Scott, Keith L.

From: Scott, Keith L.
Sent: Thursday, April 29, 2010 1:19 PM
To: 'chris.taylor@esa.int'
Cc: dstanton@keltik.co.uk
Subject: Proposed resolutions to DTN Green Book Issues

Chris and Dai:

My proposed resolutions to your comments are below. Can you please let me know if these are sufficient to address your concerns or what you might propose as a way forward if they're not?

Best Regards,

--keith

ESA Comments to DTN Green book CESG review

3.3

This scenario doesn't seem to be a realistic driver. The scenario is well served by current technology where we keep the earth stations at a low protocol level. All the issues seem to point to is to not change the way we currently do things as they all seem to arise when you make the earth stations more intelligent. Suggest we delete the section.

- I concur that the current mode of operation deals with this scenario quite well. I thought the point though was exactly to drive out the issues that would need to be addressed when using a network protocol to support it, but I'm fine with removing it. If we axe this example I think some of the 'issues' will need to be mentioned in the next example.

4.2.3

Section does not cover essential telemetry. Second paragraph on describes a very specific method of hardware commanding that, for instance, bears no resemblance to the way ESA do it. Suggest we simplify to a single para e.g.

"An orbiter, for instance, thus performs a basic control and monitoring service to the lander using low level commands and telemetry formats which directly access the point-to-point link between orbiter and lander. The data are then transferred between orbiter and earth using high level (e.g.) file services. In this way the lander can be put into a known state even when higher layer functions are not available (e.g. due to an SEU induced equipment failure)."

- Works for me. Simply remove figures 4-3 and 4-4? We could include a figure describing ESA's preferred mechanism for low-layer commanding if you can describe / generate one. I'd like to keep the last figure (What is now 4—5) and reword the text describing it to not include any details of where the commands are within the data link frame / packet / other. We can drop figure 4—5 and the descriptive text as well if you think it's not useful. In that case the section would end after the second paragraph below.
- Proposed changes below

1.1.1 Network-Based Hardware Commanding Service

Spacecraft often require mechanisms to respond to very low-level commands in case higher spacecraft functions are not available or are not operating correctly. This allows the radio itself to detect the commands and to take actions without relying on higher protocol layers. Such commands are typically used to reboot the C&DH or to place the spacecraft into a known state, usually in preparation for re-starting higher layer services. A peer ability to receive low-level telemetry from a spacecraft whose networking services are not functioning is similarly required. This subsection describes how such a service can be implemented leveraging the multi-hop communications of the network service.

An orbiter, for instance, thus performs a basic control and monitoring service to the lander using low-level commands and telemetry formats which directly access the point-to-point link between orbiter and lander. These low-level commands can put the lander into a known state, even when higher layer (e.g. networking) functions are not available due to single-event-upsets or other equipment failure. Similarly, low-level commands can be used by the orbiter to extract data from the lander via low-level telemetry, and the data can then be transferred between orbiter and Earth using high level (e.g. file) services.

[[Changed 'Hardware command' to 'Low-Level command' in what's below and removed anything that seemed to imply a specific solution.]]

Figure 4-5 shows how a network-based low-level command application might function. The steps in the low-level commanding process are:

a) Low-level Commanding (LLC) application generates the low-level command for the target spacecraft

b) The LLC application sends the commanding data via the network layer, addressed to the LLC proxy application at the proximate relay. Together with the low-level command(s), the sending LLC application identifies the target spacecraft. This requires all other network-layer relays in the path to function properly.

c) The LLC message is forwarded to the Proximate Relay.

d) The HC application on the proximate relay consumes the LLC message and generates the data link layer frame(s) containing the commands to transmit to the target spacecraft.

e) The frame(s) containing the LLC commands are sent to the target spacecraft. It is assumed that these commands can be processed when neither the networking or higher-layer services on the target spacecraft are available.

f) The LLC is detected and acted on by the target spacecraft.

