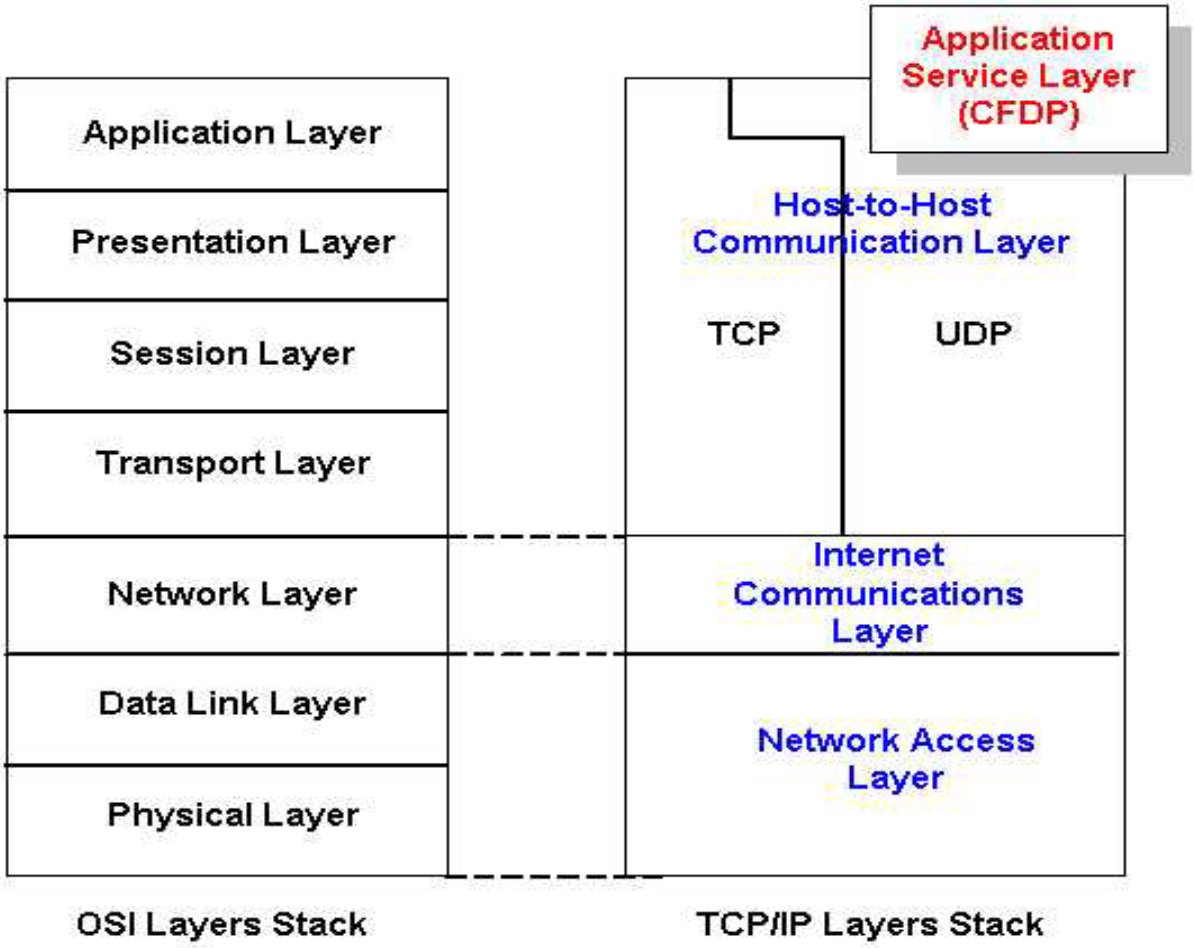


Figure 7-10: Correspondence between CFDP and OSI Layers

The UDP *Host-to-Host Communication Layer* (shown in figure 7-10) handles the services needed to provide reliable communications functionality between network hosts, and corresponds roughly to the *Transport Layer* and part of the *Session Layer* from the OSI model, but it also includes part of the *Application* and *Presentation Layers*.

7.2.6 THE DELPHI *CFDP PACKET SERVICE* COMPONENT

The Delphi CFDP Packet Service Component is a software module specially made to support the interface between the CFDP Entity (another Delphi component) and the underlying Communication System.

The Packet Service Component is connected to both the GUI and the CFDP Component, in a way that can be fully configured by the user software to operate over a set of various underlying communication systems (UDP, CCSDS, etc.), relaying CFDP packets from and to the underlying communication system (see figures 7-11 and 7-12).

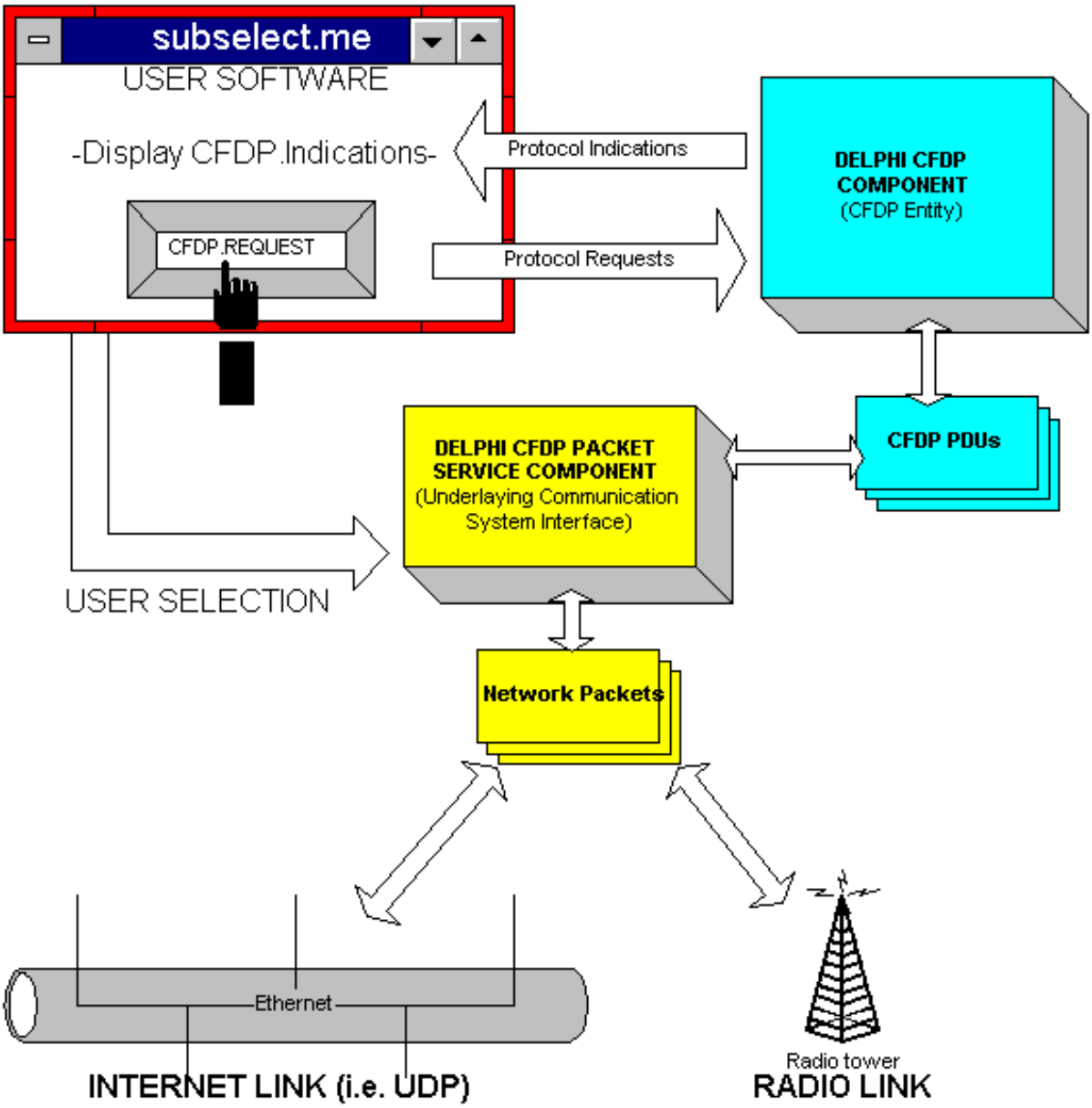


Figure 7-11: Packet Service Component Functional Diagram

In other words, it is in charge of handling all the procedures related to CFDP PDUs sending and receiving over the *underlying protocol Layer* (currently UDP/IP Host-to-Host Communication Layer for the ESTEC Version).

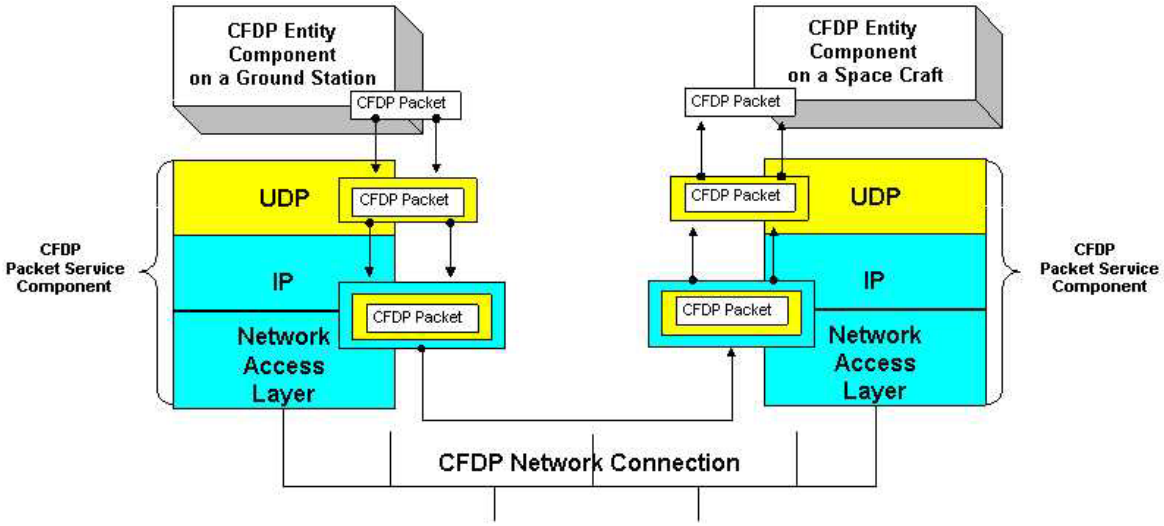


Figure 7-12: Sender-Receiver CFDP Packet Flow

Hence, the CFDP *Unit Data* (*UT) Transfer Layer Interface* can be associated with the CFDP Packet Service software component linked to the CFDP Entity Component.

Obviously, the *error handling method* for such an interface is the one related to the selected underlying layer (i.e., the error handling rates for UDP delivery and duplicate protection are not guaranteed).

So far, no *Flow Congestion Control* is performed within the UT interface, but it is anticipated that it will be implemented soon as possible (surely before any interoperability test) as a *'bit rate control mechanism'* inside the CFDP Packet Service component's sending routine. This mechanism will permit a limit on the maximum number of packets flowed on the network, and will be used to measure the CFDP bandwidth efficiency through measurements of the total bandwidth and transaction time.

NOTE – A project for integration of CFDP over a *Telecommand* (*TC) Packet Simulator* is currently being implemented in the European Space Operations Centre (ESOC).

7.2.7 THE DELPHI CFDP COMPONENT

7.2.7.1 Component Description

The CFDP component is a stand-alone software module representing the CFDP behavior (Core and Extended Procedures), with a well-defined interface to the outside world.

Using a component implies the creation of an *Object instance*. It is obtained by calling the constructor method (TCFDP.Create) of the *Class* (TCFDP) representing the object.

The *Public/Published* methods and properties of the *TCFDP* Class represent such an interface.

Public and Published are visibility specifiers. The Public methods and properties can be accessed only at run-time, while the Published properties (no methods) are also accessible from inside the IDE at design-time. This means that is possible to set the component's published values before running the application in order to make use of an object (i.e., User Interface software).

A Delphi component can only be used in a Delphi IDE.

In order to work properly, the CFDP component needs to be 'used' within an application (i.e., the User Interface software). In other words, it receives *stimulus* from user software in order to perform actions, and it raises *events* when a certain state is reached. Thus, according to the CFDP world, stimulus can be associated to all the *CFDP Request Service Primitives* and events can be associated to all the *CFDP Indication Service Primitives* (see figure 7-13). When an event is raised from the CFDP Component, the connected User Software shall be able to handle it and to undertake actions according to the event type (i.e., an info display on the user interface). This can be done by assigning to each Component's event an *event handler procedure* belonging to the User Software. When doing so, the CFDP's User should be aware that, during the processing time of the *event handler procedure*, the component itself is 'waiting' to return to continue its normal flow of execution. Considering this time-constraint, the code executed inside an event handler has to be built in order to minimize its execution time and, in turn, to reduce as much as possible its impact on the CFDP's performance.

7.2.7.2 Instructions for Use of the Component

7.2.7.2.1 Initialization

Once the User Software owns a CFDP Component object (i.e., a CFDP Component has been dragged and dropped on the User Interface form at design-time) an *instance* of the TCFDP Class has to be created at *run-time,* using the TCFDP.Create method. This method will perform all the initialization procedures of a CFDP Entity. From now on the CFDP Component is ready to be set, to receive stimulus and to raise events according to its public Properties and Methods.

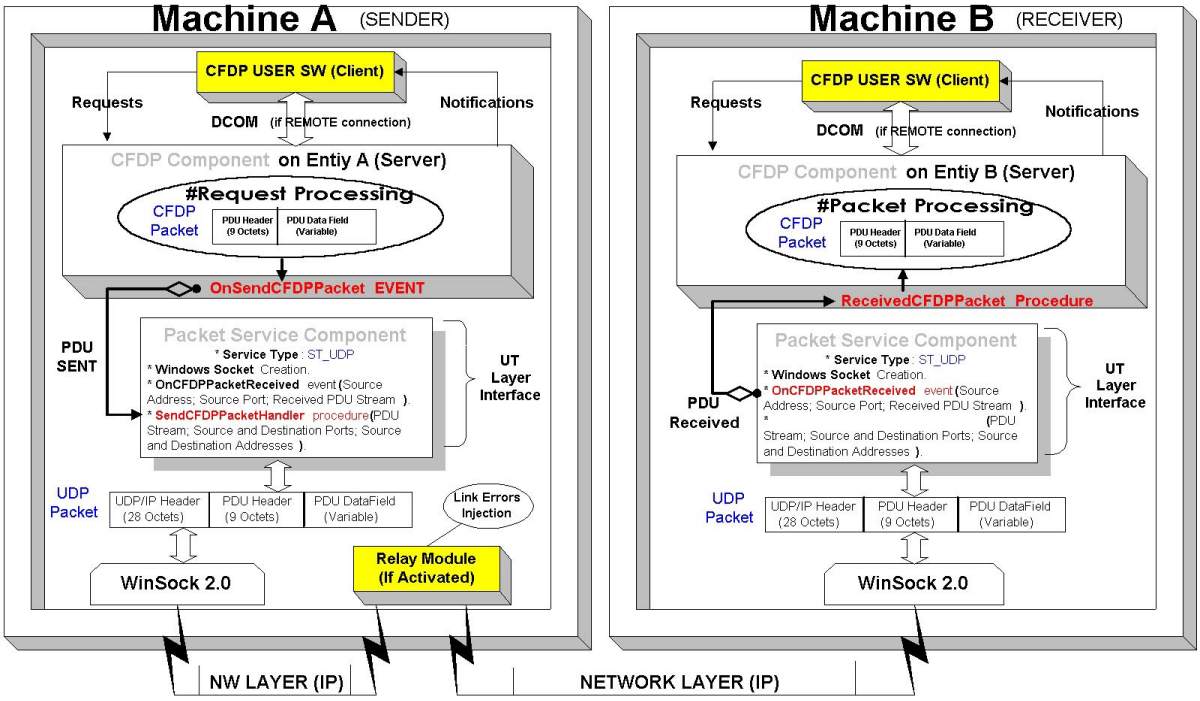


Figure 7-13: CFDP Software Elements (Components) Diagram and Packets Flow

7.2.7.2.2 The CFDP Log Window

During the initialization phase, the CFDP Component automatically creates a Log Window to display its status and all the run-time information for all of the file delivery Transactions handled by the current CFDP Entity (types of PDUs sent and received, error log, etc.). See figure 7-14.

The CFDP Log window is divided in three parts:

a) General Log (upper part).

b) Receiving Transactions Log (left side).

c) Sending Transaction Log (right side).

The General Log part displays all of the entity-oriented messages such as entity status, capabilities, and settings values, as well as messages on transaction start and end, acknowledgment timers, etc.

The other two parts are more Transaction-oriented, displaying details on each received or sent packet for both the file Sending and file Receiving transactions. Figure 7-15 shows the Server Log at run-time.

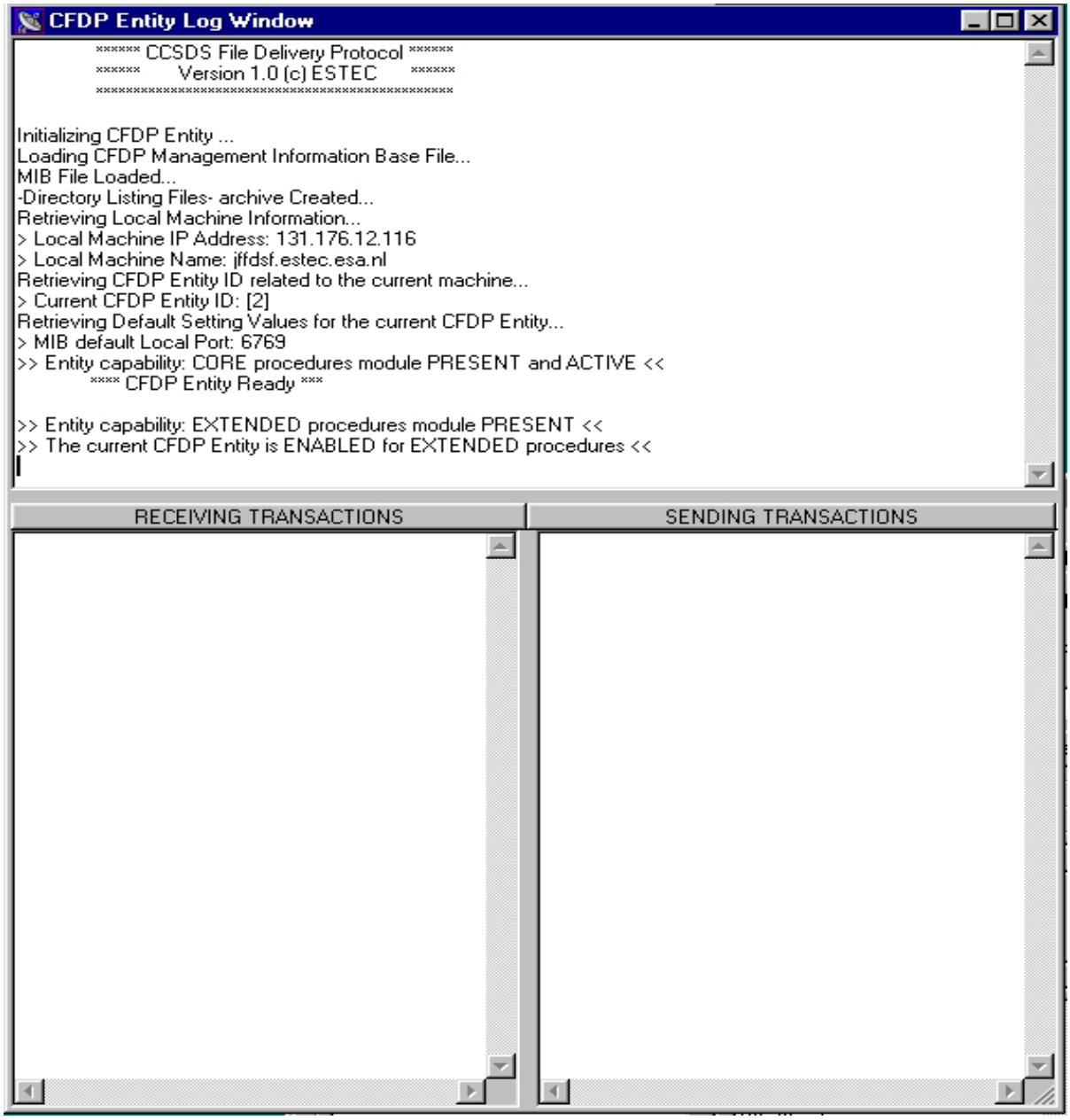


Figure 7-14: CFDP Components Log Window

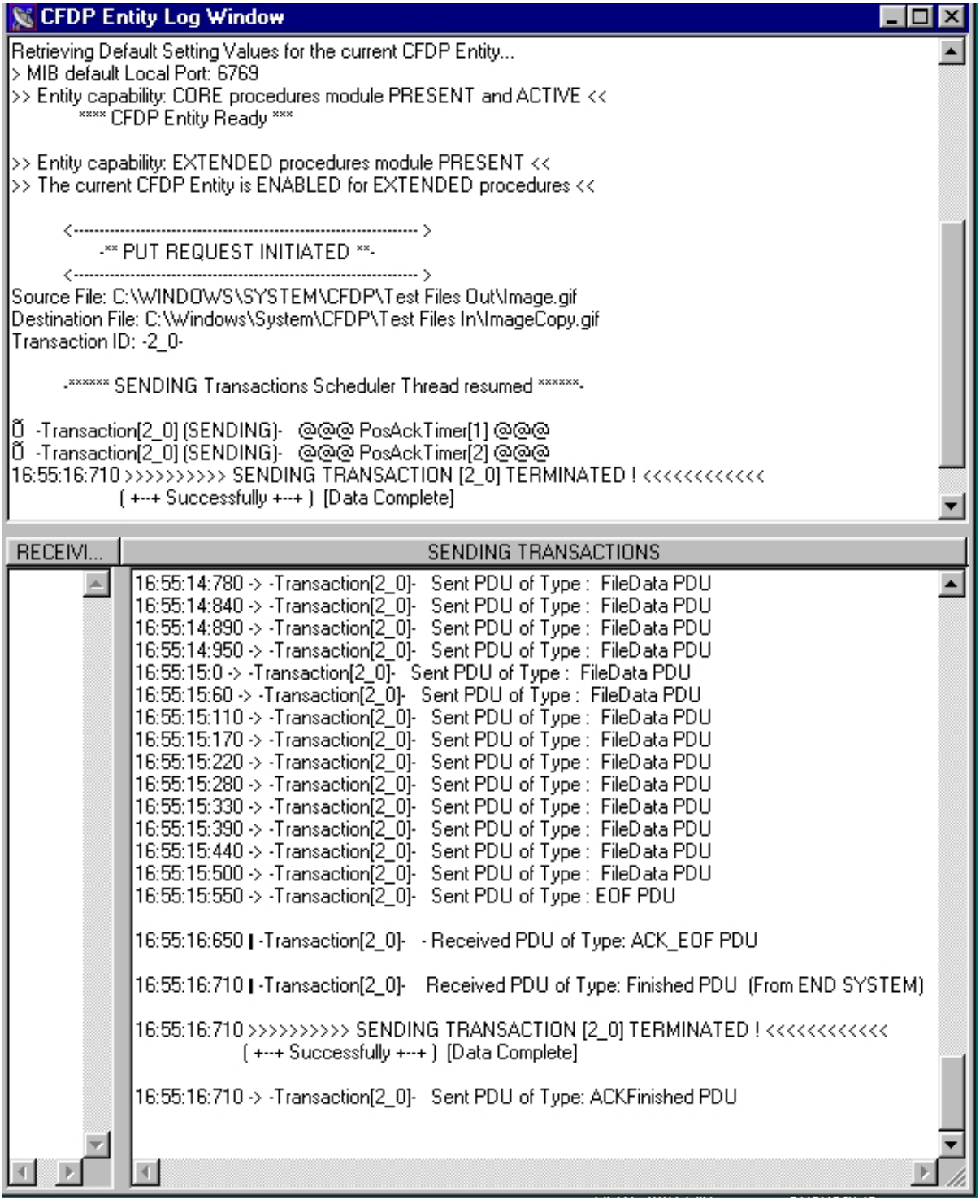


Figure 7-15: CFDP Server Log at Run-Time

7.2.7.2.3 CFDP Public Properties

7.2.7.2.3.1 List

a) property EntityLogForm: TCFDPLogForm read FLogForm;

b) property CurrentNumOfTrans: LongWord read FCurrentTransNum;

c) property CurrentNumOfRCVTrans: LongWord read FCurrentRCVTransNum;

d) property ActiveTranIDsList: TList read GetActiveTranIDs;

e) property RecvTranIDsList: TList read GetRecvTranIDs;

f) property SourceFilePath: String read FSourceFilePath write FSourceFilePath;

g) property DestFilePath: String read FDestFilePath write FDestFilePath;

h) property MIBFileString: String read FMIBString;

i) property RunMultipleTran: Boolean read FRunMultipleTran write FRunMultipleTran.

