Using the CCSDS File Delivery Protocol (CFDP) on the Global Precipitation Measurement mission

Tim Ray
NASA – Goddard Space Flight Center

Abstract

The Consultative Committee for Space Data Systems (CCSDS) developed the CCSDS File Delivery Protocol (CFDP) to provide reliable delivery of files across space links. Space links are typically intermittent, requiring flexibility on the part of CFDP. Some aspects of that flexibility will be highlighted in this paper, which discusses the planned use of CFDP on the Global Precipitation Measurement (GPM) mission.

The operational scenario for GPM involves reliable downlink of science data files at a high datarate (approximately 4 megabits per second) over a space link that is not only intermittent, but also one-way most of the time. This paper will describe how that scenario is easily handled by CFDP, despite the fact that reliable delivery requires a feedback loop.

Keywords

File Transfer Protocol, CCSDS, and CFDP.

Introduction

CFDP is a protocol that provides reliable transfer of files (see references 1, 2, and 3). It was designed with space missions in mind, so it provides a variety of capabilities that are needed by space missions (see reference 4). CFDP is gaining acceptance within the space community. It has flown in space (on the AlSat-1 mission), and has been implemented for use on the MESSENGER mission (see reference 5) and the Deep Impact mission. It is planned for use on a variety of missions, including the Mars Reconnaissance Orbiter, James Webb Space Telescope, and Global Precipitation Measurement missions.

The developers of the CFDP protocol attempted to provide a relatively simple protocol with the flexibility required to handle a variety of space missions. Multiple independent implementations of CFDP were used to validate the protocol prior to its release. The validation testing attempted to cover a wide variety of usage scenarios. As the list of missions attempting to use CFDP grows, a greater spectrum of usage scenarios is unfolding, and the intended CFDP flexibility is being put to the test.

I have developed a CFDP software library that implements all the absolutely required aspects of the protocol, while allowing the library user the freedom to supply the implementation-dependent aspects. The intention is that the software library will be reused across multiple projects (for both flight and ground software), providing the bulk of each CFDP implementation.

A new problem

The GPM mission is proposing to use CFDP in a way that was not tested during the development of CFDP. They plan to downlink science data files over a link that is one-way (telemetry only) for most of each orbit. During a single orbit, it may be necessary to reliably downlink several hundred files. The GPM usage scenario is challenging because reliable data transfer requires a feedback loop, which in turn requires two-way communication. The required data throughput rate is large enough that the initial transmission of each file must occur during the one-way link, leaving the two-way link time for operation of the feedback loop. This requires that the feedback loop mechanism essentially be disabled during the one-way link and enabled during the two-way link.

Is CFDP flexible enough to allow this problem to be solved? Assuming that a solution is possible, how much work is involved, and can that solution be implemented for GPM while preserving a generic and reusable CFDP software library?

Looking for a solution

Consider how CFDP typically accomplishes reliable file transfer. Figure 1 shows the protocol data unit (PDU) flow for a typical file transfer transaction. In this example, the underlying link is 2-way, and is almost perfect (one chunk of filedata is dropped; all other PDUs are delivered). CFDP sends the entire file once, and then retransmits any missing portions based on feedback from the Receiver.

Figure 1 – A typical transaction over a 2-way link

```
        Sender                          Receiver
        ---------                       -----------

Metadata                       -->
Filedata bytes 1-1000          -->
Filedata bytes 1001-2000       -->
…
Filedata bytes 99001-100000 -->
EOF                            -->
                                    <-- Ack-EOF
                                    <-- Nak (41001-42000)
Filedata bytes 41001-42000  -->
                                    <-- Finished
Ack-Finished                   -->
```

The following sequence occurs for each reliable file transfer:

1) The Sender sends a Metadata PDU (which includes the name of the file to be sent) and sends the entire file once (one reasonably-sized chunk of file-data per PDU). The Receiver stores the file-data chunks that it receives (each Filedata PDU identifies its offset within the file).
2) The Sender periodically sends an EOF (end-of-file) PDU until it receives an acknowledgement from the Receiver. The Receiver sends an Ack-EOF PDU in response to the EOF PDU. Once the Sender receives acknowledgement for delivery of the EOF PDU, it becomes passive, and allows the Receiver to drive the protocol.
3) After acknowledging the EOF PDU, the Receiver periodically requests retransmission of any missing Metadata/Filedata until all data is received (the Sender retransmits data when requested).
4) The Receiver periodically sends a Finished PDU until it receives an acknowledgement from the Sender. At this point, both the Sender and Receiver shut down.