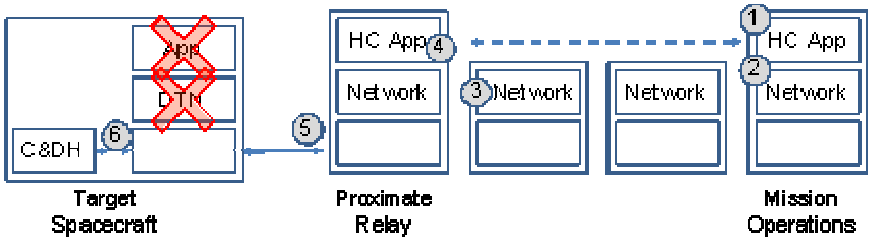

Figure 4-5: Low-Level Commanding Application Example

9.1.3 (storage) gives only a minimum indication of what is required in terms of the management of the local mass memory. Could we include some additional text which gives a bit more info:

a) Presumably DTN and LTP both need to be able to create files and delete files and maybe even copy, ?

b) This leads to the question of how the file store is managed. DTN doesn't have any built in file command capability so presumably this must be locally managed and when a bundle is received the local implementation creates the file etc and after transmission it does a clean-up?. I'm interested in this from a SOIS point of view because any request for access should go via a standard service.

c) Presumably we have the normal issues of filestore maintenance from ground when things get screwed, which may lead us to the need for a capability for file store management in the n-1 hop using direct commands? This is one of the issues I have as if DTN is always configured to use the file store (mass memory) and the filestore get screwed we cannot communicate using DTN!

- I've drafted some text below to try to address these. The main points are intended to be:
  - BP/LTP need storage to maintain state across reboots (expected or unexpected)
  - There are a number of ways the BP/LTP implementations might gain access to storage (e.g. BP/LTP given one big bunch of bytes to manage itself (I think this is ION's model); use a filesystem (the dtn2 implementation's model)
  - Mention SOIS File Access and File Management services specifically, and state that BP/LTP would want to use the SOIS File services if they're available.
  - Note that while bundles may be run through persistent storage, it's not a requirement for all bundles. You might keep small bundles that you're not the custodian for in memory, e.g. This would allow communication with a node even if the file system was hosed, provided the BP/LTP implementation had a way of delivering the bundles to applications without relying on the file system. Certainly possible, not sure if ION actually supports this or not.
  - Alternately, low-layer commanding (as above) could be developed to recover the FS from underneath BP/LTP.

- I propose the following text be added underneath the 'CFDP over BP over LTP using Zero-Copy-Objects' figure:

  In general, the implementations of the various protocol layers each require access to both persistent (e.g. disk) and ephemeral (e.g. memory) storage.

  Persistent storage is needed if the implementation needs to ensure that data are maintained across reboots (planned or unplanned) of the system. The persistent storage can take a number of forms depending on the implementation. For example, a single addressable array of octets in a solid-state data recorder or a pre-allocated file accessible via the SOIS File Access service [XXX_REF] may be allocated for network operations, with the various layers using zero-copy objects to pass pointers to data between layers. In this case the network stack would be responsible for managing the internals of the storage, adding and deleting user content as appropriate.

  If a more capable set of file services such as the SOIS File Management services are available, the networking implementation can make use of those to manage persistent data. In this case the networking implementation may create and delete files as well as simply modifying their contents. This would presumably simplify the implementations of BP and LTP. The disadvantage to this approach is that it admits the possibility that the network stack could consume more file resources than it is supposed to, which might be mitigated by a quota capability in the file store implementation.

  Sometimes file systems become corrupted to the point that they are unusable. In this case, a networking implementation that relied on a correctly functioning filestore would itself become unusable. We note here that while it is desirable to commit bundles to persistent store for reliability, it is not required and some bundles may be manipulated solely in memory. For example, bundles that are not forwarded using a reliable mechanism (e.g. bundles forwarded using the unreliable LTP service) that the node does not accept custody of need not be committed to persistent store. This would provide a mechanism for communicating with a node whose file store was corrupt, for example. Alternately, low-layer commands as described in Section 4.2.3 could be devised to attempt to recover the file store. This might provide a more robust solution, since if the filestore is corrupted and the network layer is restarted, even the network layer configuration information would be suspect.

Dr. Keith Scott

Principal Engineer, The MITRE Corporation
7515 Colshire Drive
M/S H340
McLean, VA 22102
USA