7.2.7.2.3.2 Properties Related To Transmission Mode Information

a) property ApplyCRC: boolean read FApplyCRC write FApplyCRC;

b) property ApplyAckModeIN: boolean read FApplyAckModeIN write FApplyAckModeIN;

c) property ApplyAckModeOUT: boolean read FApplyAckModeOUT write FApplyAckModeOUT;

d) property NAKType: TNAKType read FNAKType write FNAKType;

e) property NAKReissueTO: LongWord read FNAKReissueTO write FNAKReissueTO;//In Seconds;

f) property NAKMaxNum: LongWord read FNAKMaxNum write FNAKMaxNum;

g) property KAlivePeriod: LongWord read FKAlivePeriod write FKAlivePeriod;//In Seconds;

h) property KAliveMaxOffset: LongWord read FKAliveMaxOffset write FKAliveMaxOffset;//In Bytes;

i) property PosAckReissueTO: LongWord read FposAckReissueTO write FPosAckReissueTO;//In Sec;

j) property PosAckMaxRetries: LongWord read FposAckMaxRetries write FPosAckMaxRetries.

7.2.7.2.3.3 Properties Related To CFDP TLV Options

a) property MsgsToUserStream: TStream read GetMsgsToUserStream;

b) property MsgsToUserTxtNum: Byte read FMsgsToUserTxtNum;

c) property ProxyMsgsNum: LongWord read FProxyMsgsNum;

d) property RemoteValid: Boolean read GetRemoteValid;

e) property RemotePutOrder: TRemotePutOrder read FRemotePutOrder;

f) property RemoteMsgsToUserTxtNum: Byte read FRemoteMsgsToUserTxtNum;

g) property RemoteFileStReqsNum: Byte read FRemoteFileStReqsNum;

h) property RemoteFaultHndOvsNum: Byte read FRemoteFaultHndOvsNum;

i) property RemoteSegmCtrlPresent: Boolean read FRemoteSegmCtrlPresent;

j) property RemoteTranModePresent: Boolean read FRemoteTranModePresent;

k) property RemoteSegmCtrlValue: Byte read FRemoteSegmCtrlValue;

l) property RemoteTranModeValue: Byte read FRemoteTranModeValue;

m) property DirListRequestsNum: Byte read FDirListRequestsNum;

n) property DirListFilesDirectory: String read FDirListFilesDirectory;

o) {*** END OF NORMAL & PROXY MESSAGES TO USER'S PROPERTIES ***}

p) property FileStReqsNum: Byte read FFileStReqsNum;

q) property FaultHndOvsNum: Byte read FFaultHndOvsNum;

r) property FlowLabelTLV: String read FFlowLabelTLV write FFlowLabelTLV.

7.2.7.2.4 CFDP Published Properties List

a) {***** PUBLISHED PROPERTIES ******}

b) property CFDPPacketLength: Word read FCFDPPacketLength

c) write SetCFDPPacketLength;

d) property InBufferSize: Cardinal read GetInBufferSize write SetInBufferSize;

e) property OutBufferSize: Cardinal read GetOutBufferSize

f) write SetOutBufferSize;

g) property RCVInBufferSize: Cardinal read GetRCVInBufferSize

h) write SetRCVInBufferSize;

i) property RCVOutBufferSize: Cardinal read GetRCVOutBufferSize

j) write SetRCVOutBufferSize;

k) property CFDPEntityID: integer read FCFDPEntityID write FCFDPEntityID;

l) property SourceEntityID: LongWord read FSourceEntityID write FSourceEntityID;

m) property DestinationEntityID: LongWord read FDestEntityID

n) write FDestEntityID;

o) {To Be retrieved from the MIB}

p) property LocalIPAddress: String read FLocalIPAddress;

q) property LocalMachineName: String read FLocalMachineName;

r) property LocalPort: integer read GetLocalPort write SetLocalPort;

s) property FaultAction: TProtocolFaultAction read FFaultAction

t) write FFaultAction;

u) property CancelAction: TCancelAction read FCancelAction write FCancelAction;

v) property TranLifetime: LongWord read FTranLifetime write FTranLifetime.

7.2.7.2.5 CFDP Public Methods List (TBS)

7.2.7.2.5.1 Methods Related To CFDP TLV Options (TBS)

7.2.7.2.6 CFDP Events List

a) *OnSendNetworkPacket*: TsendNetPacketEvent

b) Events **related to the CFDP Indications**

1) *OnTransactionStart*: TTranStartEvent;

2) *OnMetadataReceived*: TMetadataReceivedEvent;

3) *OnFileSegmentReceived*: TFileSegmentRecvEvent;

4) *OnTransactionSuspended*: TTranSuspendedEvent;

5) *OnTransactionResumed*: TTranResumedEvent;

6) *OnTransactionFinished*: TtranFinishedEvent;

7) *OnTransactionReport*: TNotifyEvent.

8) *OnTransferConsigned*: TNotifyEvent;

The OnFailure event is raised when a CFDP method failed to be executed. If the Handled variable is set to True from the user event handler, then an error is not raised from the CFDP entity.
- *OnFailure*: TFailureEvent;

The OnError event is raised when an error occurs during a FDU transmission or a CFDP Entity initialization. The error severity level is passed back from the CFDP entity together with the error code number and a Message string to define the error.
- ***OnError***: TErrorEvent .

7.2.7.3 Software Architectural Design (SAD)

7.2.7.3.1 General

The Software Architectural Design (SAD) described in this subsection is focused on the CFDP Component software description (Core procedures only).

The SAD reflects what is inside the CFDP *black box* (Component) from the point of view of the implementer, rather than the user. In the future, the SAD will allow implementers to easily maintain the code, as well as to locate where to add new features to the component's capability.

In order to perform an object-oriented analysis of a software module, *class diagrams* have been created for the Main Protocol's class (TCFDP) and all its *nested* classes, which are only available for use within the scope of the main class and are hidden from the user (encapsulation).

This type of diagram provides a means of representing a *Class Logical View*; explaining its functionality and its relationship with other classes.

Moreover, since the entire CFDP 'classes set' is existing in a *Multithread* contest, further diagrams representing threads function and interaction, as well as the CFDP packets In/Out flow, are present.

7.2.7.3.2 CFDP's Classes

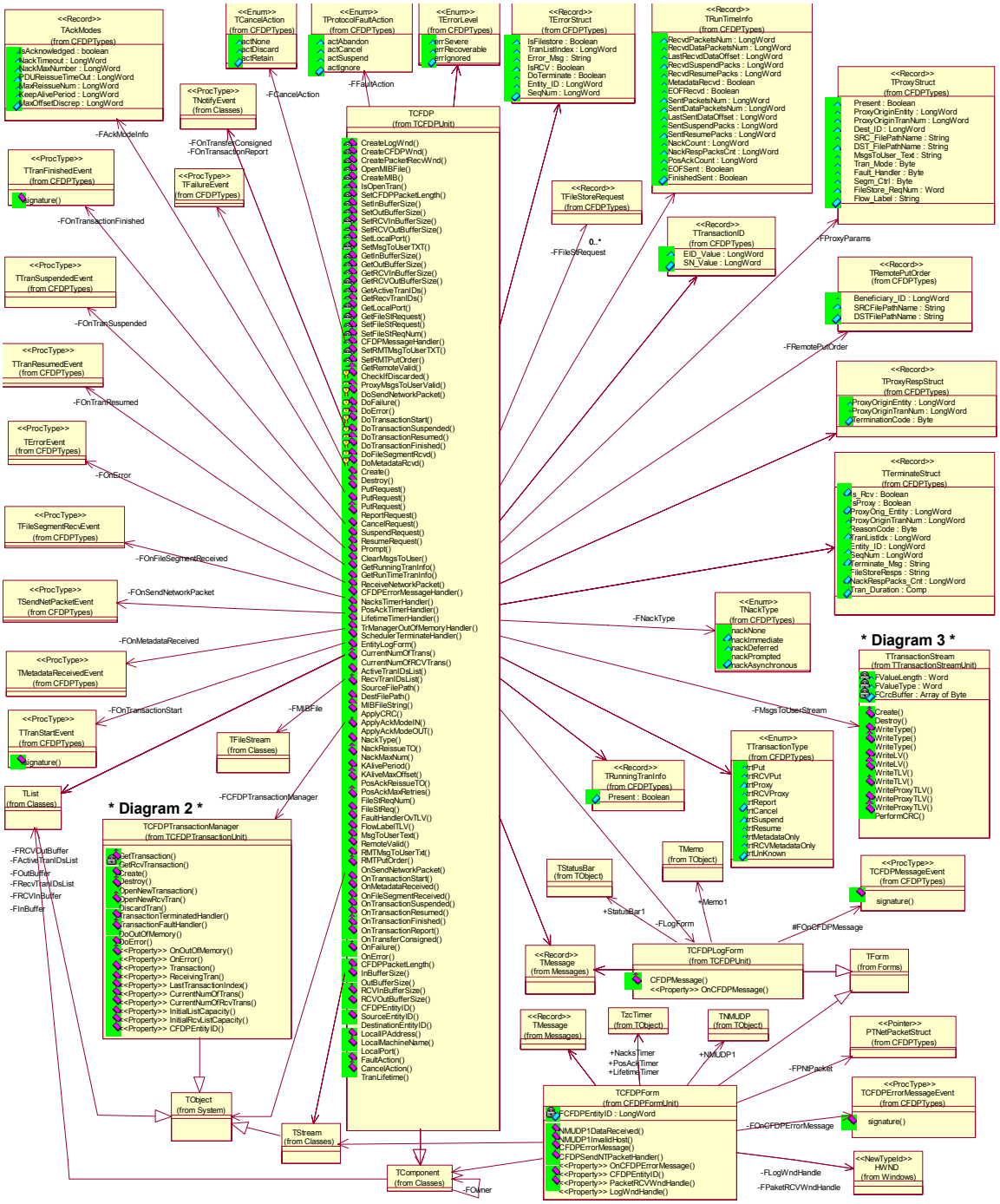
A class is a set of objects that share a common structure and a common behavior (the same attributes, operations, relationship and semantics). It is an abstraction of real-world items. When these items exist in the real world, they are instances of the class (with single or multiple cardinality) and are referred to as objects. Thus, the class representing a CFDP

Entity has a single cardinality, while the class representing an FDU Transaction within the CFDP Entity itself has multiple cardinality.

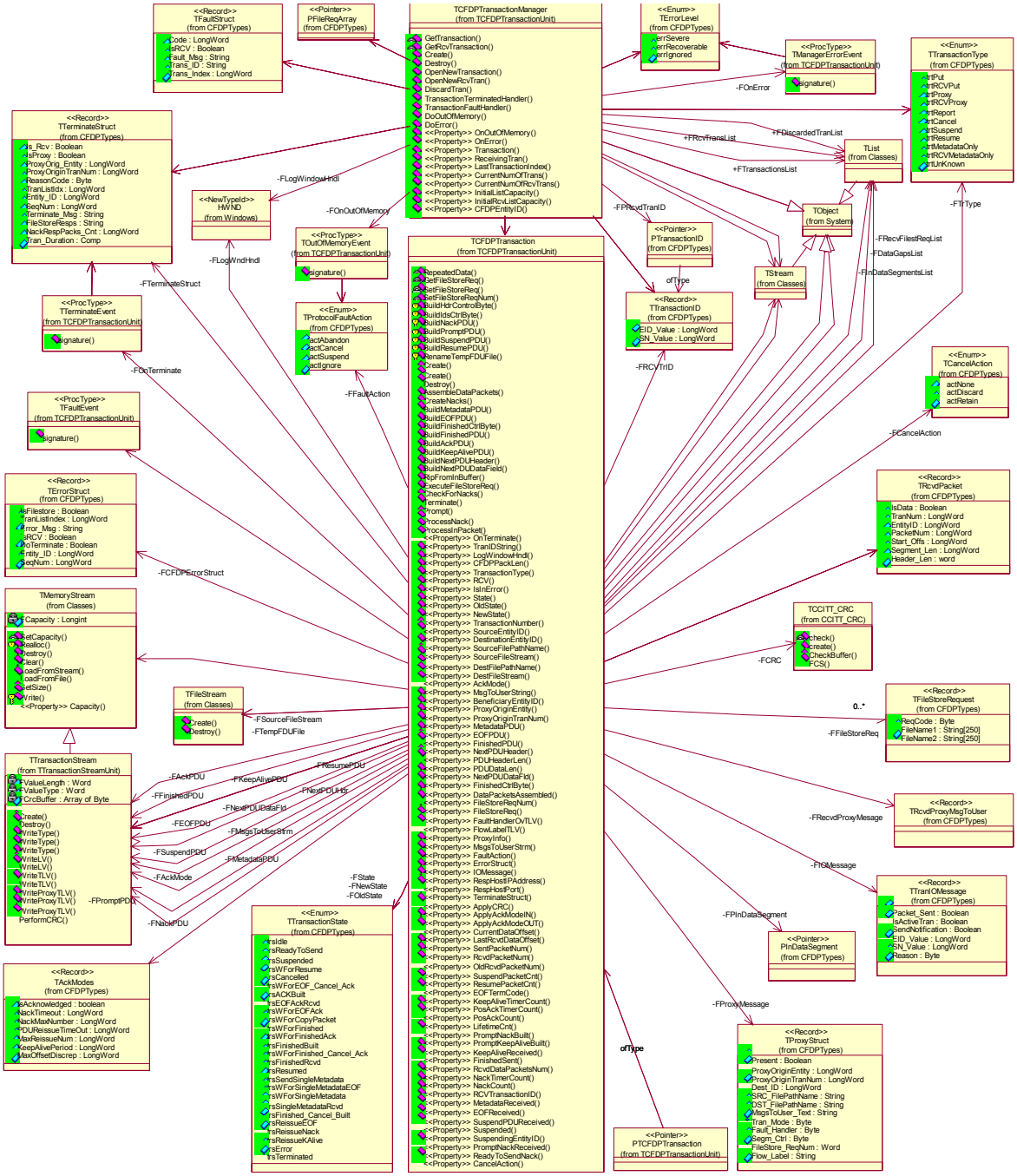
Considering the Object-Oriented nature of the code used for the CFDP Software Module implementation, the component's interface can be seen as the 'public view' of the main class defining it (TCFDP).

This means that all the *CFDP Requests* primitives are associated to class *Methods*, all the *CFDP Notifications* to class *Events*, and all the *CFDP Settings* values to class *Properties*.

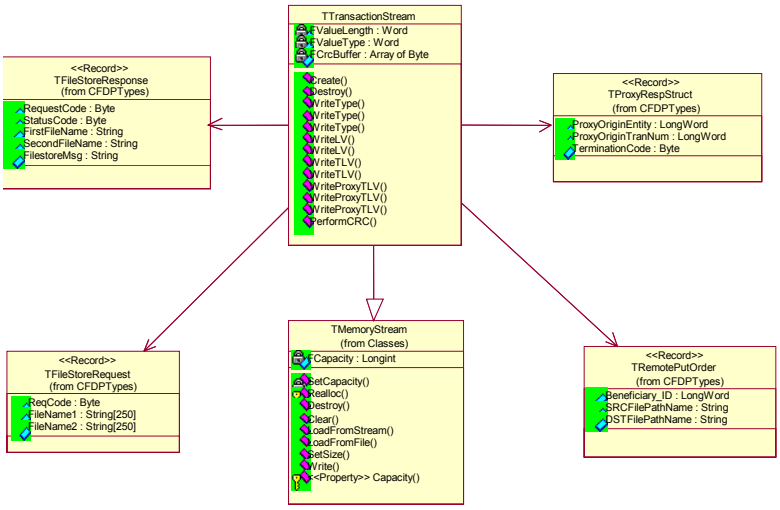
7.2.7.3.3 TCFDP Class Diagram (Main)



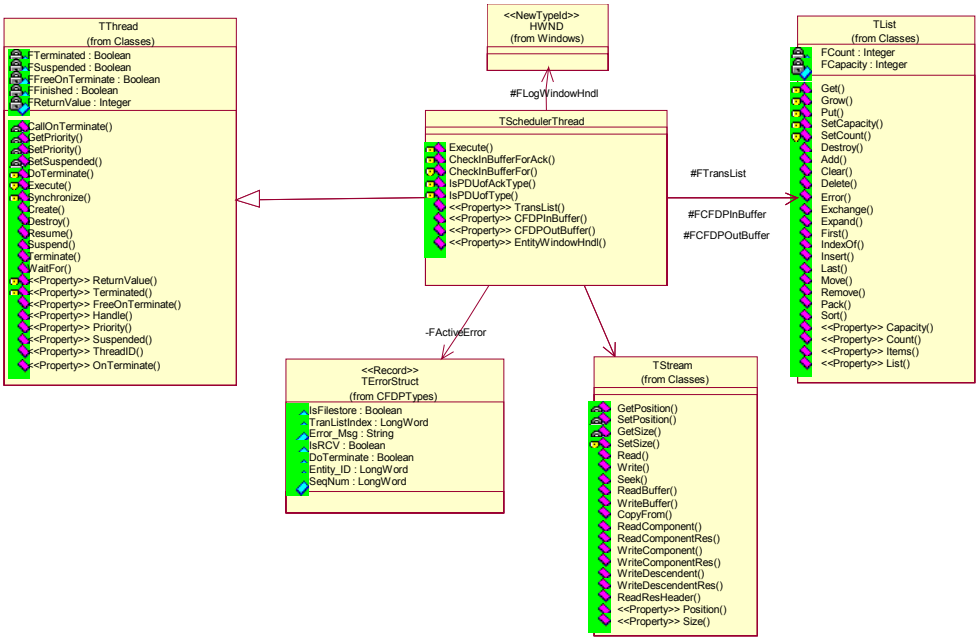
7.2.7.3.4 TCFDPTransactionManager & TCFDPTransaction Classes Diagram (2)



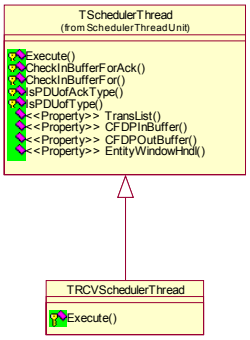
7.2.7.3.5 TTransactionStream Class Diagram (3)



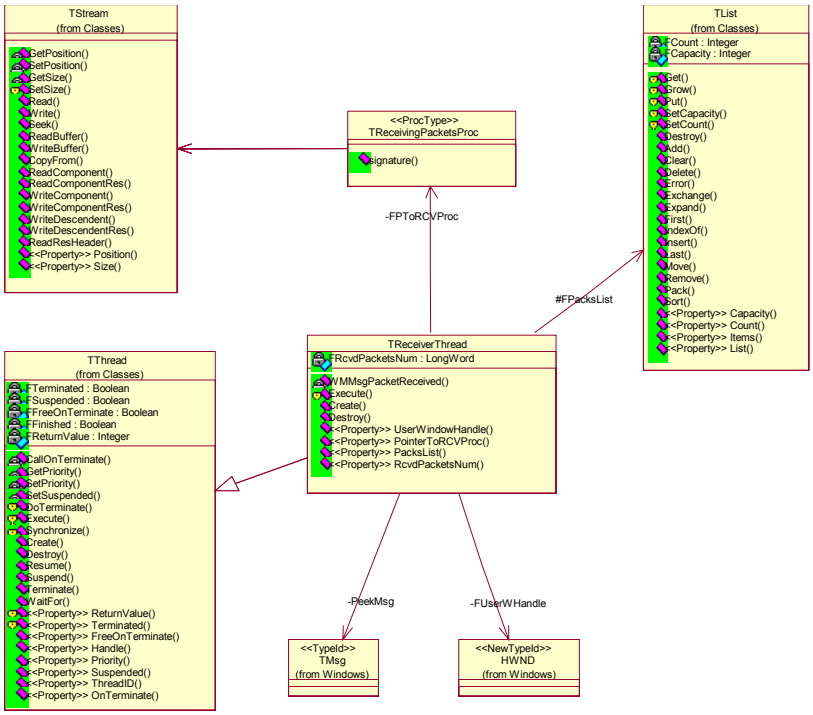
7.2.7.3.6 TSchedulerThread Class Diagram



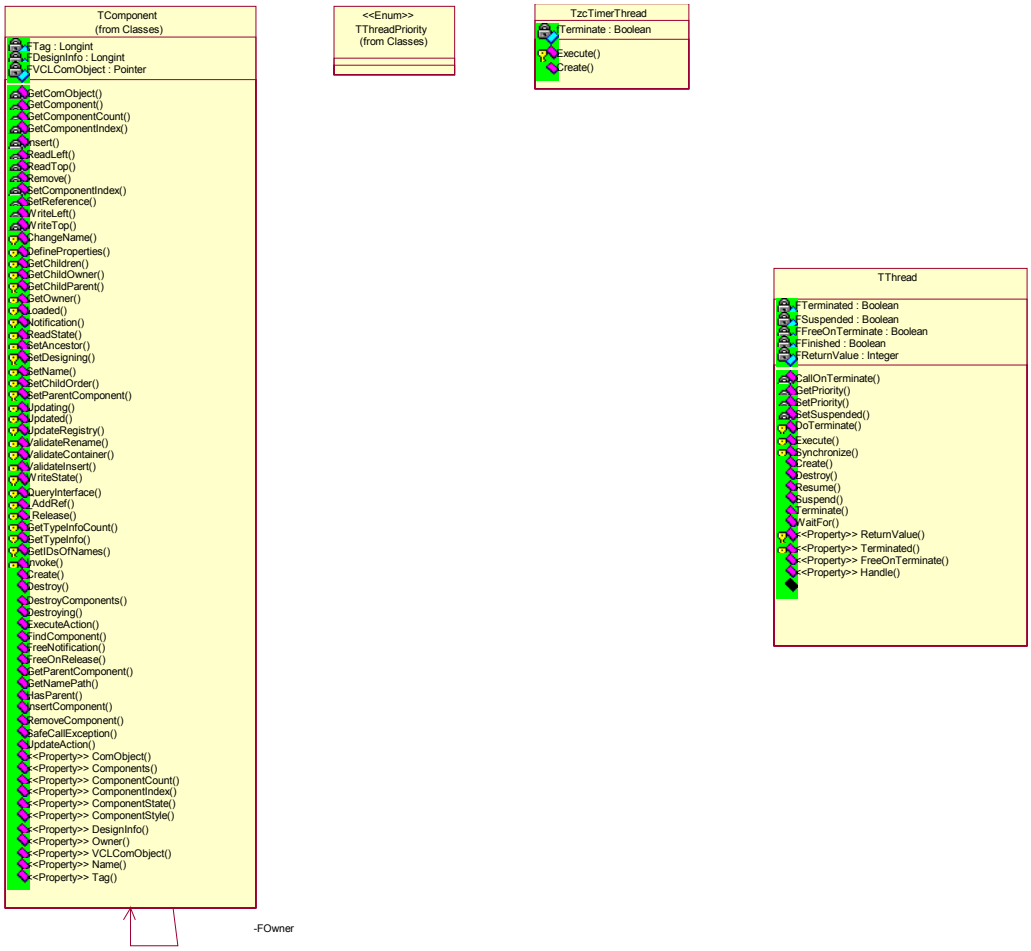
7.2.7.3.7 TRCVSchedulerThread Class Diagram