Note that steps 2, 3, and 4 each contain a feedback loop.

We first considered solving the problem by adjusting the feedback loop timers so that they would not expire during the 1-way link. That is, during the 1-way period the timers would be set to very large values, and then reset to their typical settings at the start of the 2-way period. This solution seemed simple until we realized that once a timer is running *in a particular transaction*, changes to its timeout value do not take effect *for that transaction* until the timer expires. In GPM's worst -case scenario, there would be several hundred independent file transfer transactions, and careful bookkeeping would be necessary to synchronize them. So, we kept looking.

One very good solution would be to suspend each transaction at the Sender end immediately after step 1, and resume all the transactions at the Sender end at the start of the 2-way link. With this solution, the Receiver could simply operate in the typical fashion. This would solve the problem because there is no feedback loop prior to step 2. However, CFDP does not provide a mechanism for doing this. A custom modification could be made to the CFDP software library if necessary, but a standard solution is preferable.

Another good solution is to suspend each transaction at the Sender end immediately after *sending* the first EOF message, suspend each transaction at the Receiving end immediately after *receiving* the EOF message, and then resume all the transactions at both ends at the start of the 2-way link. This effectively disables the feedback loop during the 1-way link. Figure 2 shows this solution concept. The solution assumes that the CFDP user will know the current link status (1-way or 2-way) at all times. We chose this solution because it is fairly simple, and can be implemented using standard CFDP mechanisms: **Indications** and **Requests**.

Figure 2 – the solution concept

```
During the one-way link period:
    Transaction #1 (file #1):
        Sender:
            Metadata, all Filedata, and EOF -->
            When EOF is sent, suspend the transaction, and
             initiate the transfer of file #2.
        Receiver:
            When EOF is received, suspend the transaction at the Receiver.
    Transaction #2 (file #2):
        Sender:
            Metadata, all Filedata, and EOF -->
            When EOF is sent, suspend the transaction, and
            initiate the transfer of file #3.
        Receiver:
            When EOF is received, suspend the transaction.
    …
    Transaction #n (file #n):
        Sender:
            Metadata, all Filedata, and EOF -->
            When EOF is sent, suspend the transaction.
        Receiver:
            When EOF is received, suspend the transaction.

During the two-way link period:
    Resume all transactions at the Sender and Receiver.
    The feedback loop operates in the normal manner, causing retransmission of
    any missing data.  This is similar to the bottom half of figure 1.
    Note: each transaction has an independent feedback loop.
```

CFDP Indications and Requests

CFDP provides services as requested by the **User**. The User is not necessarily a person; for example, it may be software. CFDP defines a discrete set of Indications that it uses to keep the User abreast of what is happening. For example, a **Transaction-Finished** Indication is issued to the User whenever a transaction completes. The set of Indications includes an **EOF-Sent** Indication that is issued when the first EOF message is sent (for each transaction) and an **EOF-Received** Indication that is issued when the EOF message is received (for each transaction). CFDP *does not specify the actions to take in response to Indications*; that is up to the User (i.e. the action taken is implementation-dependent).

CFDP also defines a discrete set of Requests that allow the User to control the protocol engine. For example, a **Put Request** is used to initiate a file transfer. The set of available Requests includes a **Suspend Request** (to suspend a single transaction) and a **Resume Request** (to resume a single transaction).

Using Indications and Requests to solve the problem

Figure 3 shows how CFDP Indications and Requests are used to implement the concept in Figure 2.

Figure 3 – the solution (using Indications and Requests)

During the one-way link period:

Transaction #1 (file #1):

Sender:

Metadata, all *Filedata*, and *EOF* **-->**
When EOF is sent, CFDP issues an **EOF-Sent Indication**.
User responds by submitting a **Suspend Request** (for the current transaction), and a **Put Request** (initiate the transfer of file #2).

Receiver:

When EOF is received, CFDP issues an **EOF-Received Indication**; User responds by submitting a **Suspend Request** (for the current transaction).

…

Transaction #n (file #n):

Sender:

Metadata, all *Filedata*, and *EOF* **-->**
When EOF is sent, CFDP issues an **EOF-Sent Indication**.
User responds by submitting a **Suspend Request** (for the current transaction).