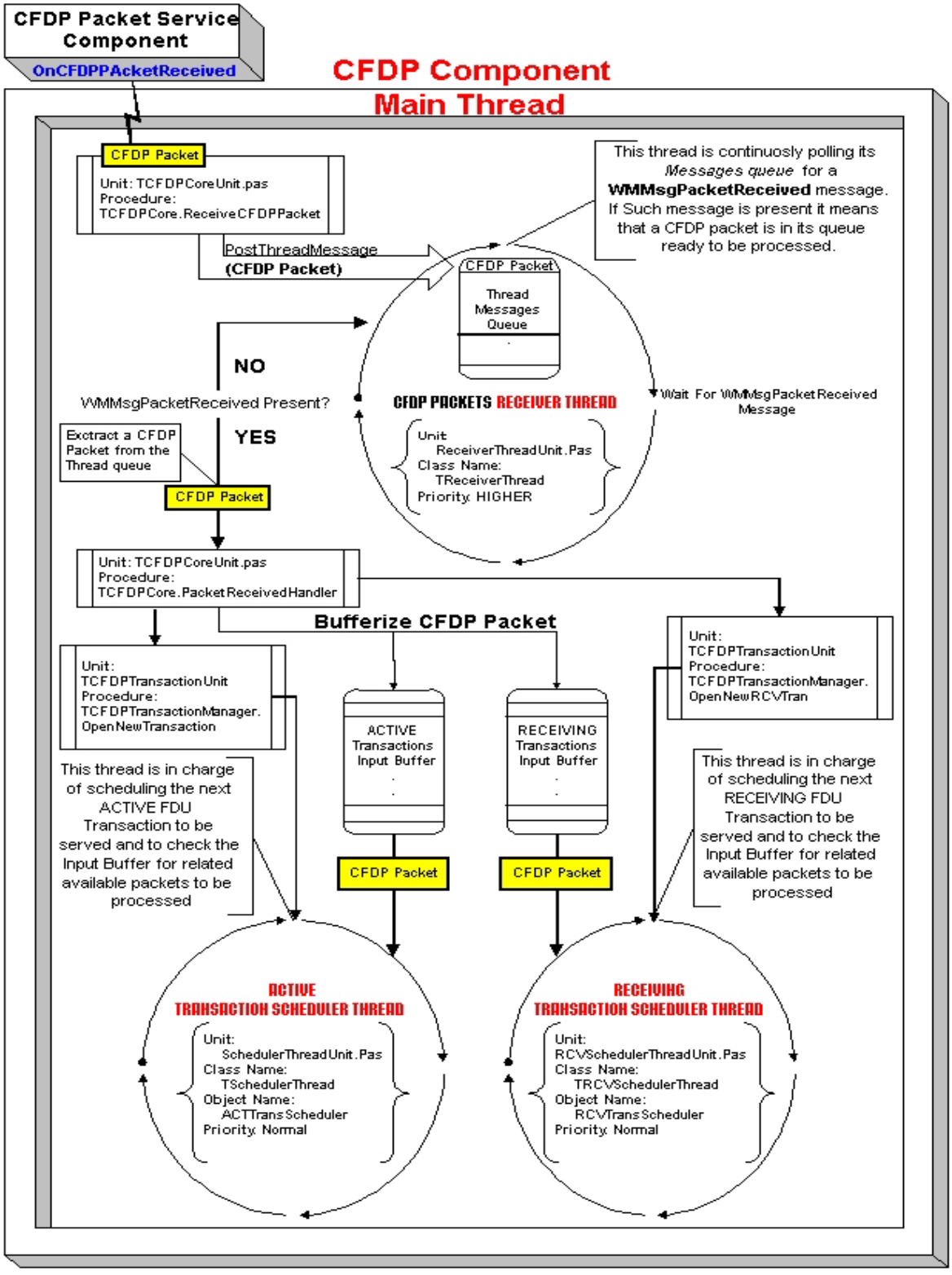
7.2.7.3.8 TReceiverThread Class Diagram



7.2.7.3.9 CFDP Timer Class Diagram

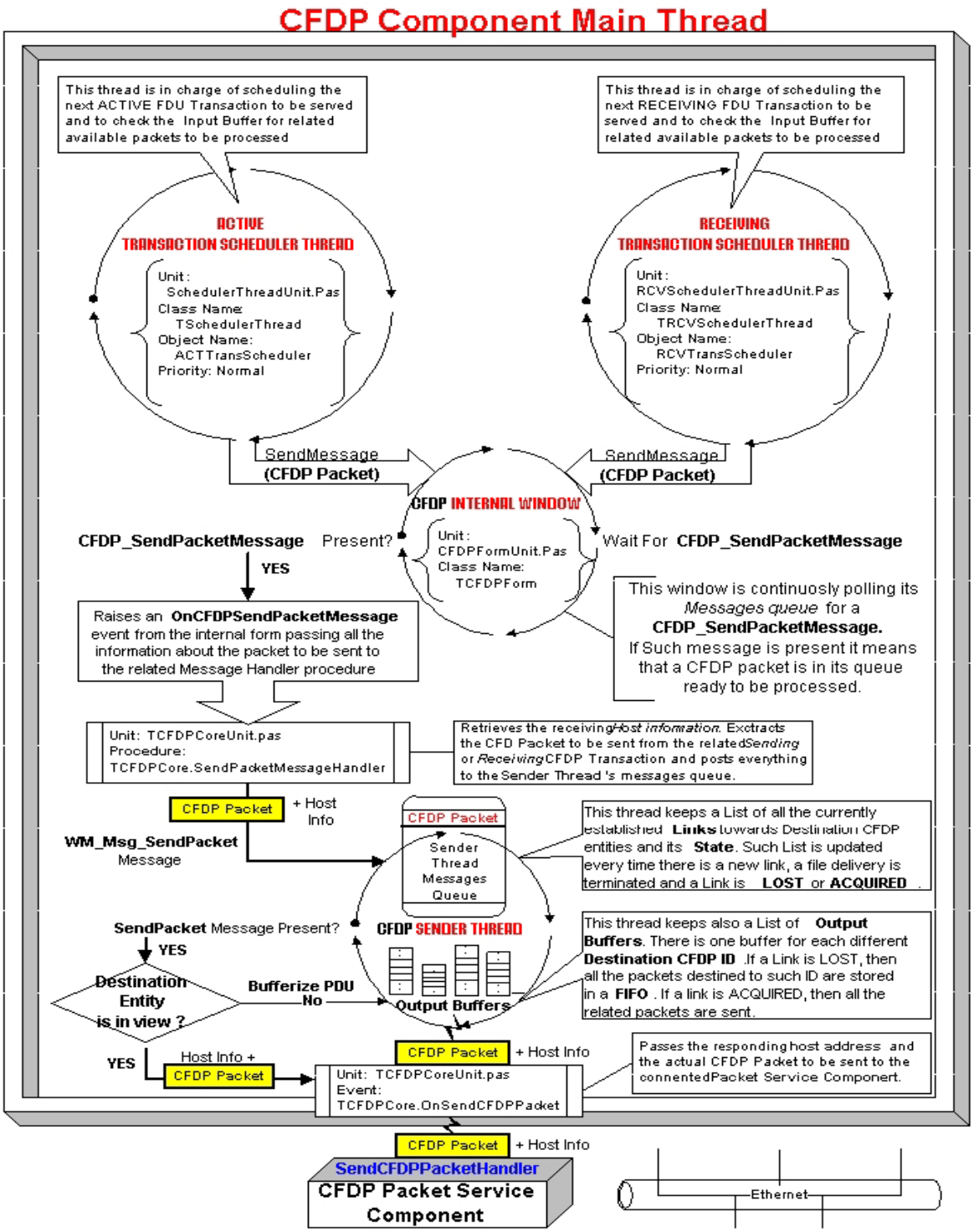


7.2.7.3.10 CFDP Threads Interaction and Input Packets Flow Diagram



7.2.7.3.11 CFDP Packets Output Flow

7.2.7.3.11.1 Diagram



To fully understand the output flow of CFDP packets inside the CFDP software component, a brief description on how such a component handles all the internal *messages* is necessary.

7.2.7.3.11.2 The CFDP Component's Internal Window

In order to perform proper actions upon the occurrence of a certain event, and due to the multi-threading nature of the CFDP component, user-defined Windows messages are sent from all the secondary threads to a *component's internal window.* In this way, the message can be processed within the component's main thread, avoiding violation of shared resources. Hence, the CFDP internal window can be seen as a non-visible 'housekeeping' window, performing all the component's message-handling procedures.

The message types handled by this form are as follows:

a) CFDP_ExtendedMESSAGE;

b) CFDP_SendPacketMESSAGE;

c) CFDP_ErrorMESSAGE;

d) CFDP_TimerMESSAGE.

7.2.7.3.11.3 An Outgoing PDU Through CFDP

The *CFDP_SendPacketMESSAGE* contains general information about the outgoing packet and is sent from within the two schedulers threads (for *Sending* and *Receiving* transactions) any time a CFDP packet is ready to be sent over the underlying communication system. Upon receipt of such message, the internal form raises an OnCFDPSendPacketMessage event. Then, the connected event handler procedure (TCFDPCore.CFDPSendPacketMessageHandler) is called within the main thread in order to:

a) Retrieve the receiving host's network address;

b) Extract the outgoing CFDP packet from the Transaction object;

c) Store everything in a memory structure together with other information (Transaction ID, Destination ID, Packet number, etc.);

d) Post a WM_Msg_SendPacket message containing the address of the packet memory structure to the *Sender thread messages queue* and return.

All the messages posted to a *thread message queue* are buffered before the thread itself processes them. The Sender thread main code is a loop that is continuously polling for WM_Msg_SendPacket messages.

When such a message has arrived, if the packet Destination ID is currently 'In View', the packet is *released* right away on the underlying communication layer, and the next pending message is processed. If the Destination CFDP entity is not in view, then the packet is

buffered in output queues made by persistent First In First Out (FIFO) linked lists and will wait for the next link acquisition. The FIFOs grow while links are inactive and shrink while they are active, but this is transparent to applications using the *CFDP Component*.

An ***Output Buffer*** is created for each CFDP destination entity involved in a file delivery transaction.

It contains outgoing packets belonging to both *Sending* and *Receiving* transactions handled by the local CFDP entity.

However, the use of output buffers can be disabled (both during design and run time) in case the CFDP component is running on a storage-constrained entity.

The CFDP component is also responsible for keeping track of all the new established, lost, acquired or dismissed links towards different destinations. This task is carried on partly by the *CFDP component* itself (in case of new and dismissed links upon file delivery transaction opening/closing), and partly in conjunction with the *User Software* (for lost and acquired links during transactions lifetime).

In the latter case, knowledge of the Link State is delivered to CFDP by mean of two functions:

a) LinkLost (Remote_CFDP_ID);

b) LinkAcquired (Remote_CFDP_ID).

Obviously, this implies that such knowledge already exists outside of CFDP.

Both *Core* and *Extended* procedures (i.e., store and forward functionality, especially with parallel waypoints) will benefit such a *Deferred Transmission mechanism*, giving the CFDP a way to ‘drive’ the starting and stopping of PDUs transmission and schedule the file delivery transactions according to an arbitrary priority scheme.

Furthermore, the use of output buffers can also be enabled and disabled by the user also at run-time. Unfortunately, this implies the loss of all the packets currently stored in the outgoing FIFOs (see figure 7-16).

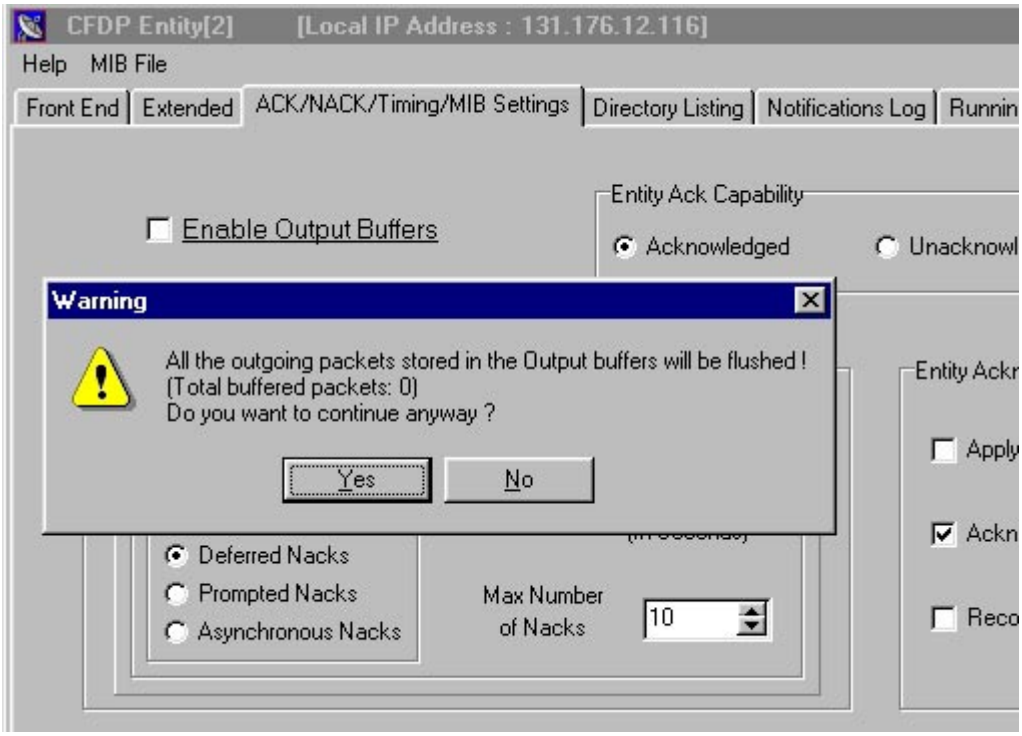


Figure 7-16: Output Buffers Enable

7.2.7.3.11.4 Advantages of Output Buffers

The same approach for buffering outgoing PDUs is adopted in the Jet Propulsion Laboratory (JPL) implementation. The following is a comprehensive recap of advantages listed in *Notes on CFDP Implementation* (from Scott Burleigh, JPL 24 July 2000).

By relying on link state cues to control the operation of File Delivery Protocol Output, we can accommodate occultation and other interruptions in connectivity simply and efficiently: when the link is lost, CFDP simply stops transmission and reception of data between the two endpoints of the link.

This implementation of deferred transmission incurs far less protocol overhead than using the *Suspend* and *Resume* PDUs to control suspension and resumption of communication:

a) Suspend and Resume procedures are protocol elements, requiring a co-operative interchange of data between entities. Deferred transmission is entirely local; no PDUs are issued or received to affect it.

b) Because deferred transmission is an entirely local mechanism, it is unaffected by delay due to the distance between the participating entities. Moreover, there is no chance of incomplete suspension/resumption due to loss of a PDU.

c) Suspend and Resume procedures are transaction-specific. This means that a link cut between any pair of CFDP entities would require the reliable transmission of Suspend

PDUs for every transaction currently in progress between them, and resumption of transmission would require the reverse. In contrast, the deferred transmission mechanism is atomic and comprehensive.

7.2.7.3.11.5 Flow Control Integration in CFDP

By using these new CFDP features for driving packet flow towards selected destinations, an efficient *Flow Control* mechanism in which the receiver provides feedback to the sender can easily be implemented, in order to throttle the sender into sending no faster than the receiver can handle the traffic.

Assuming that the communication channel is error free, and if the used link has an uneven data throughput (i.e., Packet Telecommand link), a good solution could be to implement a *Stop-and-wait* flow control protocol. In such a mechanism the sender sends one frame and then waits for an acknowledgement before proceeding. It can be accomplished by using the *LinkLost()* and *LinkAcquired()* procedures available in the CFDP component.

These capabilities would spare the CFDP component the use of an 'embedded' Flow Control algorithm. In other words, the packets received by the CFDP component would still be CFDP packets, since the flow control header has been read and filtered by an 'external' software module in charge of driving the CFDP packet flow via the *LinkLost* and *LinkAcquired* functions.

In this way the Flow Control will remain transparent to the CFDP component itself.

7.2.7.3.11.6 Transactions Priority Considerations

The PDUs pending in the sender thread's message queue are already stored in a PRIORITY order, which has been assigned from the Receiving or Sending Transactions SCHEDULERS. If such PDUs are extracted and buffered because of a link visibility cut, then the previously assigned priority is lost.

This happens because the ordering key for buffered PDUs is no longer their Transaction ID but, instead, their Destination ID. Therefore, all the PDUs stored in an Output Buffer will belong to transactions of different natures and IDs.

In other words, a new kind of priority is established between *buffered* outgoing PDUs:

The Output Buffer CREATION ORDER.

In practice, PDUs destined to a Transaction ID that was *out-of-view* in a moment 'X' will be released (as soon the link is acquired again) before the PDUs destined to a Destination ID that was *out-of-view* in a moment 'X+Y'.

7.2.8 GRAPHICAL USER INTERFACE DESCRIPTION

7.2.8.1 General

The CFDP User Interface is a Windows Application composed of 8 different sub-windows:

a) Front End (Main);

b) Extended;

c) ACK/NAK/Timing/MIB Settings;

d) Directory Listing;

e) Notifications Log;

f) Running Transaction Info;

g) Test;

h) Packet Service.

7.2.8.2 Front End Tab Window

This window (figure 7-17) can be defined as the 'main entrance' of the CFDP User Software Interface.

It allows for specification of all the basic parameters for a FDU transaction (Source and Destination ID, Source and Destination File Name, creation of Metadata TLV Options, etc.).

Moreover, it provides information about the CFDP network address (IP and Port) of the selected Destination ID according to the loaded MIB File's settings, as well as general information on the local machine system.

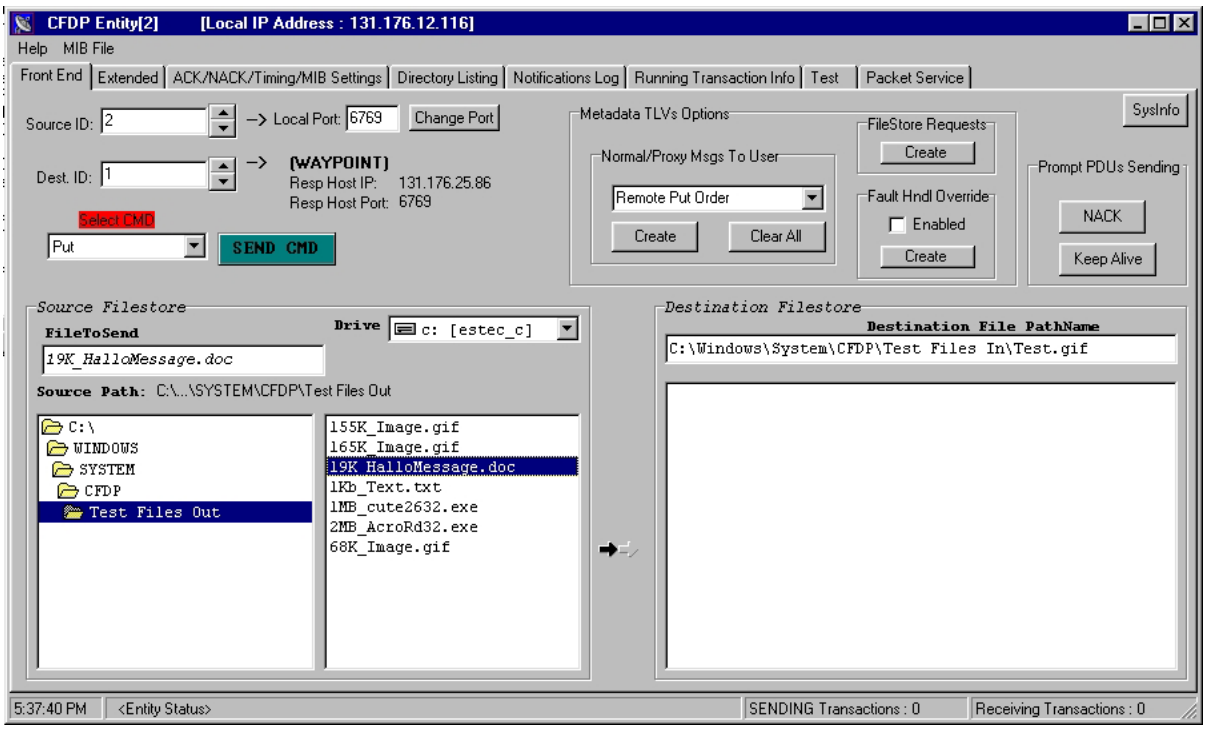


Figure 7-17: Front End Tab Window

7.2.8.3 Fault Handlers Overrides

From the CFDP User software it is possible to set up three different types of *Fault Handlers Overrides*:

a) TLV;

b) Remote TLV;

c) Local.

The *TLV Fault Handler Override Options* are contained in the Metadata PDU, and are fault actions specified by the Source CFDP entity in order to override the local fault actions undertaken by the Destination CFDP entity in case a protocol error occurs.

The *Remote TLV Fault Handler Override Options*, also contained in the Metadata PDU, are similar to the normal TLV Options but they are created by the *Originator* of a *Proxy* transaction (Remote Put) in order to be sent from the *Proxy Source* CFDP entity to the *Proxy Destination* CFDP entity.

These first two types of Fault Handler Override Options can be created from the Front End Tab Window of the CFDP User Software (see figure 7-18).

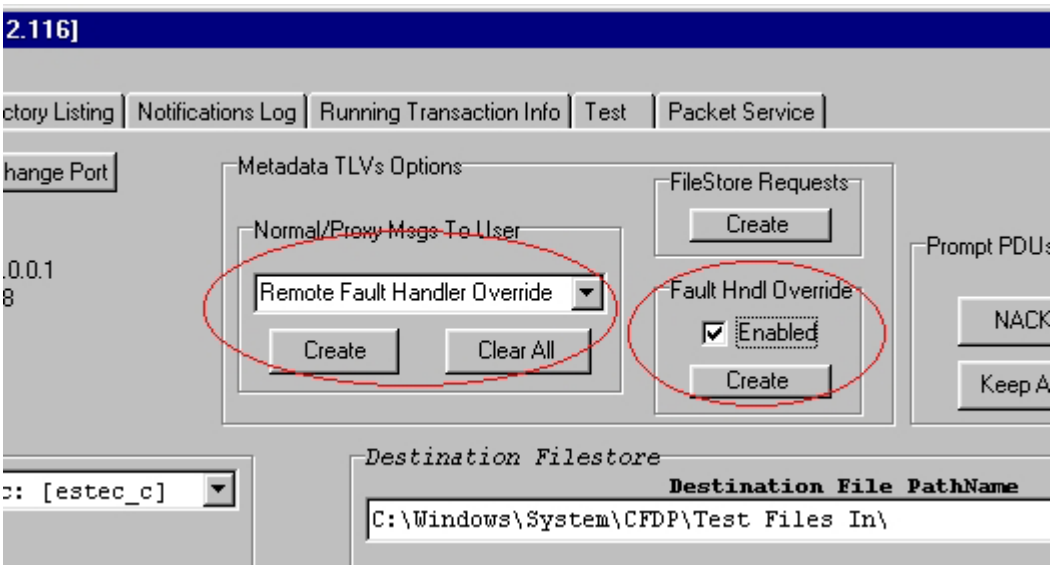


Figure 7-18: TLV and Remote Fault Handler Overrides Selection

The *Local Fault Handler Override Options* is not contained in the Metadata PDU but it is just 'local' to the CFDP entity related to the User Software.

This option is used to override the default Fault Handler loaded from the MIB file in case a protocol error occurs. This type of Fault Handler Override Option can be created from the *'ACK/NAK/Timing/MIB Settings'* tab window of the CFDP User Software (see figure 7-19).

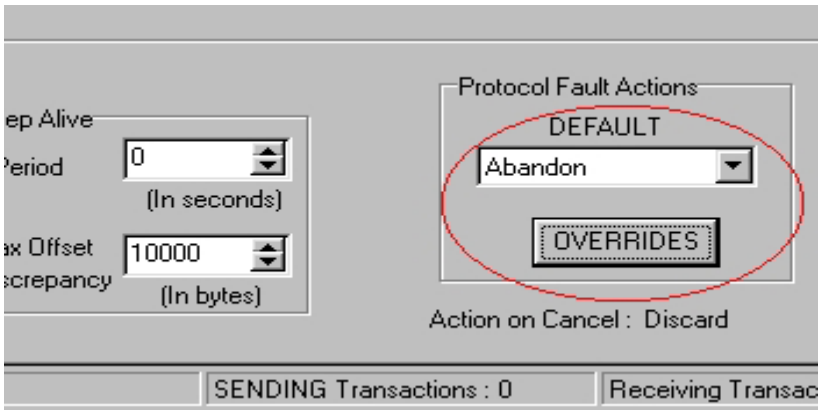


Figure 7-19: Local Fault Handlers Override Set-Up

In all of the cases discussed in this subsection, the creation of a Fault Handlers Override scheme is made possible by the use of a dedicated Table that is fully user-configurable (see figure 7-20).

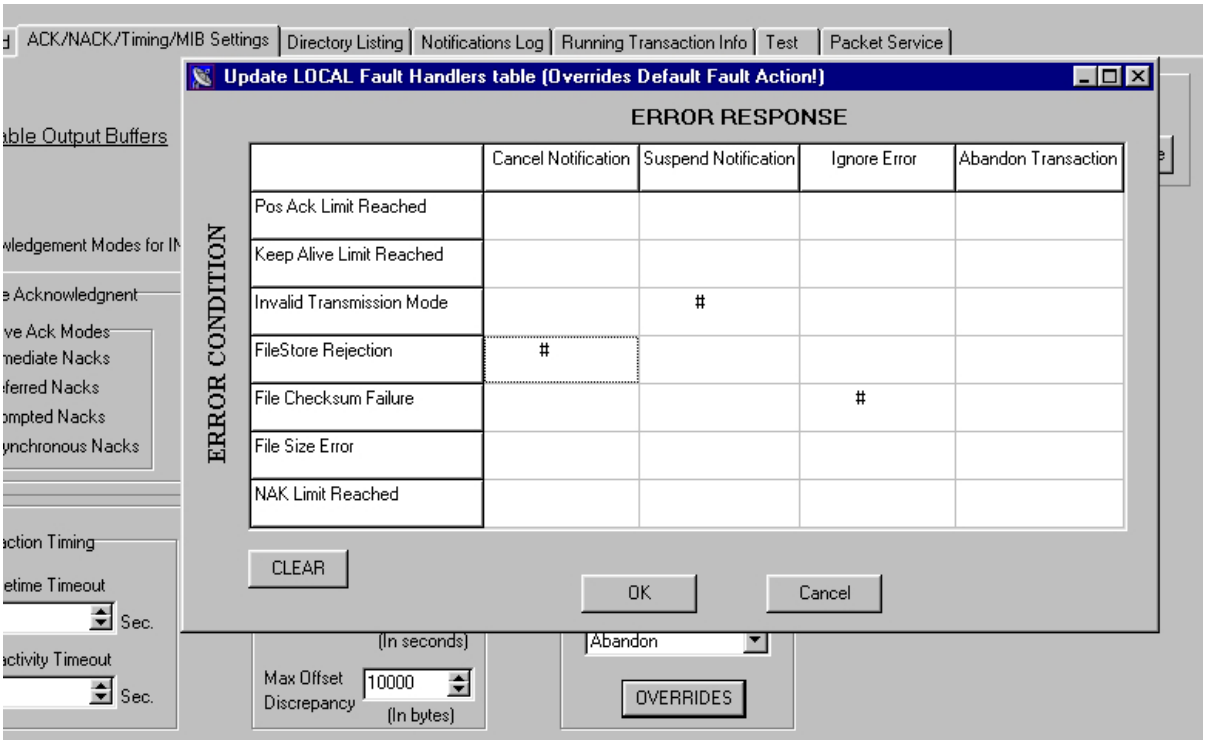


Figure 7-20: Local Fault Handlers Override Configuration Table

7.2.8.4 Extended Tab Window

The Extended Tab Window is used to set up the behavior of the local CFDP Entity related to the User Software, in case it has to act as a Waypoint performing a Store & Forward procedure during a file transfer between CFDP entities having a non-direct link. See figure 7-21.

From this window the user can:

a) Retrieve, if the local CFDP entity connected to the User Software is capable of Extended procedures;

b) Select the Forwarding Mode as:

1) *Immediate Incremental* during the file reception;

2) *Deferred* once the local CFDP entity acquired a complete custody of the file being transferred.;

3) *Prompted* by the user for the selected file delivery transaction.

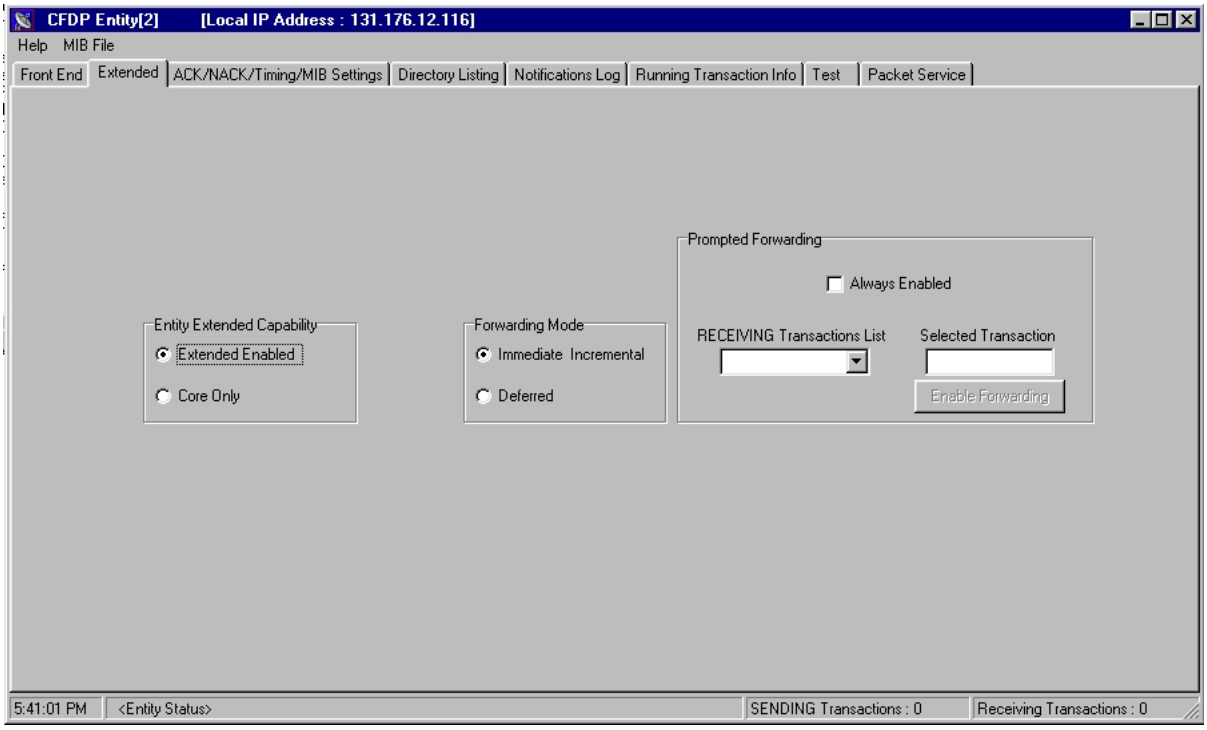


Figure 7-21: Extended Tab Window

7.2.8.5 ACK/NAK/Timing/MIB Settings Tab Window

The ACK/NAK/Timing/MIB Settings Tab Window (figure 7-22) contains all of the setting values parameters loaded from the CFDP MIB File during the Entity's initialization phase and related to:

a) *Positive* and *Negative Acknowledged* modes for Incoming/Outgoing FDU transmissions, as well as the local CFDP entity's acknowledgment capability;

b) *Keep Alive* parameters;

c) *Local* default and override *Protocol Fault Actions*;

d) *File Delivery* transaction *Timing* values.

Once loaded from the MIB, all of these parameters can be fully configured by the user for each sending or receiving FDU Transaction.

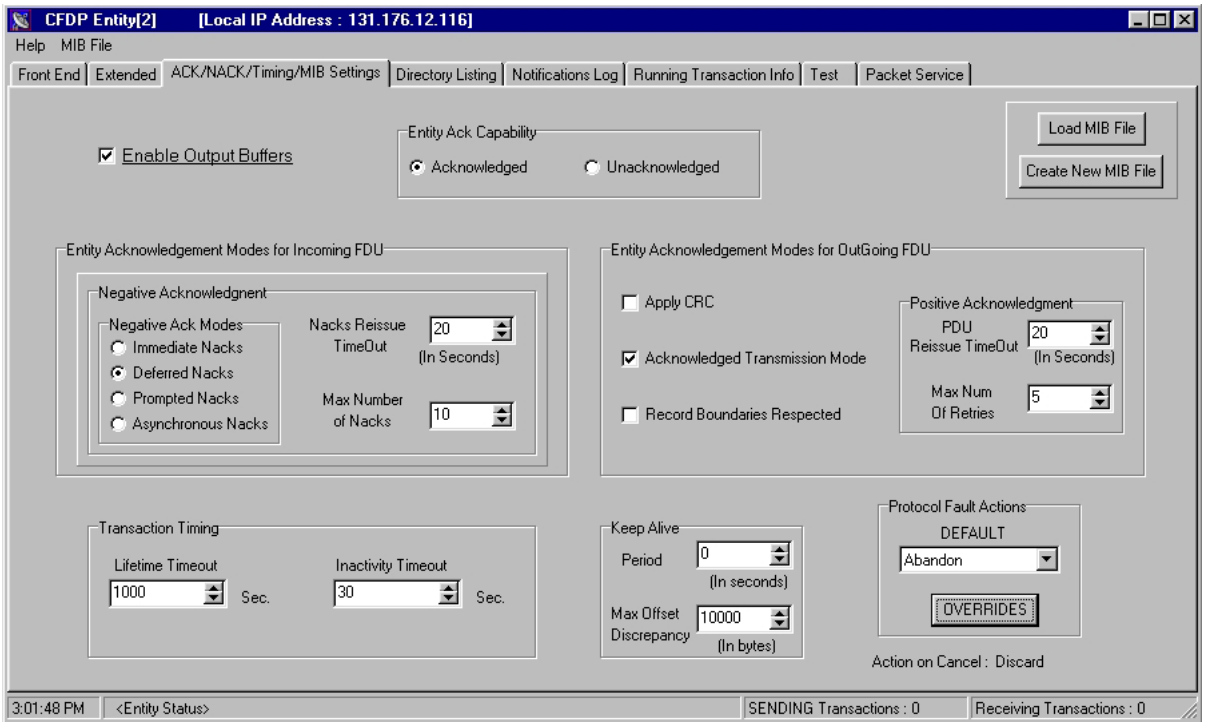


Figure 7-22: Acknowledgement Modes Tab Window

This window also allows the user to:

a) Load or create a new MIB file;

b) Enable/Disable the use of output buffers.

7.2.8.6 Directory Listing Tab Window

The Directory Listing Tab Window (figure 7-23) is used once a Directory-listing file has been received to the local filestore. The user is then able to easily load and display the files from this interface.

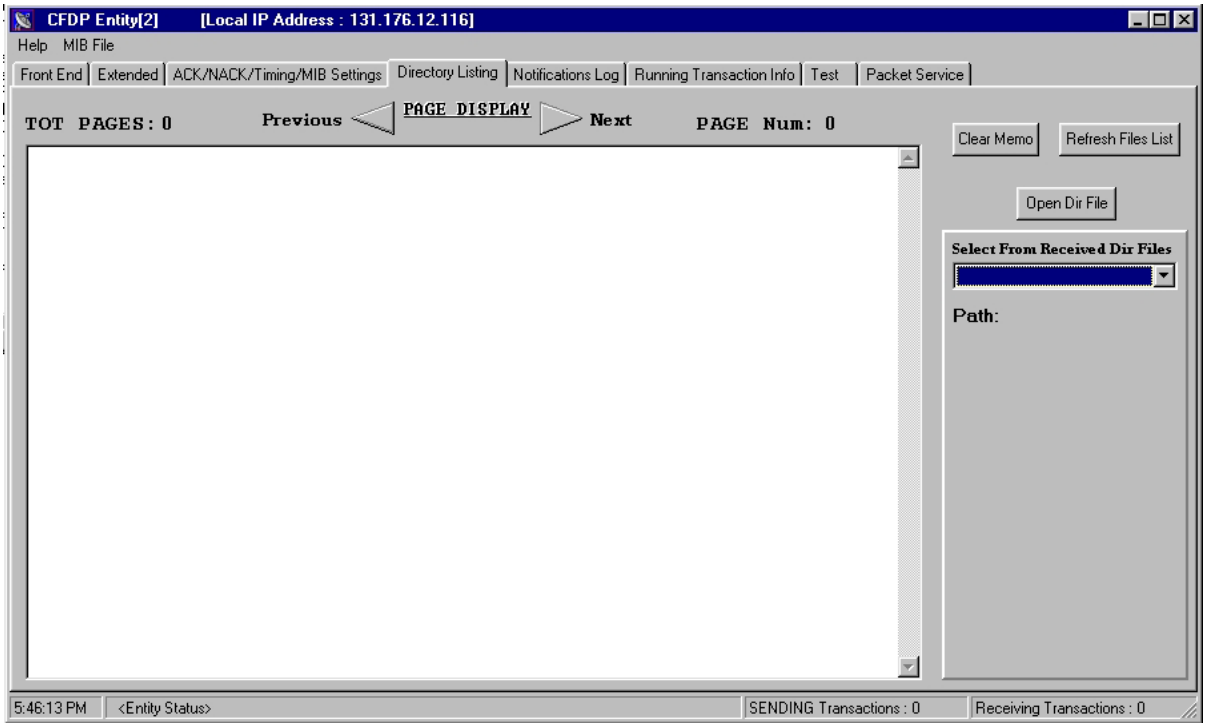


Figure 7-23: Directory Listing Tab Window

7.2.8.7 Notifications Log Tab Window

The Notifications Log Tab Window (figure 7-24) contains all of the notifications deriving from *Indication Primitives* delivered by the local CFDP entity during each FDU Transaction.

Additional information (like File Data Offset, Resulting Transfer Time, NAK PDUs details, etc.) are also shown in this window.

Note that the logs are cleaned every 350 lines. A procedure to dump all logs in a file before they are canceled is under implementation.

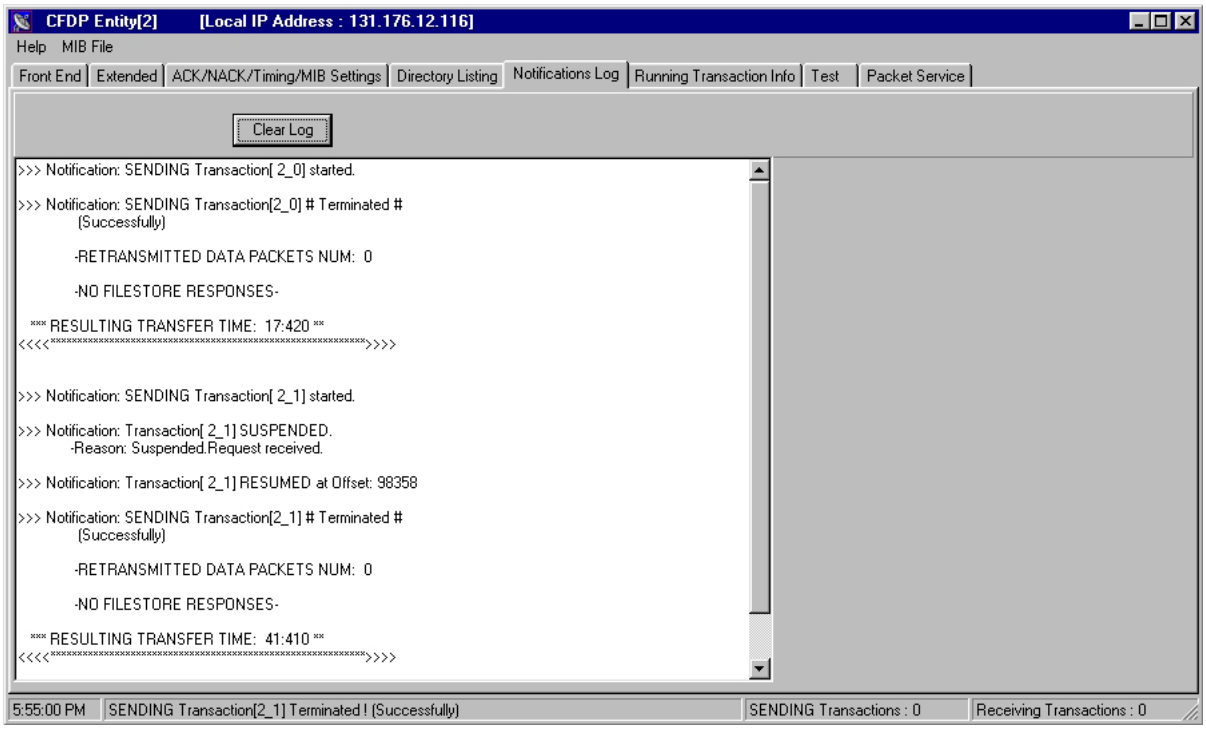


Figure 7-24: Notifications Log Tab Window

7.2.8.8 Running Transaction Info Tab Window

The Running Transaction Info Tab Window (figure 7-25) allows the user to select a Running FDU Transaction (both Sending and Receiving) and display all its parameters as well as *run-time* information on its status. See figure 7-26

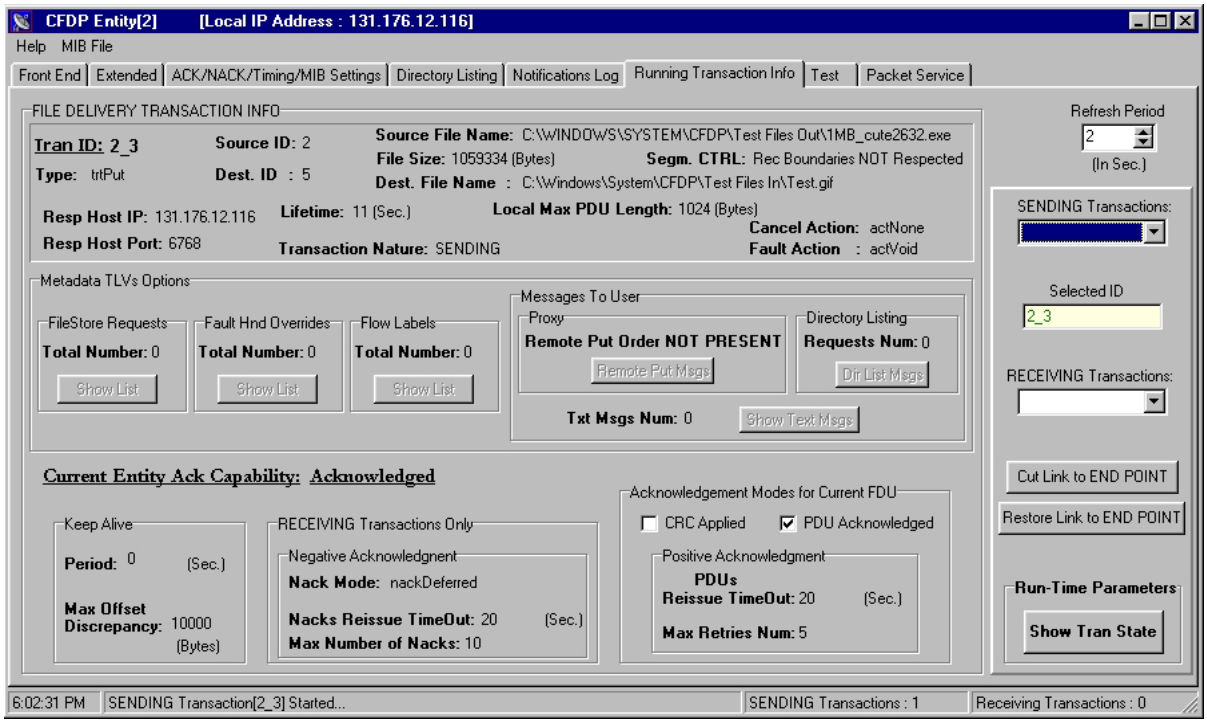


Figure 7-25: Running Transaction Info Tab Window

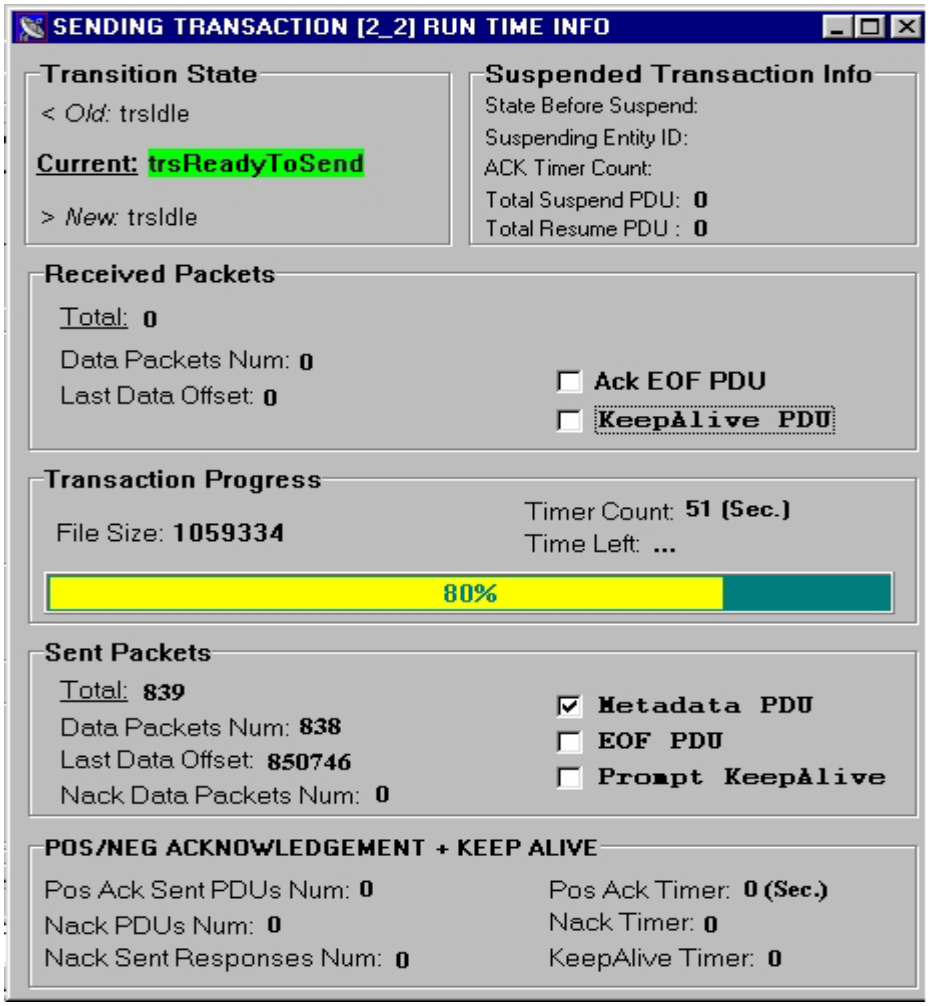


Figure 7-26: Run-time Info Window

7.2.8.9 Test Tab Window

The Test Tab Window (figure 7-27) can be used to run pre-configured Test Scenarios.