Receiver:

When EOF is received, CFDP issues an **EOF-Received Indication** to the User; the User submits a **Suspend Request** (for the current transaction).

During the two-way link period:

The User submits a **Resume Request** for each transaction at both the Sender and Receiver; this enables the feedback loop, and the transactions complete in the normal manner.

Solution specifics (CFDP software library and GPM user code)

GPM uses one CFDP task onboard and one CFDP task in their ground system. Each CFDP task consists of the generic CFDP software library plus GPM-specific (user) code. The CFDP software library source code is the same for both flight and ground tasks. The user code is not.

The CFDP software library defines a callback routine interface for handling Indications. GPM software developers wrote a routine that responds to each Indication as they desire (e.g. as shown in figure 3), and registered it as the callback routine.

The CFDP software library defines a callback routine for outputting a PDU. GPM software developers wrote (and registered) their own callback routine for PDU output.

The CFDP software library provides an enhanced Resume Request that allows *all* transactions to be resumed with a single request. This enhancement was put in specifically for GPM, but does no harm (and may prove useful for other missions).

For GPM, science data files are placed into directories onboard as they are generated (this is outside of the CFDP application). A "hot-directory" mechanism automatically feeds files to CFDP as they appear in these directories; a priority-based scheme is used. The current plan includes the capability to enable or disable the hot-directory mechanism as desired. The current thinking is that we may want to disable new transactions at the beginning of the 2-way link (until the existing transactions complete).

The solution requires buffering of outgoing Ack-EOF and Nak PDUs at the receiver during the 1-way link. This buffering is necessary because the CFDP protocol engine "sends" the Ack-EOF and (if any data is missing) a Nak immediately after receiving the EOF PDU (just prior to issuing the EOF-Received Indication). This buffering is accomplished within the PDU-Output callback routine that is part of the GPM user code.

Finally, a mechanism was added to the user code to accept *Link Status* directives. These directives are issued (from outside of the CFDP task) at the transitions between 1-way and 2-way links. This allows the callback routines to operate properly (for example, the Receiver buffers outgoing PDUs if the Link Status is "one-way link; input only").

Conclusions

CFDP handled this problem well; ***the solution uses standard CFDP mechanisms.*** This was possible because the protocol allows the user to suspend and resume transactions, and also provides feedback to the user (specifically, Indications) without specifying the response to be taken.

The CFDP software library also handled this problem well; ***the solution preserves the generic and reusable nature of the software library*** (all customization required for the solution is in user code). This was possible because the library interface was designed for maximum flexibility. In particular, ***the flexibility provided by the CFDP protocol was passed on to the library user***. For example, the user response to CFDP Indications is handled by callback routines, as is the output of PDUs. As another example, callback routines are defined for all file access, allowing a library user to customize their Virtual Filestore (for example, to make a solid-state recorder appear to be a filesystem). The file access callback routine interfaces match those provided with the C compiler (e.g. *fopen, fread, fwrite*), and by default, the file access callback routines point to the standard routines (e.g. the *fopen* callback routine points to 'fopen'). The typical library user can simply accept the default file access built into the library, while a user with a special situation can easily customize file access within their implementation of CFDP.

Acknowledgements

Jeff Condron of Raytheon provided vital input regarding the design of the software library interface. Art Ferrer of NASA/Goddard helped in understanding the user perspective (and led the effort to produce the user code).

References

[1] CCSDS File Delivery Protocol (CFDP), CCSDS 727.0-B-2, Washington D.C., Consultative Committee for Space Data Systems, October 2002

[2] CCSDS File Delivery Protocol (CFDP) – Part 1: Introduction and Overview, CCSDS 720.1-G-1, Washington D.C., Consultative Committee for Space Data Systems, January 2002

[3] CCSDS File Delivery Protocol (CFDP) – Part 2: Implementers Guide, CCSDS 720.2-G-1, Washington D.C., Consultative Committee for Space Data Systems, January 2002

[4] Ray, Tim, "CCSDS File Delivery Protocol (CFDP) – Why it's Useful and How it Works", International Telemetering Conference, Las Vegas, Nevada, October 2003.

[5] Christopher Krupiarz, Scott Burleigh, Constantine Frangos, Brian Heggestad, Douglas Holland, Kevin Lyons, William Stratton, "The Use of the CCSDS File Delivery Protocol on MESSENGER", Space Ops, Houston, Texas, October 2002.