In this way, each Test Scenario can correspond to a given FDU Transaction that is believed to be representative for testing a certain CFDP behavior.

Once an FDU Transaction is loaded (both from File and from the User Interface Front End), it can be run by selecting a customized *CFDP Packet Size* and *Simultaneous TransactionNnumber*.

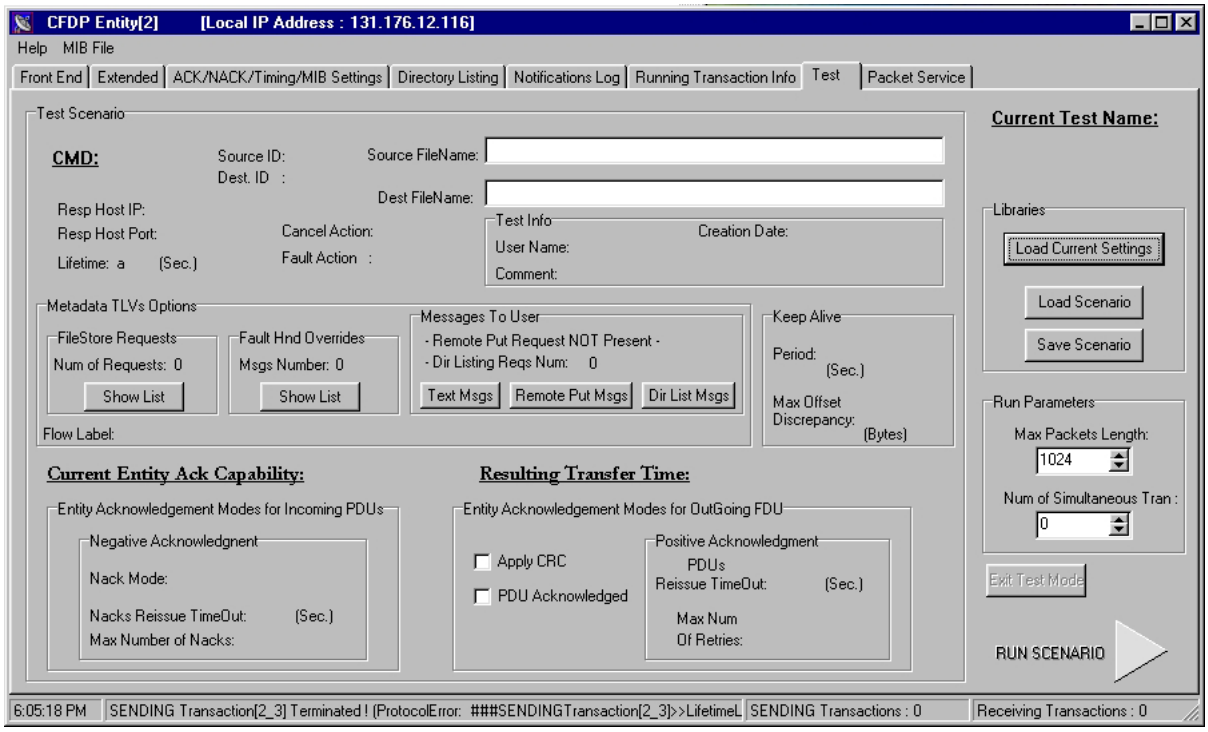


Figure 7-27: Test Tab Window

7.2.8.10 Packet Service Tab Window

The Packet Service Tab Window (figure 7-28) represents the user interface to the Packet Service component connected to the local CFDP entity.

This component is in charge of receiving CFDP packets and handing them over using the user-selected underlying communication protocol. On the other side, it also receives packets from the underlying communication protocol, extracts the CFDP PDUs, and provides them to the connected CFDP entity.

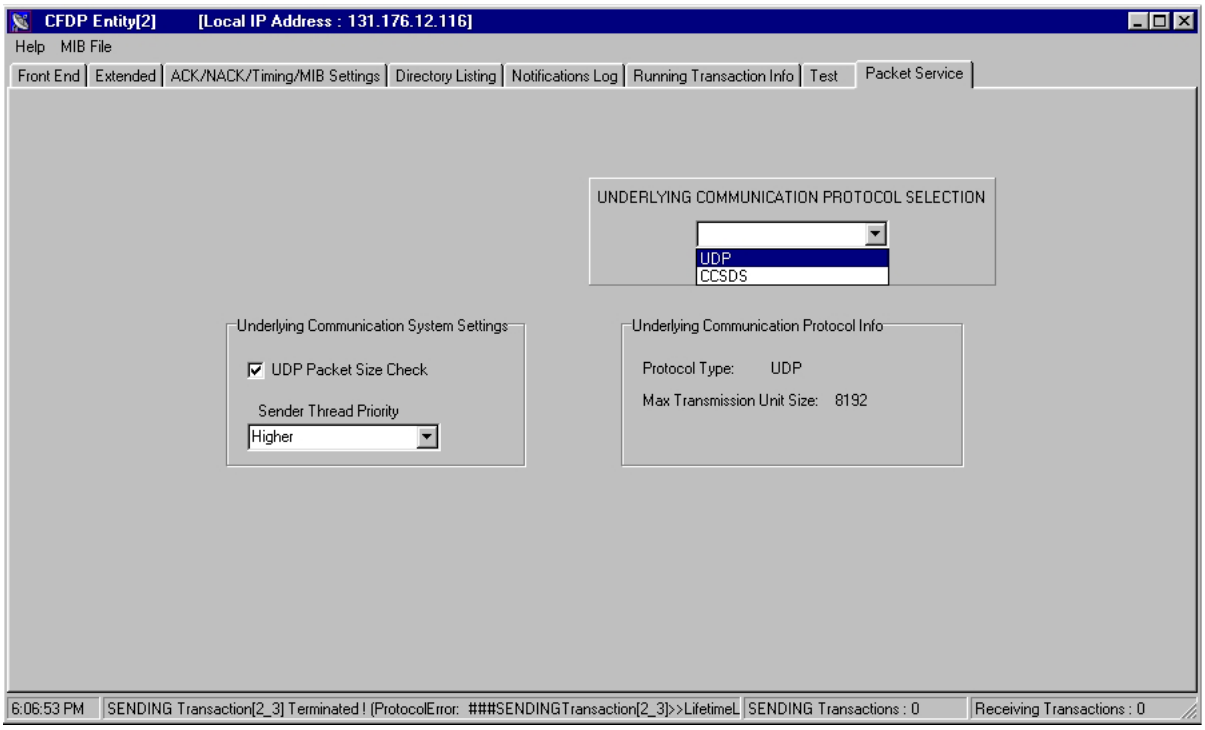


Figure 7-28: Packet Service Tab Window

7.2.9 CFDP OPERATIONAL TEST AND VALIDATION PLAN

7.2.9.1 Introduction

This subsection defines the ESTEC CFDP test and validation plan in a simulated space mission link. It also provides a detailed description of all test parameters to be taken into account during a CFDP operational test, as well as an explanation of how these parameters can affect an FDU transfer. This subsection also contains a description of the environment in which this test would take place.

In order to perform a full evaluation of the CFDP performance, most of the *operational modes* need to be represented. To achieve that, several *Test Sets* must be built, covering as much as possible the entire range of *Use Cases*. The first step to performing a complete CFDP performance measurement is to define a procedure for a *theoretical evaluation*. Therefore, once all the planned tests have been conducted, the resulting values defining the CFDP throughput (File Size/ Transmission Time) can be analyzed and compared to the expected ones.

The primary test phase focuses on the Core CFDP procedures verification using different *Test Configuration parameters* within different Link configurations.

7.2.9.2 Relay Testing Module (RTM)

As the CFDP deals with the space communication link and a real-world test is obviously not feasible unless a flying test is performed on a satellite, the need for a tailored space link simulation tool is born.

For this purpose, the *Relay Testing Module* (RTM) software has been designed and implemented.

This module is a UDP packet-based *space link simulator*, which provides a simulated space environment in which the CFDP can be run. It is especially suitable for testing the CFDP ability to overcome all the perturbations and the anomalies (due to a noisy space link) and complete the file transfer. It is understood that even the performance evaluation phase will benefit from such a tool.

The RTM can be enabled for listening on two different Local Ports for UDP packets and forward them to a third destination (Remote Host) within the CFDP Network, introducing several types of *perturbations* on the link.

The following are the types of *Link Perturbation* that can be injected on the link by RTM during a test phase. They can be classified by nature as follows:

a) Delay.

b) Diversity:

   1) Packet Duplication Rate;

   2) Packet Out-Of-Order Rate.

c) Error (Packets Losses):

   1) Bit Error Rate (BER);

   2) Burst Error Occurrence Probability (BEOP) or Burst Intensity.

d) Typed Packets Dropping (Metadata, EOF, Data, ACK, etc.)

Furthermore, the previously mentioned *Test Window* can be used in order to create, save and load different *Test Scenarios*. Once a test scenario is present, multiple FDU Transactions of a certain *User-Defined* type can run *simultaneously* together with different selected CFDP packet sizes.

The RTM is a stand-alone software module able to receive all the PDUs concerning an FDU Transaction and *Forward*, *Delay* or *Drop* them toward the final destination according to the *Error Injection Algorithm* represented by the given user-settings. A brief explanation on the possible Link Perturbations injected by this module will follow, but for a complete coverage of the RTM please refer to the Users Manual available on the CFDP Web site: http://cfdp.jpl.nasa.gov. Figure 7-29 shows the RTM GUI.

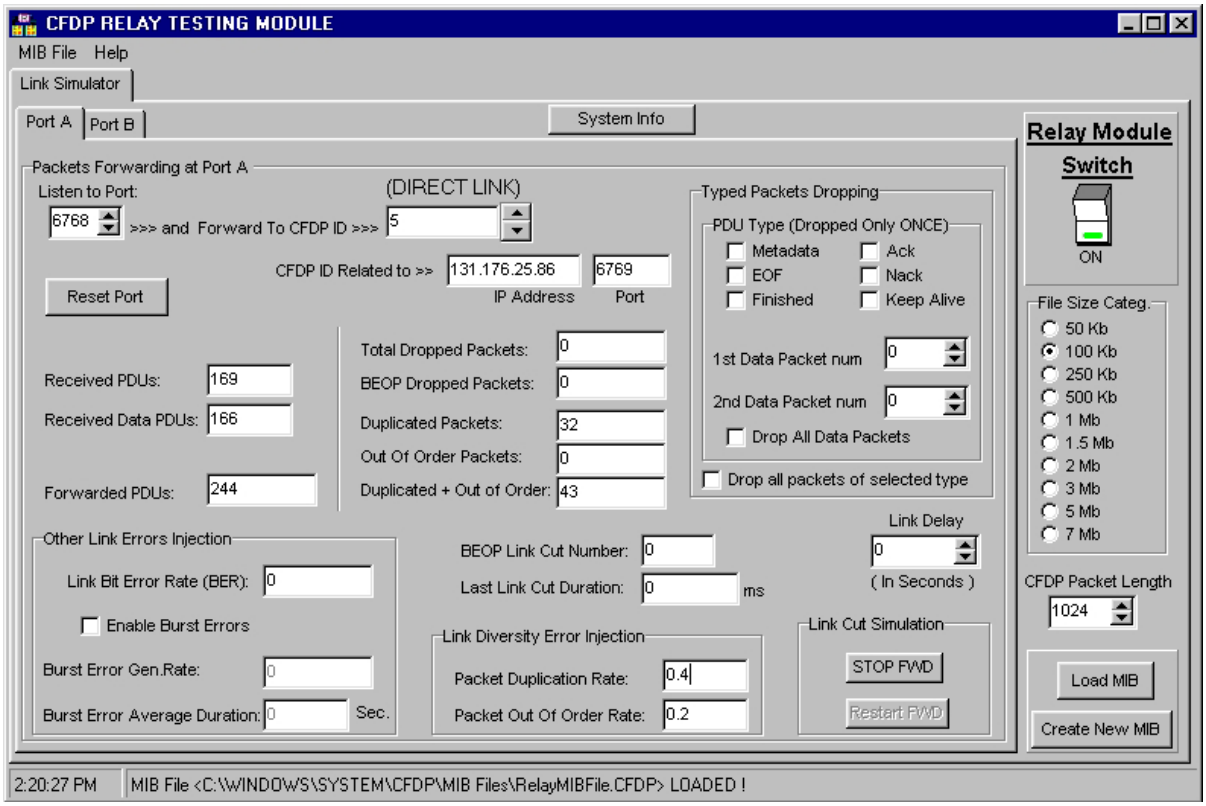


Figure 7-29: Relay Testing Module GUI

7.2.9.3 Throughput Theoretical Evaluation

The performance of the CFDP can be described by the *throughput* of the data transfer for a given constant bit-rate communication link. Unfortunately, the dynamic routing capability and the suspend/resume scheme make it impossible, or at least very difficult, to formulate the overall link throughput. In addition, no other protocols can be compared with CFDP in such an environment, as they do not support dynamic routing or suspend/resume capability. In the bid of the throughput comparison with the other protocols, the direct line-of-sight CFDP throughput (no intermediate waypoints) is evaluated.

The throughput of the CFDP could be evaluated by three different cases:

a) No Acknowledgement: Simplex link;

b) Immediate Negative Acknowledgment;

c) Deferred Negative Acknowledgment.

In the simplex link case, the CFDP does not assure the integrity of the file at the other end, and the transfer time would not vary according to the different configuration and the environmental conditions.

In this subsection, two other retransmission strategies are considered to evaluate the theoretical throughput.

For the Immediate NAK case, the total number of packets N_{total} which are transmitted by the file sender is given as:

$$\begin{aligned} \text{Ntotal} &= \text{N} + \text{N} \times \text{PER} + \text{N} \times \text{PER2} + \text{N} \times \text{PER3} + \ldots \\ &= N / (1 - PER) \end{aligned} \tag{1}$$

where

N : Total number of packets = File Size / Packet Length

PER : Packet Error Rate induced from the link bit error rate (BER) = $1 - (1 - \text{BER})^{8L}$

L : Packet Length (in bytes)

N_{total} takes into account the repeating retransmissions of the packets.

To send this number of packets to the file receiver, the maximum value of the total transmission time $T_{Tx,max}$ of a file is calculated as:

$$T_{Tx,max} = N_{total} \times T_{packet} + RTT = \frac{N \times T_{packet}}{(1 - PER)} + RTT \tag{2}$$

where

$T_{Tx,max}$: Maximum value of the total transmission time

T_{Tx} : Total transmission time for a given file

T_{packet} : Packet time for a packet to arrive at the receiver = Packet Length / Transmission Bit Rate

Round Trip Time (RTT) between file file sender and the file receiver (sec)

The term *RTT* is inserted in Equation *(2)*, because if the last packet of the file transfer is lost, *RTT* seconds will be taken for the receiver to send a NAK packet and receive back the corresponding packet, which is the last one of the whole transfer. In this equation, the processing time of both file sender and receiver's CPU is ignored.

Consequently, the total retransmission time of a file transfer T_{Tx} would be smaller than $T_{Tx,max}$ but greater than ($N_{total} \times T_{packet}$).

On the other hand, for the Deferred NAK case, the file receiver waits for the end of file transfer from the file sender before it issues the set of NAKs for all the lost packets during the transfer. In this case, the total transmission time T_{Tx} becomes

$$T_{\mathrm{Tx}} = (N \times T_{\mathrm{packet}}) + (RTT + N \times T_{\mathrm{packet}} \times PER) + (RTT + N \times T_{\mathrm{packet}} \times PER^2) + (RTT + \ldots \quad (3)$$

In Equation *(3)*, the first term in () represents the first time transfer of the given file data, and if there are errors, an RTT will be passed before starting to receive another set of the missing packets. Recursively calculated, the resulting equation becomes

$$T_{\mathrm{Tx}} = \frac{N \times T_{\mathrm{packet}}}{(1 - PER)} + k\, RTT \quad (4)$$

where

k : number of retransmissions. determined from N and PER

The number of retransmission k varies according to N and PER. Repeating the transmissions for the erroneous packets will decrease the number of packets by PER, so k^{th} retransmission consists of $(N \times PER^k)$ packets. If this value $(N \times PER^m)$ is smaller than 1, then there will be no more packets to send. From N and PER, k is obtained:

$$k = -\frac{\log N}{\log PER} \quad (5)$$

For example, to send 10000 packets with PER equal to 0.01, the retransmission time k is equal to 2. Of course, it is a theoretical estimation.

In summary, the total transmission time of the CFDP for the two different retransmission strategies can be described as:

c) Immediate NAK case: $$T_{\mathrm{Tx}} = \frac{N \times T_{\mathrm{packet}}}{(1 - PER)} + RTT + T_{\mathrm{o}}$$

d) Deferred NAK case: $$T_{\mathrm{Tx}} = \frac{N \times T_{\mathrm{packet}}}{(1 - PER)} - \frac{\log N}{\log PER} RTT + T_{\mathrm{o}} \quad (5)$$

Where T_{o} is the overall link overhead, which consists of the initial connection establishment time and the final closing time. In CFDP, Put request PDU does not need to wait for the response from the file receiver, and the file sender starts immediately to send the file data. On the other hand, for the Get request PDU, which is issued by the file receiver, the file receiver need to wait single link time (half the RTT) before the file sender starts to send data packets. At the end of file transfer, both file sender and file receiver can issue the [Finished] PDU, and in a normal case, it is the file receiver who issues this [Finished] PDU. In this case, to close the connection ([Finished] PDU and then [ACK (Finished)] PDU), an RTT is needed for both ends.

The theoretical throughput of the CFDP is obtained in both cases:

Throughput of CFDP = File Size / T_{Tx} *(6)*

7.2.9.4 CFDP Link Configurations

The *CFDP Service Classes* refer to the following types of Link Configurations (figure 7-30):

a) Direct-Link (point-to-point) transactions (Class 1, Class 2);

b) One transparent Waypoint (*Relay Testing Module*) for test purpose only (via '*relays*' as opposed to CFDP '*waypoints*');

c) Link via CFDP Waypoint (Class 3, Class 4);

d) Proxy transactions either as Get or Remote Put (Class 5).

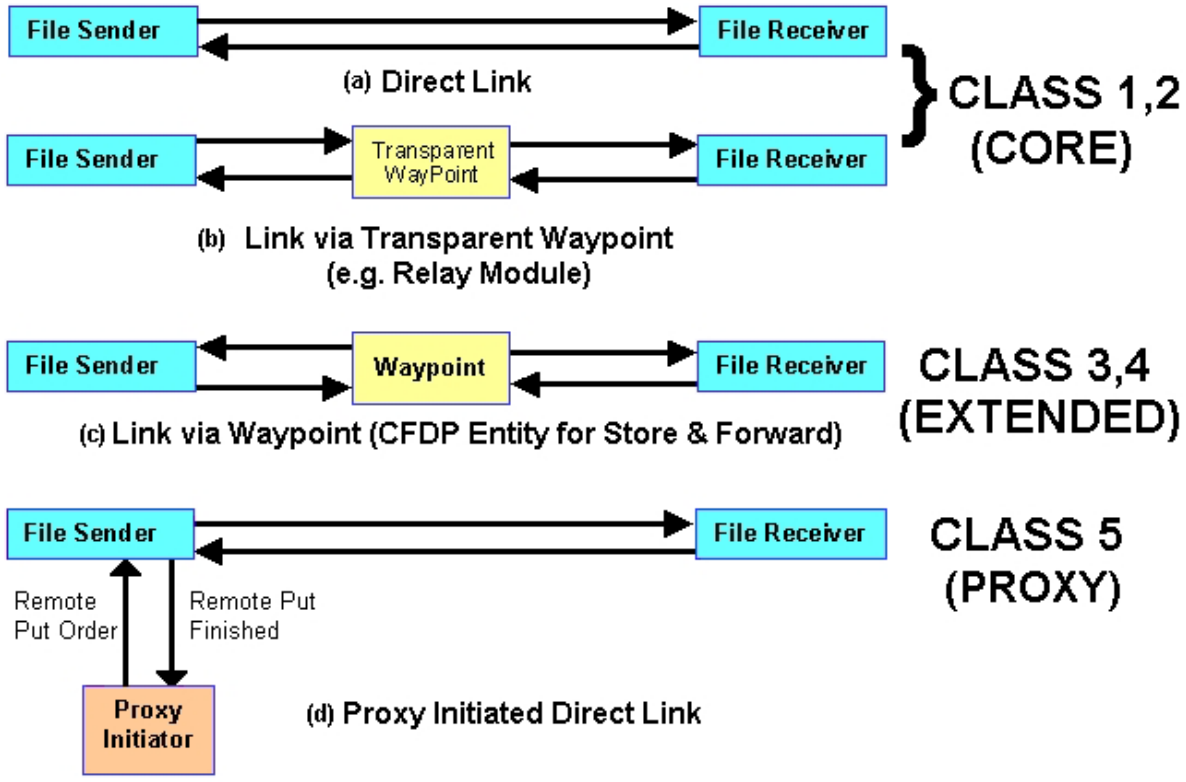


Figure 7-30: Test Link Configurations Diagram

The ESTEC *Relay Testing Module* can be added to any of the above links in order to test the related CFDP entity's behavior over a simulated noisy space link.

7.2.9.5 Test Parameters

7.2.9.5.1 General

A test parameter is a parameter that can affect the resulting performances of an FDU. Test parameters can be of two types:

a) CFDP related.

b) Link Simulation Related (*Relay Testing Module*).

In both cases, certain types of parameters can vary their value from test to test.

7.2.9.5.2 CFDP Parameters

a) FIXED:

1) Positive Ack Reissue Time Out (on sender side);

2) Positive Ack Max Reissue Number (on sender side);

3) Negative Ack Reissue Time Out (on receiver side);

4) Negative Ack Max Reissue Number (on receiver side);

5) Keep Alive Period;

6) Keep Alive Max Discrepancy offset (Bytes);

7) Forward Link bit rate (Kbps).

b) VARIABLES:

1) File Size;

2) Packet Length.

c) Retransmission Strategy Modes:

1) No Acknowledgments (Unreliable File Transfer);

2) Immediate NAKs (negative Acknowledgments);

3) Deferred NAKs.

7.2.9.5.3 Simulated Space Link Parameters

a) Delay.

b) Diversity:

1) Packet Duplication Rate;

2) Packet Out-Of-Order Rate.

c) Error (Packets Losses):

1) BER;

2) BEOP or Burst Intensity.

d) Number of simultaneous Transactions.

Refer to table 7-9 for the CFDP test parameters values range.

Table 7-9: CFDP Test Parameters Value Range

Variable Parameters	Unit	Values Range (default values in bold)					
File Size	Kbytes	20	100	1000	10000		
Packet Size	bytes	128	256	512	1024		
Link Delay	msec	0	100	200	400		
Bit Error Rate (BER)		0	1e-5	1e-4	1e-3		
Packet duplication rate		0	1e-2				
Packet out-of-order rate		0	1e-2				
Burst Generation rate		0	0.03				
Burst Error Average duration	Sec	0	0.1				
Simultaneous Transactions		0	5	10			
Fixed Parameters	Unit	Value					
Transmission bit rate	Kbps	100					
Pos Ack Reissue Time Out	Sec	30					
Pos Ack Max Reissue number		5					
Neg Ack Reissue Time Out	Sec	20					
Neg Ack Max Reissue number		10					
Keep Alive Period	Sec	0					
Keep Alive Max Discrepancy	bytes	0					
Transaction Lifetime	Sec	1000					

7.2.9.6 Results Parameters

Resulting Transfer Time (and related CFDP Throughput).

7.2.9.7 Test Sets Definition

7.2.9.7.1 General

To obtain the most representative Test results, different categories of 'Testing Sets' have been defined assuming the previous mentioned 'Test Configuration Parameters'.

For each Set of Tests there is a *variable* Link Simulation parameter ranging between different pre-defined values, while the remaining Test Configuration Parameters are left *fixed.*

Furthermore, each Set will be performed in any of the three CFDP acknowledgment modes.

For the entire test duration, fixed *Positive Acknowledgement, Negative Acknowledgement* and *Keep Alive* setting values will be assumed (according to table 7-9).

7.2.9.7.2 Size Test Sets

7.2.9.7.2.1 Size Set 1

a) *Fixed* parameters:

1) File Size (1 MB);
2) No Link Delay;
3) No Link Diversity;
4) No Link Errors;
5) No Simultaneous Transactions.

b) *Variable* parameter: Packet Size.

7.2.9.7.2.2 Size Set 2

a) *Fixed* parameters:

1) Packet Size (1024 bytes);
2) No Link Delay;
3) No Link Diversity;
4) No Link Errors;
5) No Simultaneous Transactions;

b) *Variable* parameter: File Size.

7.2.9.7.3 Delay Test Sets

7.2.9.7.3.1 Delay Set 1

a) *Fixed* parameters:

1) File Size (1 MB);
2) Packet Size (1024 bytes);
3) No Link Diversity;

4) No Link Errors;

5) No Simultaneous Transactions;

b) *Variable* parameter: Link Delay.

7.2.9.7.3.2 Delay Set 2

a) *Fixed* parameters:

1) File Size (1 MB).

2) Packet Size (1024 bytes).

3) No Link Diversity.

4) BER = 1e-3.

5) No Burst Errors.

6) No Simultaneous Transactions.

b) *Variable* parameter: Link Delay.

7.2.9.7.4 Link Error Test Sets

7.2.9.7.4.1 Link Error Set 1

a) *Fixed* parameters:

1) File Size (1 MB);

2) Packet Size (1024 bytes);

3) Link Delay = 100 ms;

4) No Link Diversity;

5) No Burst Errors;

6) No Simultaneous Transactions;

b) *Variable* parameter: BER.

7.2.9.7.4.2 Link Error Set 2

a) *Fixed* parameters:

1) File Size (1 MB);

2) Packet Size (1024 bytes);

3) Link Delay = 100 ms;

4) Packet Duplication Rate = 1e-?;

5) Packet Out-Of-Order Rate = 1e-?;

6) BER = 1e-3.

7) No Simultaneous Transactions.

b) *Variable* parameter: BEOP:

1) Burst Generation Rate = 0.03;

2) Burst Average Duration = 0.1 sec.

7.2.9.7.5 Link Diversity Test Sets

7.2.9.7.5.1 Link Diversity Set 1

a) *Fixed* parameters:

1) File Size (1 MB);

2) Packet Size (1024 bytes);

3) Link Delay = 100 ms;

4) BER = 1e-3;

5) No Burst Errors;

6) No Packet Out-Of-Order;

7) No Simultaneous Transactions;

b) *Variable* parameter: Link Diversity. Packet Duplication Rate = 1e-?.

7.2.9.7.5.2 Link Diversity Set 2

a) *Fixed* parameters:

1) File Size (1 MB);

2) Packet Size (1024 bytes);

3) No Link Delay;

4) No Link Errors;

5) No Packet Duplication;

6) No Simultaneous Transactions.

b) *Variable* parameter: TBS. Packet Out-Of-Order Rate = 1e-?.

7.2.9.8 Test Results

Test 1. Different Packet Size					
Test Parameters			Retransmission Strategy		
Type	Name	Unit	No ACKs	Imm. NAKs	Deferred NAKs
File	File size	Bytes	1,001,078	1,001,078	1,001,078
Constants					
CFDP parameters	Pos Ack Timeout	Sec	0	30	30
	Pos Ack Max Retries		0	10	10
	Negative Ack Timeout	Sec	0	20	20
	Negative Ack Max Retries		0	10	10
	KeepAlive Timeout	Sec	0	0	0
	Forward Link bit rate	Kbps	100	100	100
LinkSim parameters	Link delay	ms	0	0	0
	Bit Error Rate		0	0	0
	Packet duplication rate		0	0	0
	Packet out-of-order rate		0	0	0
	Burst mean duration	Sec	0	0	0
	Burst mean arrival time	Sec	0	0	0
Variables	Packet Length (Bytes)		Resulting Transfer Time (Sec)		
	128				
	256				
	512				
	1024				

7.2.10 MIB FILE

Currently, the MIB file used for the ESTEC CFDP implementation is a text file divided into three sections:

a) Addresses Table;

b) Default Setting Values;

c) Entity Capabilities.

Each section can contain an undefined number of Items**,** contained in table 7-10.

Table 7-10: MIB Items

SECTION	ITEM	COMMENTS
Addresses Table		
"	Cfdp**N**	CFDP Entity addresses mapping. Used to retrieve the physical network address (IP + Port) of a certain CFDP Entity. This line is not considered if the Cfdpnh**N** is present. **N** = CFDP Entity identifier
"	Cfdpnh**N**	Identifies the Next Hop of a certain CFDP Entity. If this line is present, it means that the local entity has no direct connectivity to the final destination entity. Then an INTERMEDIATE CFDP Entity shall be used as a WAYPOINT. N = CFDP Entity identifier
Default Settings Values		
"	Transaction Lifetime	In seconds
"	Max File Length	In Kbytes
"	Max File Segment Length	In Bytes
"	Put File Type	Bounded/Unbounded
"	Put Data Type	Octets/Packets
"	Put Report Mode	Prompted/Periodic
"	Transaction Finished Indication	Active/Not active (Sender – NAK only)
"	Transmission Mode	Acknowledged/Unacknowledged (Transmission mode for outgoing FDU)
"	NAK Mode	Immediate/Deferred Prompted/Asynchronous

SECTION	ITEM	COMMENTS
“	NAK Timeout	In seconds
“	NAK Max Number	Error threshold – number of retries
“	PDUs Reissue Timeout	In seconds
“	PDUs Max Reissue Number	Error threshold – number of retries
“	Keep Alive Period	In seconds
“	Max Offset Discrepancy	In Kbytes
“	Fault Action	Abandon/Cancel/Suspend/Ignore
“	Cancel Put Action	Discard/Retain
Entity Capabilities		
“	RCV Transmission Mode	Acknowledged/Unacknowledged (Transmission mode for incoming FDU) If Unacknowledged then none of the Acknowledgements settings are considered

Example:

```
#    -***** This is the MIB file used by CFDP Version for ESTEC *****-
# This file contains the mappings of Host names to IP addresses and of
# CFDP Entity IDs to IP addresses.
# Each entry should be kept on an individual line.
# The Host name or the CFDP Entity ID should be kept on an individual
# line and placed in the first column followed by the corresponding IP
# address.
# When a line is to identify the CFDP Entity address mapping, it starts #
with the sequence "cfdp" followed by the CFDP Entity ID (No space
# between).
# The sequence "cfdpnh" identifies the Next Hop of a certain CFDP Entity
# If Such line is present, it means that the local entity has no direct #
connectivity to the final Destination entity.
# Then an INTERMEDIATE CFDP Entity shall be used as a WAYPOINT.
# The character ":" should separate the first and the second columns
# and at least one space. No space chars are allowed in the first
# column up to the ":" .
# This File is subdivided in SECTIONS. Each section contains different
# types of information. The beginning of a
# section is delimited by the characters sequence "$###$".
# The Section names are located after the Section Delimiter up to the
# end of the line.
# Only the FIRST section is delimited by the sequence "$$#####$$" and it #
contains the CFDP address mapping.
# The EOF is marked as "$$$$$".
# Additionally, comments (such as these) must be denoted by a "#"
# symbol and may be inserted on individual lines or following the
# second column.
#
# For example:
#    rhino.acme.com     102.54.94.97    # source server
#    x.acme.com         38.25.63.10     # x client host
#    cfdp13             171.25.32.11    # Lander
```

```
$$#####$$ Addresses Table
localhost:       127.0.0.1/6768
cfdp0:           127.0.0.1/6768 #Local Network
cfdp1:           131.176.12.116/6766 #Max PC
cfdpnh1:         131.176.12.116/6766 #Max PC Next Hop
cfdp2:           131.176.12.116/6769 #Max PC
cfdpnh2:         131.176.12.116/6770 #Max PC Next Hop
cfdp3:           131.176.12.141/6769 #Lab PC
cfdprelay:       127.0.0.1/6768 #CFDP Relay Testing Module
$####$ Default Settings Values
Transaction Lifetime: 150
Max File Length: 5000 #In Kbytes
Max File Segment Length: 1024 #In Bytes
Put File Type: bounded
Put Data Type: octets
Put Report Mode: Prompted
Transaction Finished Indication: active #Sender - NAK only
Transmission Mode: acknowledged #Transmission mode for outgoing FDU
NAK Mode: deferred
NAK Timeout: 30
NAK Max Number: 10
PDUs Reissue Timeout: 20 #In Seconds
PDUs Max Reissue Number: 4
Keep Alive Period: 100 #In seconds
Max Offset Discrepancy: 10000 #10 Kbytes
Fault Action: abandon
Cancel Put Action: discard
$####$ Entity Capabilities
RCV Transmission Mode: acknowledged/unacknowledged
$$$$$
```

8 IMPLEMENTATION CAPABILITIES SURVEY

8.1 DERA/BNSC

CFDP Implementation Survey

Agency	Name	Submitted by
DERA	BNSC	R Smith

General Implementation Information

Platform	OS	Language
Force Sparc 3CE	VxWorks 5.3.1	C

Max. File Size	Max. Segment Size	Mechanism Used for Persistent Storage	Other Persistent Storage Options
FFFFFFFF	512 (UDP packet size limitation)	RAM-based DOS FS	None on development system

Underlying Communications Systems

CCSDS AOS VCDU	CCSDS TM_TC	CCSDS Prox_1	SCPS_NP	UDP_IP	Other
				X	

1. CFDP Procedures

CRC Proced.	Put Proced.	Transaction Start Proced.	PDU Forwarding Proced.	Copy File Proced.	Positive Ack. Proced.	Faults Proced.	Filestore Proced.
X	X	X	X	X	X		

2. CFDP Protocol Classes

Unreliable Transfer	Reliable Transfer	Reliable Transfer by Proxy
X	X	

3. CFDP Protocol Options

Options

End Type

Sender	Receiver
X	X

Put Modes

UnACK	NAK
X	X

Put NAK Modes

Immediate	Deferred	Prompted	Asynchronous
	X		

Put File Types

Bounded	Unbounded
X	X

Segmentation Control (Record Boundaries Respected)

Yes	No
	X

Put Primitives (Receiving End)

File_segment_receive.ind
X

Put Error Responses (Sending End)

Ignore	Abandon	Cancel	Suspend
	X		

Put Error Responses (Receiving End)

Ignore	Abandon	Cancel	Suspend
	X		

Put Actions

Cancel_PutAction_	Suspend_PutAction_
X	

Cancel Put Action (Receiving End)

Discard data	Forward incomplete
X	

Put Report Modes (Sending End)

Prompted_Rpt_	Periodic

File Store Options

Create File	Delete File	Rename File	Append File	Replace File	Create Dir	Remove Dir

Directory Operations

Directory Listing Request	Directory Listing Response

Release of Retransmission Buffers

Incremental and Immediate	In total When "Finished" Received
	X

4. Timers and Counters

Timers

NAK Retry Timer	ACK Retry Timer	Prompt _NAK_ Timer	Async NAK Timer	Keep Alive Timer	Prompt _Keep Alive_ Timer	Inactivity Timer
X	X					X

Counters

NAK Retry Counter	ACK Retry Counter
X	X

8.2 EUROPEAN SPACE AGENCY (ESA)/EUROPEAN SPACE RESEARCH AND TECHNOLOGY CENTRE (ESTEC)

CFDP Implementation Survey

Agency	Name	Submitted by
European Space Agency(ESA)	ESTEC	Eric Bornschlegl and Max Ciccone

General Implementation Information

Platform	OS	Language
PC	Windows NT/95	Object Pascal on Delphi

Max. File Size	Max. Segment Size	Mechanism Used for Persistent Storage	Other Persistent Storage Options
up to 2 Mbytes	1024 bytes	DOS System	

Underlying Communications Systems

CCSDS AOS VCDU	CCSDS TM_TC	CCSDS Prox_1	SCPS_NP	UDP_IP	Other
				X	

1. CFDP Procedures

CRC Proced.	Put Proced.	Transaction Start Proced.	PDU Forwarding Proced.	Copy File Proced.	Positive Ack. Proced.	Faults Proced.	Filestore Proced.
X	X	X	X	X	X	X	X

2. CFDP Protocol Classes

Unreliable Transfer	Reliable Transfer	Reliable Transfer by Proxy
X	X	X

3. CFDP Protocol Options

Options

End Type

Sender	Receiver
X	X

Put Modes

UnACK	NAK
X	X

Put NAK Modes

Immediate	Deferred	Prompted	Asynchronous
X	X	X	X

Put File Types

Bounded	Unbounded
X	

Segmentation Control (Record Boundaries Respected)

Yes	No
	X

Put Primitives (Receiving End)

File_segment_receive.ind
X

Put Error Responses (Sending End)

Ignore	Abandon	Cancel	Suspend
X	X	X	X

Put Error Responses (Receiving End)

Ignore	Abandon	Cancel	Suspend
X	X	X	X

Put Actions

Cancel_PutAction_	Suspend_PutAction_
X	X

Cancel Put Action (Receiving End)

Discard data	Forward incomplete
X	

Put Report Modes (Sending End)

Prompted_Rpt_	Periodic
X	X

File Store Options

Create File	Delete File	Rename File	Append File	Replace File	Create Dir	Remove Dir
X	X	X	X	X	X	X

Directory Operations

Directory Listing Request	Directory Listing Response

Release of Retransmission Buffers

Incremental and Immediate	In total When "Finished" Received
	X

4. Timers and Counters

Timers

NAK Retry Timer	ACK Retry Timer	Prompt _NAK_ Timer	Async NAK Timer	Keep Alive Timer	Prompt _Keep Alive_ Timer	Inactivity Timer
X	X			X		

Counters

NAK Retry Counter	ACK Retry Counter
X	X

8.3 NASA/JPL

CFDP Implementation Survey

Agency	Name	Submitted by
NASA	JPL	Alan Schlutsmeyer

General Implementation Information

Platform	OS	Language
PPC single-board computers	VxWorks	C
Sun Sparc/UltraSparc	Solaris	C

Max. File Size	Max. Segment Size	Mechanism Used for Persistent Storage	Other Persistent Storage Options
no fixed limit tested up to 40K	512	SDR persistent object system	DRAM, NFS, local file system

Underlying Communications Systems

CCSDS AOS VCDU	CCSDS TM_TC	CCSDS Prox_1	SCPS_NP	UDP_IP	Other
X	X	X		X	

1. CFDP Procedures

CRC Proced.	Put Proced.	Transaction Start Proced.	PDU Forwarding Proced.	Copy File Proced.	Positive Ack. Proced.	Faults Proced.	Filestore Proced.
	X	X	X	X	X	X	X

2. CFDP Protocol Classes

Unreliable Transfer	Reliable Transfer	Reliable Transfer by Proxy
X	X	

3. CFDP Protocol Options

Options

End Type

Sender	Receiver
X	X

Put Modes

UnACK	NAK
X	X

Put NAK Modes

Immediate	Deferred	Prompted	Asynchronous
X	X		

Put File Types

Bounded	Unbounded
X	

Segmentation Control (Record Boundaries Respected)

Yes	No
X	

Put Primitives (Receiving End)

File_segment_receive.ind
X

Put Error Responses (Sending End)

Ignore	Abandon	Cancel	Suspend
		X	

Put Error Responses (Receiving End)

Ignore	Abandon	Cancel	Suspend
		X	

Put Actions

Cancel_PutAction_	Suspend_PutAction_

Cancel Put Action (Receiving End)

Discard data	Forward incomplete
X	

Put Report Modes (Sending End)

Prompted_Rpt_	Periodic

File Store Options

Create File	Delete File	Rename File	Append File	Replace File	Create Dir	Remove Dir
X	X	X	X	X	X	X

Directory Operations

Directory Listing Request	Directory Listing Response

Release of Retransmission Buffers

Incremental and Immediate	In total When "Finished" Received
X	

4. Timers and Counters

Timers

NAK Retry Timer	ACK Retry Timer	Prompt _NAK_ Timer	Async NAK Timer	Keep Alive Timer	Prompt _Keep Alive_ Timer	Inactivity Timer
X	X					

Counters

NAK Retry Counter	ACK Retry Counter
X	X

8.4 NASDA/NEC CORPORATION

CFDP Implementation Survey

Agency	Name	Submitted by
NASDA		Hiroaki Miyoshi

General Implementation Information

Platform	OS	Language
PC	Windows-NT Ver.4.0 SP3 or later	C++

Max. File Size	Max. Segment Size	Mechanism Used for Persistent Storage	Other Persistent Storage Options
65535	2000	DOS	

Underlying Communications Systems

CCSDS AOS VCDU	CCSDS TM_TC	CCSDS Prox_1	SCPS_NP	UDP_IP	Other
				X	

1. CFDP Procedures

CRC Proced.	Put Proced.	Transaction Start Proced.	PDU Forwarding Proced.	Copy File Proced.	Positive Ack. Proced.	Protocol Errors Proced.	Filestore Proced.
	X	X	X	X	X	X	

2. CFDP Protocol Classes

Unreliable Transfer	Reliable Transfer	Reliable Transfer by Proxy
X	X	

3. CFDP Protocol Options

2.A Options

End Type

Sender	Receiver
X	X

Put Modes

UnACK	NAK
X	X

Put NAK Modes

Immediate	Deferred	Prompted	Asynchronous
	X		

Put File Types

Bounded	Unbounded
X	

Segmentation Control (Record Boundaries Respected)

Yes	No
	X

Put Primitives (Receiving End)

File_segment_receive.ind
X

Put Error Responses (Sending End)

Ignore	Abandon	Cancel	Suspend
		X	

Put Error Responses (Receiving End)

Ignore	Abandon	Cancel	Suspend
		X	

Put Actions

Cancel_PutAction_	Suspend_PutAction_
X	

Cancel Put Action (Receiving End)

Discard data	Forward incomplete
X	

Put Report Modes (Sending End)

Prompted_Rpt_	Periodic

File Store Options

Create File	Delete File	Rename File	Append File	Replace File	Create Dir	Remove Dir

Directory Operations

Directory Listing Request	Directory Listing Response

Release of Retransmission Buffers

Incremental and Immediate	In total When "Finished" Received
	X

4. Timers and Counters

Timers

NAK Retry Timer	ACK Retry Timer	Prompt _NAK_ Timer	Async NAK Timer	Keep Alive Timer	Prompt _Keep Alive_ Timer	Inactivity Timer
X	X					X

Counters

NAK Retry Counter	ACK Retry Counter
X	X

9 INTER-AGENCY TESTS

NOTES

1 The tests reported in this section were executed in the process of developing the CFDP specifications. In the next release of this document, the reports herein will be replaced by reports on the testing using implementations conforming to the CFDP specifications contained in the next approved version of reference [1].

2 All references to Test IDs or Test Sequence IDs refer to the tests defined in annex B.

9.1 PURPOSE OF INTER-AGENCY TEST PROGRAM

The CFDP inter-Agency testing program has four distinct purposes.:

a) Verify the correctness of the protocol specification by creating multiple implementations according to that specification, and thoroughly test those implementations.

b) Provide measurements of the performance of the protocol and the resources required by the protocol from its hosting system, including the size of the software implementations.

c) Demonstrate the interoperability of independent implementations by inter-implementation testing.

d) Make available the tested implementations as reference implementations for the use of projects and programs which wish to use the CFDP.

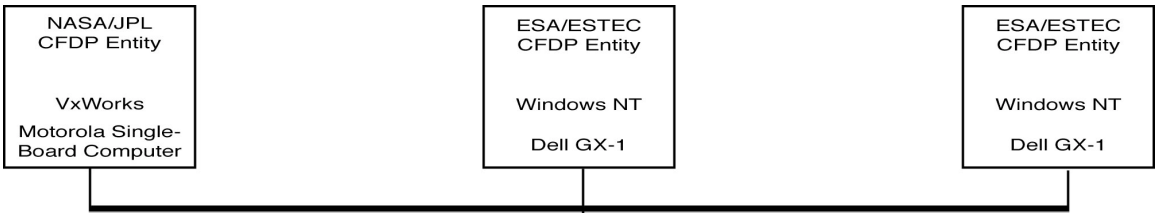
9.2 OVERVIEW OF TEST PROGRAM

Three inter-Agency testing workshops have been held: the first was held at the Applied Physics Laboratory (APL) of the Johns Hopkins University (JHU) in May 2000; the second and third were held at the DERA facility at Farnborough, U.K., in September and November 2000.

The participants in the first workshop were ESA/ESTEC and NASA/JPL, in the second DERA/BNSC and ESA/ESTEC and, in the third, DERA/BNSC, ESA/ESTEC, NASDA/NEC and, via the Internet, NASA/JPL. A summary of the results of these workshops is provided in the remainder of this section.

9.3 TEST REPORT SUMMARIES

9.3.1 CFDP MAY 2000 TEST WORKSHOP AT APL/JHU



Std. Test ID	Time	from/to	Mode	File size	Result	Notes
F1-1	0840	J to E	Unack	17 bytes	good	One data packet
F1-2	1755	J to E	Unack	50Kbytes	good	Multiple data packets
F1-1	1800	E to J	Unack	10 bytes	good	One data packet
F1-2	1840	E to J	Unack	50Kbytes	good	Multiple data packets
F1-3	1845	E to J	Ack	6K bytes	good	Multiple data packet, no errors. No TLV
F1-3	1850	J to E	Ack	6K bytes	good	Multiple data packets, no errors. No TLV
F1-3	1853	J to E	Ack	50K bytes	good	Multiple data packets, no errors. No TLV
F1-3	1855	E to J	ACK	50K bytes	incorrect	NAK and retransmission PDU formats verified. Duplicate data recognized in J entity.
F1-7	1910	J to E	ACK	msg to user	incorrect	Msg okay, process problem
F1-7	1916	E to J	ACK	msg to user	good	
F1-7	10/5/ 1150	J to E	ACK	msg to user	good	
F1-4	1310	J to E	ACK	5 Kbytes	good	Dropped one File Data PDU. Minor non-fatal Ack/NAK problem.
F1-4	1346	J to E	ACK	5 Kbytes	good	Problem fixed

F1-4	1402	E to J	ACK	5 Kbytes	good	Dropped one File Data PDU.
F1-8	1420	E to J	ACK		good	Sender Cancel.
F1-8	1435	J to E	ACK		good	Sender Cancel.

9.3.2 CFDP SEPTEMBER 2000 TEST WORKSHOP

9.3.2.1 Participants

a) BNSC/DERA

b) ESA/ESTEC

9.3.2.2 Functional Test Series 1

Src	**BNSC/DERA**	**ESA/ESTEC**
Dest	**ESA/ESTEC**	**BNSC/DERA**
1	Y	Y
2	Y	Y
3	Y	Y
4	Y	Y
5	Y	Y
6	1	1
7	Y	Y
8	2	Y
9	Y	Y
10	Y	Y

NOTES

1 Unable to generate out-of-order PDUs.

2 Cancellation only took effect after all File Data PDUs were transmitted for the first time.

BNSC code ignores incoming PDUs during the initial transmission of File Data PDUs, meaning it cannot be canceled or suspended during this phase.

9.3.2.3 Functional Test Series 2

Src	BNSC/DERA	ESA/ESTEC
Dest	**ESA/ESTEC**	**BNSC/DERA**
1	Y	Y
2	Y	Y
3	Y	Y
4	Y	Y
5	Y	Y
6	Y	Y
7	1	1

NOTE – Test skipped as BNSC code does not currently support Error Procedures.

9.3.2.4 Functional Test Series 3

Tests skipped as BNSC/DERA code does not support Filestore Operations.

9.3.2.5 Functional Test Series 4

Src	BNSC/DERA	ESA/ESTEC
Dest	**ESA/ESTEC**	**BNSC/DERA**
1	Y	Y
2	Y	Y
3		
4		
5		
6	Y	Y
7	1	Y
8	1	2
9		
10		
11		
12		

NOTES

1 Sender could not suspend during initial transmission of all File Data PDUs.

2 Discovered conflict with Suspend Procedures when both entities attempt to suspend at the same time, and only the suspending entity can resume. This had been identified previously from a theoretical argument, but it was reproduced accidentally during the workshop.

Suggested improvements:

a) Clarification of the default states of controls like Segmentation Control for other tests in the book, e.g., either test 5 or 6 will be redundant as it will have been performed in another test series, i.e., F1.3. This is similarly true for NAK modes.

b) Clarification of which Protocol Errors are involved in tests 9-12.

c) Review of Suspend Procedures.

9.3.3 CFDP NOVEMBER 2000 TEST WORKSHOP

9.3.3.1 Participants

a) BNSC/DERA

b) ESA/ESTEC

c) NASA/JPL (via Internet)

d) NASDA/NEC

9.3.3.2 Functional Test Series 1

Src	BNSC/DERA			ESA/ESTEC			NASA/JPL			NASDA/NEC		
Dest	**E-E**	**N-J**	**N-N**	**B-D**	**N-J**	**N-N**	**B-D**	**E-E**	**N-N**	**B-D**	**E-E**	**N-J**
1	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
2	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
3	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
4	Y	Y	4	Y	Y	4	Y	Y	Y	4	4	Y
5	Y	3		Y	Y		Y	Y	Y			Y
6	Y	1		Y	Y		1	1	1			1
7	Y	Y		Y	Y		Y	Y	Y			Y
8	Y	Y		Y	Y		Y	Y	Y			Y
9	Y	Y		Y	Y		Y	Y	Y			Y
10	Y	Y		2	Y		2	Y	Y			Y

NOTES

1 Unable to generate out-of-order PDUs.

2 Receiver had memory storage problems with 1 Mb file, but success with 500 kb.

3 Unable to generate duplicated PDUs.

4 NASDA/NEC initially had problems interpreting NAKs. This was fixed and tested successfully against NASA/JPL, but time constraints did not allow re-testing with BNSC/DERA or ESA/ESTEC.

9.3.3.2.1 Problems Encountered

a) NASDA had trouble understanding the offsets required in NAK PDUs from the description contained in reference [1]. This description requires some clarification.

b) NASDA had implemented the checksum in a different way than ESA, BNSC and NASA. However, the NASDA method seemed stronger than that originally proposed, and it has been adopted in issue 3.3 of reference [1].

9.3.3.2.2 Suggested Improvements

Add versions of tests 8 and 9 in unacknowledged mode.

9.3.3.3 Functional Test Series 2

Src	BNSC/DERA			ESA/ESTEC			NASA/JPL			NASDA/NEC		
Dest	**E-E**	**N-J**	**N-N**	**B-D**	**N-J**	**N-N**	**B-D**	**E-E**	**N-N**	**B-D**	**E-E**	**N-J**
1	Y	Y		Y	Y		2	Y	Y			Y
2	Y	Y		Y	Y		Y	Y	Y			Y
3	Y	Y		Y	Y		Y	Y	Y			Y
4	Y	Y		Y	Y		Y	Y	Y			Y
5	Y	Y		Y	Y		Y	Y	Y			Y
6	Y	Y		Y	3		2	Y	Y			Y
7	1	1	1	1	1	1	1	1	1	1	1	1

NOTES

1 Test skipped as description needs clarification.

2 Receiver gave incorrect NAK response to missing Metadata PDU.

3 Receiver gave incorrect NAK response to EOF PDU first.

9.3.3.3.1 Problems Encountered

Time was the main problem. NASDA fixed their NAK offsets remarkably quickly, but there was insufficient time to test against all of the other codes. Effort was concentrated on NASA-NASDA to speed progress.

9.3.3.3.2 Suggested Improvements

Clarification of Protocol Error definition. Test 7 refers to a Protocol Error without specifying which one, or which entity causes it. Specifying two particular Protocol Errors, e.g., dropping all ACK (EOF Complete) and Finished (Complete) PDUs to cause an error at the sender, and dropping all NAK PDUs to cause an error at the receiver, would make the tests clearer.

9.3.3.4 Functional Test Series 3

Src	BNSC/DERA			ESA/ESTEC			NASA/JPL			NASDA/NEC		
Dest	**E-E**	**N-J**	**N-N**	**B-D**	**N-J**	**N-N**	**B-D**	**E-E**	**N-N**	**B-D**	**E-E**	**N-J**
1												
2									Y			Y
3									Y			Y
4												
5												
6												
7									Y			Y
8									Y			Y
9												

Many of the filestore operations are not currently supported by the BNSC and NASDA codes.

10 REQUIREMENTS

10.1 GENERAL

This section contains the requirements for the CFDP. The development of the requirements was driven by a reference set of five scenarios. These scenarios are included herein. The requirements proper are divided into two subsections: the first lists the requirements for the protocol itself, and the second lists the requirements for the implementation of the protocol.

10.2 CONFIGURATION SCENARIOS

10.2.1 BASIS

Five operational configuration scenarios were used as the basis for the requirements for CFDP. The scenarios are described as both space-to-ground file transfer operations and as ground-to-space file transfer operations. The primary difference for ground-to-space transfers is that most spacecraft are capable of receiving transmissions from only one ground station at a time. Therefore, those configurations implying multiple simultaneous transmissions to a spacecraft in fact have serial non-overlapping access for uplink transmissions.

10.2.2 SPACECRAFT/NETWORK CONTROL CENTER (NCC) WITH NO INTERMEDIATE FILE TRANSFER ENTITY

10.2.2.1 Scenario 1

Scenario 1 consists of End-to-End service using no intermediate File Transfer (FT) entities, as shown in figure 10-1.

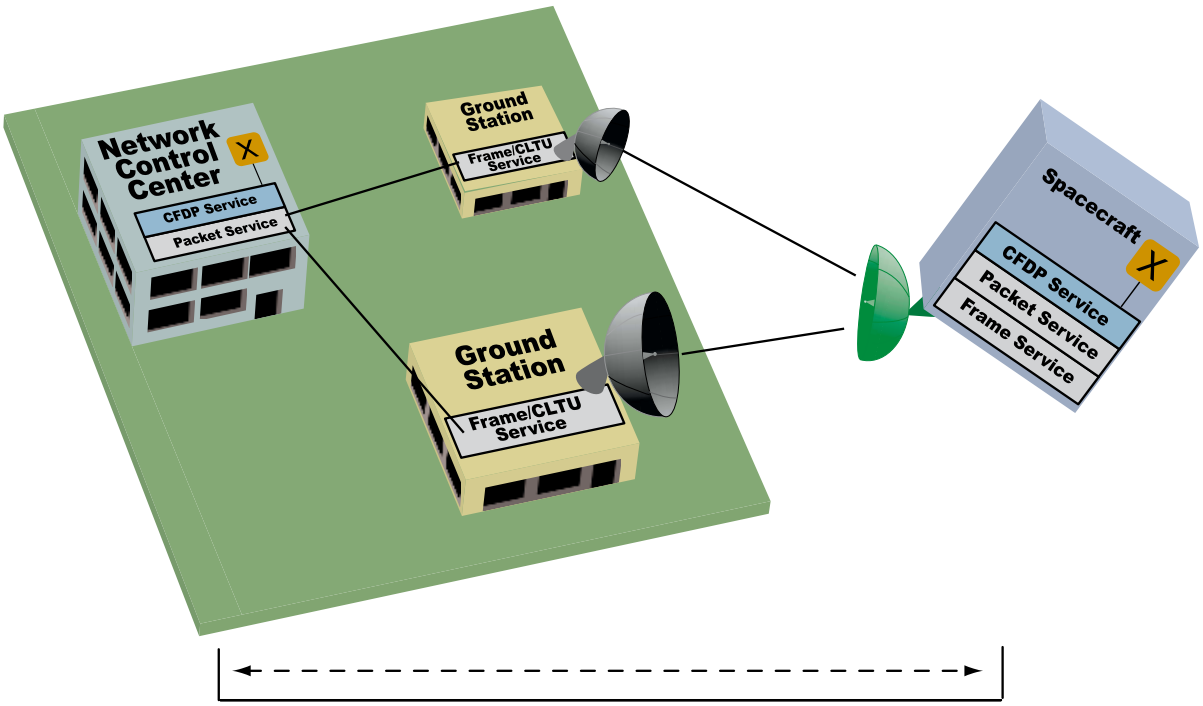


Figure 10-1: Scenario 1

10.2.2.2 Scenario 1: Space-to-Ground

In Scenario 1, the file transfer takes place from a spacecraft to its associated NCC. Multiple ground stations receive frames from the spacecraft and route them to the NCC, with or without extracting packets (i.e., the ground stations may extract the packets using the SLE packet service and forward the packets, or may instead forward the frames, in which case the packets are extracted at the NCC). The ground stations' frame acquisition may overlap one another in time or be entirely disjoint. At the NCC, the packets are passed to the FT entity for assembly and report generation. The reports are routed to the spacecraft's FT entity via the in-view ground station.

NOTE – The NCC's FT entity discards duplicate blocks received during overlapping contacts. The management of frame data at the ground station is not addressed by the protocol.

The NCC's FT entity detects loss and/or corruption of data blocks and requests that they be retransmitted; it also tells the spacecraft's FT entity which blocks it has successfully received. The spacecraft's FT entity retransmits blocks in response to requests from the NCC's FT entity, or in response to determination that an acknowledgment from the NCC's FT entity is overdue (either because the acknowledgment itself was lost, or because the blocks to be acknowledged were not received). The source FT entity (on the spacecraft) continues retransmission until the destination entity (in the NCC) has taken custody of the entire FTU.

On notification of complete reception, or on transaction cancellation (initiated by either of the two FT entities), the spacecraft's FT entity need no longer retain its copy of the FTU in a

retransmission buffer. If the data path is simplex (i.e., the NCC can never send data to the spacecraft), then the spacecraft's FT entity assumes that FTU reception is complete as soon as it has finished transmitting the FTU; it may optionally send some or all data blocks multiple times (i.e., 'proactive retransmission') in an attempt to improve the likelihood of successful initial FTU reception.

NOTES

1 The protocol is used to transfer files between space and ground file systems.

2 The protocol can cause file system management commands to be executed with respect to the remote file system (ground or space). FT entities issue those commands in response to file system management command PDUs.

3 The spacecraft can be anywhere in space, from near-Earth orbit to the furthest reaches of the solar system and beyond.

4 Multiple transfers may be in flight concurrently.

5 The protocol may operate over TM/TC packets.

6 Transfers can span link passes (contacts).

7 The protocol delivers a file completion map along with the file (which may be incomplete).

8 A file is defined to be an array of octets (not bits).

9 The 'ground' (the NCC) is a single protocol endpoint, a single FT entity; individual receiving stations are not FT entities in this scenario.

10 The protocol discards duplicate data.

11 The protocol is defined in levels to facilitate a range of implementation complexities from simple to complex. Metadata can command the destination FT entity to:

a) get and put;

b) plus delete, rename, etc.;

c) plus mkdir, rmdir, etc.;

d) perform other functions yet to be defined (e.g., append, rename, patch, read).

12 Support for time-outs: each FT entity involved in one link of a communication path is aware of the one-way light time between the two, and the presumed operative state of the other

13 Optional features:

a) send and forget (simplex transmission);

b) incremental NAK: the receiving FT entity additionally reports on its reception state (sends a NAK) immediately whenever it detects any missing data block (again, the NAK is automatic, but provides for manual intervention in case of anomaly).

10.2.2.3 Scenario 1: Ground-to-Space

Scenario 1 is also valid for ground-to-space file transfer. In that case, the file transfer takes place, for example, from an NCC to a spacecraft. Multiple ground stations receive packets or frames from the NCC (i.e., the ground stations may insert the packets into frames, or this may be done at the NCC, in which case the ground stations receive frames) and route them to the spacecraft. Because spacecraft usually (with the possible exception of large manned spacecraft) can support only one uplink at a time, the frames are sent to the spacecraft from one ground station at a time, in separate contacts. At the spacecraft the packets are passed to the FT entity for assembly and report generation. The reports are routed to the NCC's FT entity via the in view ground station.

NOTE – The spacecraft's FT entity discards any duplicate blocks which might have been caused by ground station-to-ground station switchovers.

The spacecraft's FT entity detects loss and/or corruption of data blocks and requests that they be retransmitted; it also tells the NCC's FT entity which blocks it has successfully received. The NCC's FT entity retransmits blocks in response to requests from the spacecraft's FT entity, or in response to determination that an acknowledgment from the spacecraft's FT entity is overdue (either because the acknowledgment itself was lost or because the blocks to be acknowledged were not received). The source FT entity (in the NCC) continues retransmission until the destination entity (in the spacecraft) has taken custody of the entire FTU.

On notification of complete reception, or on transaction cancellation (initiated by either of the two FT entities), the NCC's FT entity need no longer retain its copy of the FTU in a retransmission buffer. If, perhaps because of a spacecraft anomaly, the data path is simplex (i.e., the spacecraft cannot send data to the NCC), then the NCC's FT entity assumes that FTU reception is complete as soon as it has finished transmitting the FTU; it may optionally send some or all data blocks multiple times ('proactive retransmission') in an attempt to improve the likelihood of successful initial FTU reception.

10.3 PROTOCOL REQUIREMENTS

10.3.1 GENERAL

This subsection contains the File Delivery Protocol Functional Requirements. For ease of review, they are divided into five groups. These groups are:

a) Requirements Related to Communications.

b) Requirements Related to Underlying Layers.

c) Requirements Related to Structure.

d) Requirements Related to Capabilities.

e) Requirements Related to Records, Files, and File Management.

10.3.2 REQUIREMENTS RELATED TO COMMUNICATIONS

Many of the requirements for the protocol are set by the environment in which it must operate. These include the physical characteristics of the communications links, as well as the availability of those links. The physical characteristics of the communications links include their quality (noisiness), bandwidth, propagation delay, operating mode (Simplex, Half-Duplex, Full-Duplex), and availability. Refer to table 10-1.

Table 10-1: Requirements Related to Communications

Group Num.	Requirement	Req. Ref. Num.	Source
comm 01	The protocol shall be appropriate for both deep space and near earth missions.	01	E11, G1, I1, J15
comm 02	The protocol shall provide effective and efficient service over communications links with propagation delays spanning milliseconds to tens of hours.	02	C4, G3
comm 03	Round trip communications time shall be provided to the protocol from an external source.	66	J37
comm 04	The protocol shall provide effective and efficient service over communications links which are typically bandwidth-restricted.	03	C3
comm 05	The protocol shall provide effective and efficient service over communications links which may be significantly unbalanced in bandwidth.	04	C3, G2
comm 06	The protocol shall provide effective and efficient service when allocation of the available bandwidth is not under the control of the protocol.	05	C1
comm 07	The protocol shall provide effective and efficient service over communications links which have frequent outages.	06	J30
comm 08	The protocol shall provide effective and efficient service over communications links which have long outages.	07	G4, G12, J31
comm 09	The protocol must be capable of providing effective and efficient service over a simplex link.	19	C5, J16, J19
comm 10	The protocol must be capable of providing effective and efficient service over a half-duplex link.	20	C5, E15, J16
comm 11	The protocol must be capable of providing effective and efficient service over full-duplex links.	21	J16
comm 12	Where the underlying protocols can provide the appropriate level of responsiveness, the protocol shall operate when the underlying protocols in both directions provide Reliable service.	22	C1
comm 13	Where the underlying protocol can provide the appropriate level of responsiveness, the protocol shall operate when the underlying protocol in only one direction provides Reliable service.	23	C1, G5, J14
comm 14	The protocol shall operate when the underlying protocols in both directions provide Unreliable service.	24	C1, G5, J14

10.3.3 REQUIREMENTS RELATED TO UNDERLYING LAYERS

The protocol must be able to operate over a wide range of underlying services. Where the underlying services are CCSDS, it must operate over the CCSDS Path Service in Grades of Service 2 and 3. In addition, it must operate over conventional commercial protocols in order to provide required store-and-forward services. Refer to table 10-2.

Table 10-2: Requirements Related to Underlying Layers

Group Num.	Requirement	Req. Ref. Num.	Source
undr 01	The protocol shall provide the capability to operate over current CCSDS Packet Telemetry, Advanced Orbiting Systems, and Telecommand protocols and shall not inhibit the normal operation of these protocols.	11	C8, E2, E3, E4, E5, G9, G10, I12, J2, J26, J27
undr 02	The protocol shall provide the capability to operate over Transmission Control Protocol (TCP)/User Datagram Protocol (UDP).	50	E27, J2
undr 03	The protocol shall provide full capabilities over the services provided by existing packet recommendations.	75	E03
undr 04	Full advantage shall be taken of the characteristics of the Packet TM/TC service i.e. normally 'perfect' data in sequence with possible omissions.	76	E05

10.3.4 REQUIREMENTS RELATED TO STRUCTURE

Two requirements relate to the user-visible structure of the protocol, as described in table C-1.

Table 10-3: Requirements Related to Structure

Group Num.	Requirement	Req. Ref. Num.	Source
struct 01	The protocol shall operate between automated, essentially symmetrical peer entities.	09	I3
struct 02	A single service interface will be presented to the client.	10	E10
struct 03	The protocol shall be scaleable so that it may be used on relatively simple, current technology spacecraft, as well as on sophisticated, advanced design spacecraft.	60	G6, G7, G8, J1

10.3.5 REQUIREMENTS RELATED TO PROTOCOL CAPABILITIES

The largest group of requirements relate to the capabilities and operating characteristics which the protocol must possess. Refer to table C-2.

Table 10-4: Requirements Related to Capabilities

Group Num.	Requirement	Req. Ref. Num.	Source
cap 01	A protocol Peer shall be capable of both receiving and transmitting files simultaneously.	25	E23, G13, J1, J5
cap 02	A protocol Peer shall be capable of concurrently supporting multiple file transfer instances.	26	E23, G14, J4
cap 03	The protocol shall provide the capability to transfer both files (arrays of octets, which may or may not be further structured as arrays of CCSDS packets) and metadata (which may or may not pertain to those files).	39	I02
cap 04	A file is defined to be an array of octets (not bits).	65	J35
cap 05	The protocol shall handle variable record sizes.	40	E19
cap 06	The protocol shall allow file transfer up to (2^32)-1 octets.	42	E13
cap 07	The protocol shall allow requests for a file transfer to specify the file by name.	43	I8
cap 08	The protocol shall provide immediate access to the received data as it is received, i.e., without waiting for the file to be completed	37	E25, G17, J3
cap 09	The protocol shall provide the capability to operate in a 'Single Transmission' mode, in which the data are sent once and only once.	28	C10
cap 10	The protocol shall provide the capability to operate in a 'Selective Retransmission' mode, in which missing or corrupted sub-data units are identified by the receiving Peer to the sending Peer, and the sending Peer then retransmits those and only those sub-data units.	30	C12, G15, I10
cap 11	The protocol shall be automatic, but shall provide for manual intervention in case of anomaly.	78	E09
cap 12	The protocol shall support suspend and resume operations.	53	I13
cap 13	The receiving protocol Peer shall remove any duplicate data received.	61	G16, J36
cap 14	The protocol shall provide the capability of initiating a file transfer without transfer initiation handshaking between the Peers.	31	I7, J19
cap 15	The protocol receiving Peer shall provide the capability to, during the file transfer process, make available to the using	35	C9, I6, J7, J8,

Group Num.	Requirement	Req. Ref. Num.	Source
	Application the status of the available received data, including reporting that: a) data are still being received (and the available data do or do not contain errors), and b) data have been completely received (and retransmission requests are or are not pending) (and the available data do or do not contain errors).		J9
cap 16	The protocol receiving Peer shall provide the capability to periodically report comprehensive status back to the sending Peer.	32	J7, J8, J9
cap 17	The protocol receiving Peer shall not require acknowledgment of the comprehensive status reports to proceed if the file integrity is detected to be correct.	33	J19
cap 18	The protocol receiving Peer shall provide the capability to, upon receiving a complete and correct file, provide a final acknowledgment to the sending Peer.	36	I6, J24
cap 19	The protocol shall be capable of completion of a file transfer without transfer completion handshaking between the Peers.	38	I7, J3
cap 20	The protocol shall provide the capability to allow file transfers to span protocol Sender/protocol Receiver contacts.	62	E24, J33
cap 28	The protocol shall inform the recipient application that the file is available for use. If the file is incomplete, the temporary name being used by the protocol process shall be provided along with a completeness map.	64	J34
cap 29	The scope of the data being transferred may be multiple extents (not just a single length starting at zero), which may change over time.	72	J43
cap 30	The protocol shall provide proxy file service.	81	I15
cap 31	For operation over unreliable lower layers, a checksum for each file segment shall be optionally provided.	82	E28
cap 32	For bounded files, a checksum for the entire file shall be provided.	83	E29

10.3.6 MANAGEMENT

The requirements which delineate the record handling, file handling, file management, and directory management which the protocol must possess are listed in table 10-5.

Table 10-5: Requirements Related to Records, Files, and File Management

Group Num.	Requirement	Req. Ref. Num.	Source
rfm 01	The protocol shall assume the following set of file access primitives from the local file system: 'Open', 'Read', 'Write', 'Seek', 'Remove', and 'Close'.	44	E18, J28
rfm 02	The protocol shall provide File transfer capabilities of 'Get' (request file transfer from remote Peer to local Peer), and 'Put' (request file transfer from local Peer to remote Peer).	45	E20, G11, J32
rfm 03	The protocol shall provide the following file handling services: Load a New File, Send a File, Modify a File, and Replace an Existing File.	46	G11, J10, J32
rfm 04	The protocol shall provide the following file management services: Request a File, Rename a File, Delete a File, and Report a File Status.	47	E21, G11, J11, J32
rfm 05	The protocol shall provide the following file directory management services: Create directory, List directory, Rename directory, Delete directory, Change to directory, and Report current directory.	48	E22, G11, J29, J32
rfm 06	The protocol file transfer services shall be independent of local filing systems.	63	E26

10.4 IMPLEMENTATION REQUIREMENTS

The requirements on the implementation of the File Delivery Protocol are shown in table 10-6.

Table 10-6: Implementation Requirements

Group Num.	Requirement	Req. Ref. Num.	Source
imp 01	The protocol shall minimize the load on onboard computing resources.	58	C6, G8
imp 02	The protocol shall minimize the use of onboard memory resources.	59	C7, E1, G8
imp 03	The protocol specification shall be fully validated and tested.	56	J13
imp 04	The protocol sending Peer shall have the option of responding to the final acknowledgment of receipt by deleting the file that is known to have been correctly transmitted.	51	J25

ANNEX A

ACRONYMS AND ABBREVIATIONS

ACK	Positive Acknowledgment
APL	Applied Physics Laboratory (at Johns Hopkins University)
BEOP	Burst Error Occurrence Probability
BNSC	British National Space Centre
CCSDS	Consultative Committee for Space Data Systems
CFDP	CCSDS File Delivery Protocol
CNES	Centre National d'Etudes Spatiales
DERA	Defence Evaluation and Research Agency
EOF	End of File
ESOC	European Space Operations Centre
ESTEC	European Space Research and Technology Centre
FD(n)	File Data Segment
FIN	Finished (receiver to sender)
FDU	File Delivery Unit
FIFO	First-In-First-Out
FT	File Transfer
GEO	Geosynchronous Earth Orbit
GTO	Geosynchronous Transfer Orbit
GUI	Graphical User Interface
IDE	Integrated Development Environment
JHU	Johns Hopkins University

LEO	Low Earth Orbit
M	Metadata
MCC	Mission Control Center
MIB	Management Information Base
MSB	Most Significant Bit
NAK	Negative Acknowledgment
NCC	Network Control Center
OSI	Open Systems Interconnection
PDU	Protocol Data Unit
PRMPT	Prompt
RTM	Relay Testing Module
SAD	Software Architectural Design
TBS	To Be Supplied
TC	Telecommand
TCP	Transmission Control Protocol
TM	Telemetry
UDP	User Datagram Protocol
UT	Unitdata Transfer

ANNEX B

CFDP EXTENDED PROCEDURES

B1 CFDP EXTENDED PROCEDURES OPTIONS

Table B-1: Finished PDU Field Codes

Parameter	Values
Delivery Code	'0' - Data Complete '1' - Data Incomplete
End System Status	'0' - Generated by Waypoint '1' - Generated by End System

Table B-2: Extended Procedures Transaction Waypoint Options

Forwarding Method	Effect
Incremental and Immediate	Sends received PDUs to next entity as soon as received.
In Total Upon Complete Custody Acquisition	Sends FDU to next entity only when entire FDU has been received.

ANNEX C

REQUIREMENTS FOR CFDP EXTENDED PROCEDURES

C1 SPACECRAFT/USER VIA A SINGLE RELAY ENTITY

C1.1 OVERVIEW

Scenario 2 consists of a Hop-by-Hop service using an intermediate store-and-forward process, as shown in figure C-1.

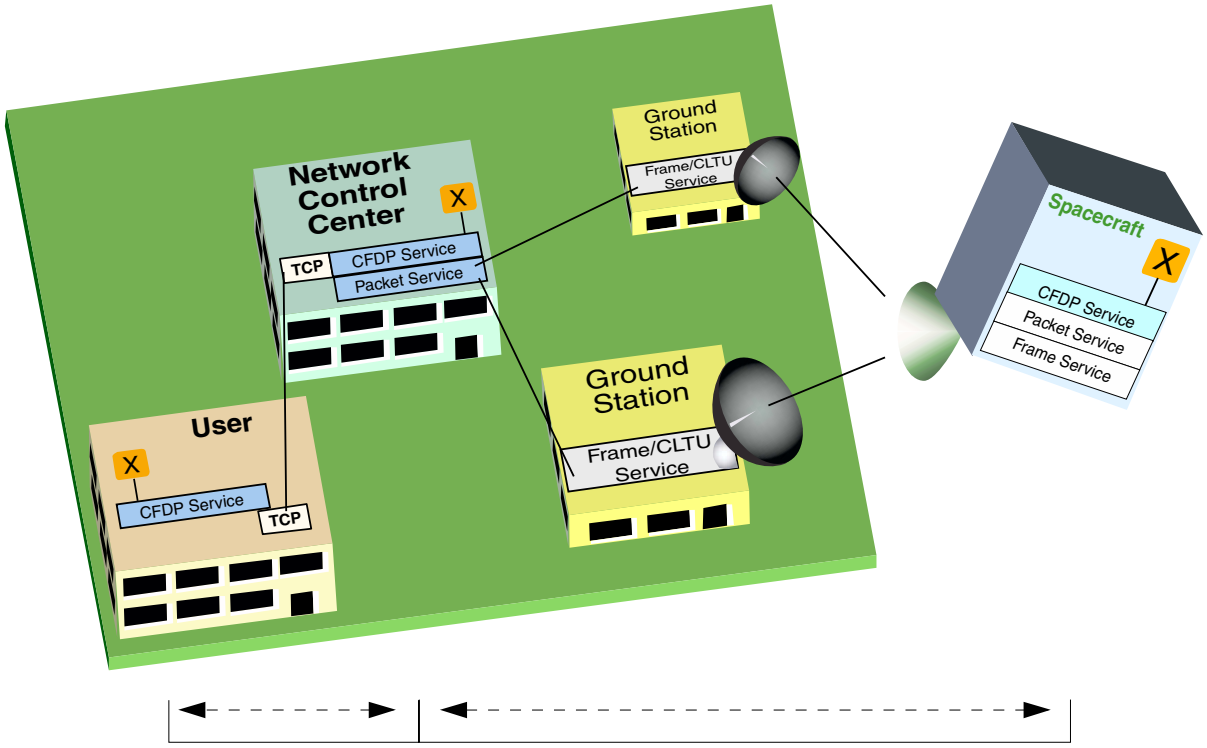


Figure C-1: Scenario 2

C1.2 SCENARIO 2: SPACE-TO-GROUND

The first Scenario 2 example is a file transfer from a spacecraft to a User via one intermediate entity, the NCC. The User may not always be online, or connection rate limitations might require the NCC to provide store and forward delivery. The file transfer from the spacecraft is performed by the NCC's FT entity. The NCC's FT entity serves as a reliable forwarding entity, allowing the spacecraft's FT entity to delete its copy of the file if necessary. File transfer to the User Application is accomplished by the NCC.

NOTE – The NCC's operations with the ground stations and spacecraft are as described in Scenario 1. The protocol can delete the file from the NCC when transfer to the User is accomplished. A protocol status report is sent from the User to the spacecraft.

The source FT entity (on the spacecraft) continues retransmission until the intermediate receiving entity (in the NCC) has taken custody of the entire FTU. The intermediate receiving entity (in the NCC) begins transmission of the FTU to the destination receiving entity (the User process) as soon as the applicable interim-acquisition rule has been satisfied; this rule might be declared in transaction metadata, or a default rule might be in effect. The intermediate receiving entity continues retransmission until the destination receiving entity has taken custody of the entire FTU, at which time the destination receiving entity notifies the User application.

NOTES

1 The file has proximate as well as final destinations; thus, the protocol has data block relay functionality.

2 There are also final and proximate sources; thus, the protocol has status report-relay functionality.

3 Each intermediate entity has store and forward capability; a ground station might or might not be configured as an intermediate entity.

4 The protocol has interim-acquisition rules in effect at each receiving FT entity, for example:

a) forward when N% of the file is received;

b) forward when the link from the sender is lost;

c) forward when the link to the receiver is available.

C1.3 SCENARIO 2: GROUND-TO-SPACE

Scenario 2 is also valid for ground-to-space file transfer. An example is a file transfer from a User to a spacecraft. As in the space-to-ground case, the transfer is via one intermediate entity, the NCC. The spacecraft may not always be online, or connection rate limitations might require the NCC to provide store and forward delivery. The file transfer from the User is performed by the NCC's FT entity. The NCC's FT entity serves as a reliable forwarding entity, allowing the User's FT entity to delete its copy of the file if necessary. File transfer to the spacecraft is accomplished by the NCC. As in Scenario 1, because spacecraft usually can support only one uplink at a time, the frames are sent to the spacecraft from one ground station at a time, in separate contacts.

C2 ROVER/NCC VIA MULTIPLE RELAY ENTITIES IN SERIES

C2.1 OVERVIEW

Scenario 3 consists of a service from a source through multiple relaying entities in series to a final destination, as shown in figure C-2.

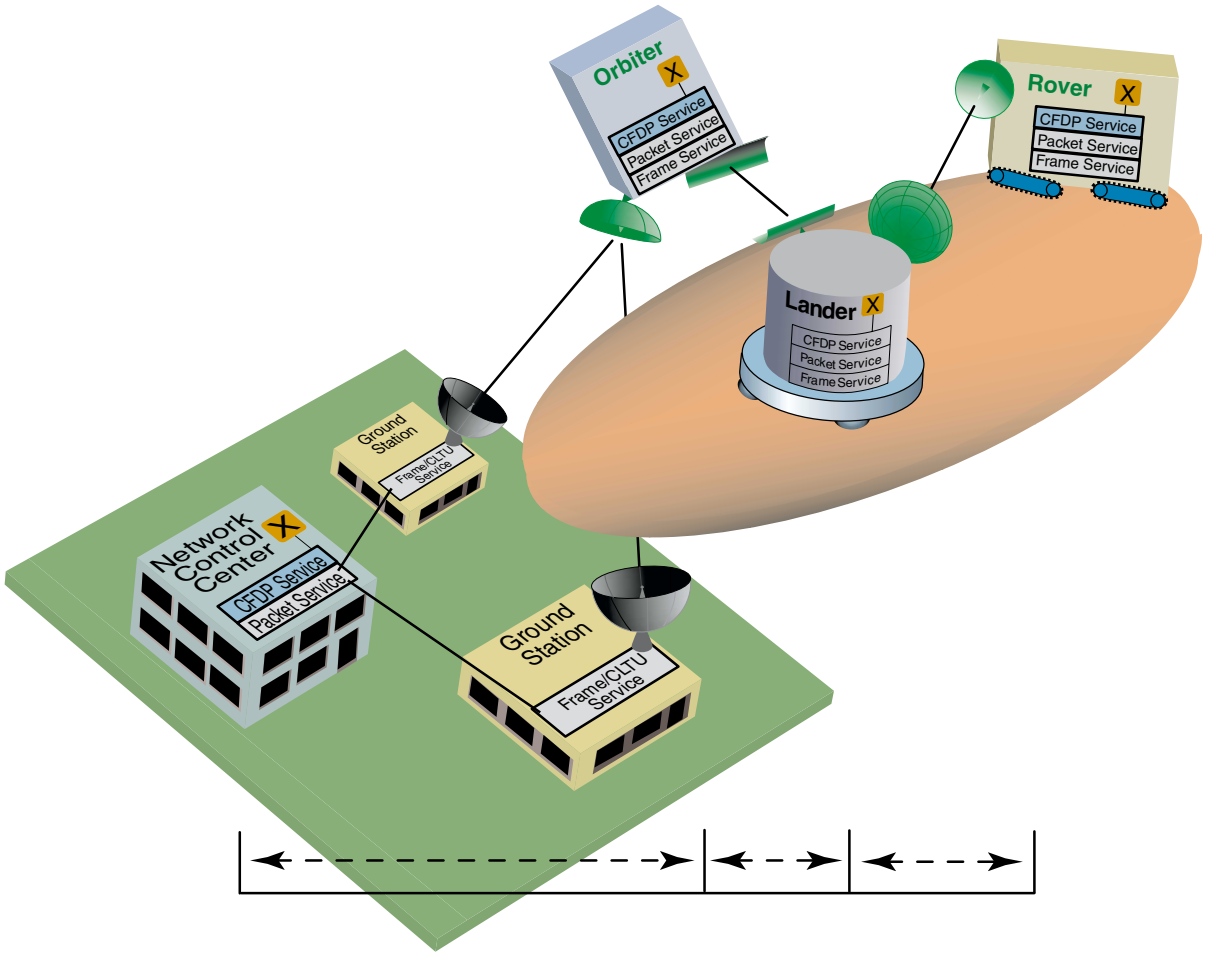


Figure C-2: Scenario 3

C2.2 SCENARIO 3: SPACE-TO-GROUND

The space-to-ground example is a file transfer from a planetary Rover to an NCC, via a planetary Lander, a planetary Orbiter, and ground stations on Earth. In the example, the Lander and the Orbiter are reliable entities. The files on the Rover and subsequently on the Lander and Orbiter are deleted after acknowledged transfer to the next 'reliable forwarding entity' is completed.

Each intermediate FT entity begins transmission as soon as the applicable interim-acquisition rule has been satisfied (and it has contact with the next FT entity), and continues retransmission until the corresponding receiving entity has taken custody of the entire FTU.

A minor variation of this scenario is to combine it with Scenario 2, i.e., make the NCC another in the series of intermediate entities and add a User application at the destination FT entity for the transaction.

C2.3 SCENARIO 3: GROUND-TO-SPACE

The ground-to-space example of Scenario 3 is a file transfer from an NCC to a planetary Rover, via ground stations on Earth, a planetary Orbiter, and a planetary Lander. In the example, the Orbiter and the Lander are reliable entities. The files in the NCC, and subsequently on the Orbiter and Lander, are deleted after acknowledged transfer to the next 'reliable forwarding entity' is completed.

Each intermediate FT entity begins transmission as soon as the applicable interim-acquisition rule has been satisfied (and it has contact with the next FT entity), and continues retransmission until the corresponding receiving entity has taken custody of the entire FTU.

As in Scenario 1, because spacecraft usually can support only one uplink at a time, the frames are sent to the Orbiter from one ground station at a time, in separate contacts.

C3 SPACECRAFT/USER VIA MULTIPLE INDEPENDENT RELAY ENTITIES IN PARALLEL

C3.1 OVERVIEW

Scenario 4 consists of a service from a source to a destination via multiple independent ground stations, as shown in C-3.

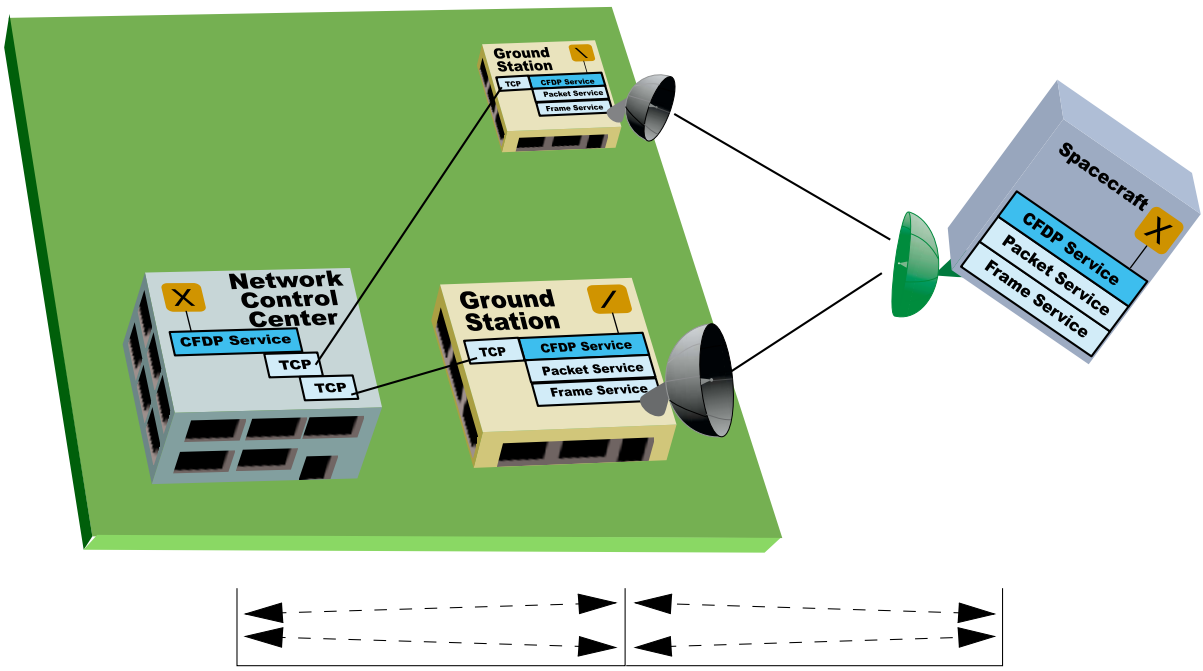


Figure C-3: Scenario 4

C3.2 SCENARIO 4: SPACE-TO-GROUND

In the space-to-ground version of Scenario 4, a file is sent from a spacecraft to an NCC via multiple ground stations, each of which acts as an intermediate FT entity. The source FT entity (on the spacecraft) transmits as much of the FTU as the contact time allows to the first intermediate entity that comes into view; that intermediate entity takes custody of that portion of the FTU and informs the source entity of how much of the FTU is in its custody. The source entity transmits the rest of the FTU to the next intermediate entity that comes into view; that entity too informs the source entity of what extents of the FTU are now in its custody. This transmission and retransmission continues until every extent of the FTU is in the custody of at least one of the intermediate entities, at which time the source entity relinquishes custody of the entire FTU.

The intermediate entities do not communicate with each other. Each one begins transmission to the destination entity as soon as the applicable interim-acquisition rule has been satisfied, and continues retransmission to the destination entity until the destination entity has taken custody of whatever extents of the FTU are in the custody of that intermediate entity; meanwhile, each entity requests retransmission from the source entity, as necessary, of whatever extents of the FTU it does not have in its custody.

The source entity's copy of the file (on the spacecraft) can be deleted once custody has been shifted to the reliable forwarding entities (ground stations). The partial files at each intermediate entity are deleted after transfer to the NCC.

NOTES

1 An intermediate entity can request 'retransmission' of data that was not originally transmitted to it (i.e., data that was transmitted to some other intermediate entity).

2 There may be multiple known proximate and/or final destinations.

C3.3 SCENARIO 4: GROUND-TO-SPACE

In the ground-to-space version of Scenario 4, a file is sent from an NCC to a spacecraft via multiple ground stations, each of which acts as an intermediate protocol entity. The NCC's FT entity transmits the entire file to each of the intermediate entities. Since each of the intermediate entities is a reliable forwarding entity, the source entity relinquishes custody of the entire FTU. The first intermediate entity at the first opportunity sends as much of the file to the spacecraft as the contact time allows. It then sends a status report to the source entity reporting its stop point in the file. The source entity sends that information to the next intermediate entity. This intermediate entity, when it gains contact with the destination entity, begins transmission of the file at that point. Again, as much more of the file is sent to the spacecraft as the contact time allows, and if the file transfer is not completed, a status report is sent to the source entity reporting its stop point in the file, and the process continues with the next intermediate entity. When the initial transmission of the FTU is complete, any required retransmission requests are sent from the destination entity to whichever intermediate entity is on contact with it. Since every intermediate entity possesses the entire FTU, it can honor any retransmission request. If retransmissions are not completed during that contact, then, when contact with the next intermediate entity begins, either a) a time-out condition occurs in the receiving entity, causing retransmission of any uncompleted retransmission requests, or b) the transition from Pause to Resume in the receiving entity causes such retransmission. The final intermediate entity releases custody of its FTU and also notifies the source entity. The source entity then notifies each of the intermediate entities, which then release custody of their copies of the FTU.

The intermediate entities do not communicate directly with one another. This can be especially important when the ground stations involved do not belong to the same organization, as for example in international cross support.

As in Scenario 1, because spacecraft usually can support only one uplink at a time, the frames are sent to the spacecraft from one ground station at a time, in separate contacts.

C4 SPACECRAFT/NCC VIA MULTIPLE COORDINATED RELAY ENTITIES IN PARALLEL

C4.1 OVERVIEW

Scenario 5 consists of a service from a spacecraft to an NCC via multiple ground stations, each of which acts as an intermediate FT entity, as in Scenario 4 above. However, in this

scenario the intermediate entities **do** communicate among themselves, as shown in figure C-4.

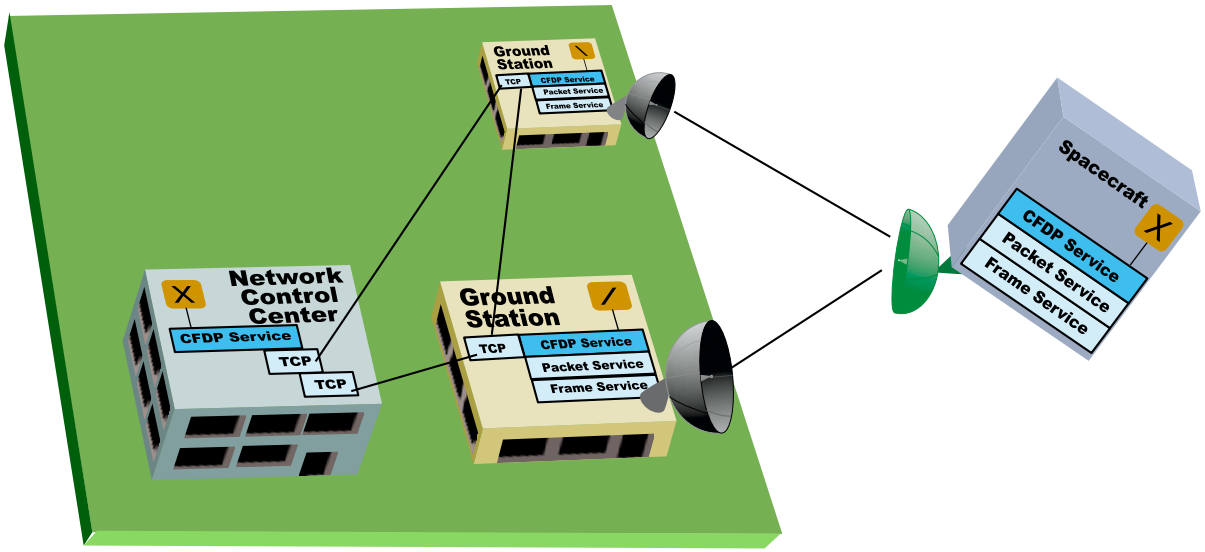


Figure C-4: Scenario 5

C4.2 SCENARIO 5: SPACE-TO-GROUND

In the space-to-ground example of this Scenario, the fact that the intermediate entities **do** communicate among themselves enables each one to know what extents of the FTU are in the custody of each of the others, so each intermediate entity need only request retransmission from the source entity (as necessary) of whatever FTU extents are not in the custody of any other intermediate entity.

C4.3 SCENARIO 5: GROUND-TO-SPACE

In the ground-to-space example of Scenario 5, as in the space-to-ground, the fact that the intermediate entities **do** communicate among themselves enables each one to know what extents of the FTU are in the custody of each of the others. Therefore, each intermediate entity can retransmit whatever extents are required by the destination entity, since, if it does not locally posses that extent, it can acquire it from that intermediate entity which does possess it. Refer to tables C-1 and C-2 for requirements related to structure and capabilities.

Table C-1: Requirements Related to Structure

Group Num.	Requirement	Req. Ref. Num.	Source
struct 02	A single service interface will be presented to the client (the addition of the extended protocol shall be evident in the quality of service and the multi-hop capability).	10	E10

Table C-2: Requirements Related to Capabilities

Group Num.	Requirement	Req. Ref. Num.	Source
cap 21	The protocol shall provide an automatic store-and-forward transfer capability.	49	I4
cap 22	The protocol shall support the transfer of files between multiple protocol agents in series (e.g., between ground and space as spacecraft and lander).	54	J17
cap 23	A file transfer operation may have proximate as well as final destinations. (The protocol shall provide relay functionality.)	67	J38
cap 24	A file transfer operation may also have final and proximate sources. (The protocol shall provide ACK-relay functionality.)	68	J39
cap 25	In store-and-forward modes the intermediate protocol agent shall provide the capability to forward a file which it has only partially received (e.g., forward that part of a file received during a single protocol Sender/protocol Receiver contact while waiting for the next contact, in which more of the file will be received).	73	J44
cap 26	In store-and-forward modes the intermediate protocol agent shall provide the capability to begin forwarding a file while it is still receiving that file.	74	J45
cap 27	For store-and-forward, the protocol shall have the optional capability to: – forward when the link from the sender is lost; – forward when the link to the receiver is available.	70	I14, J41