

Disruption Tolerant Networking Flight Validation Experiment on NASA's EPOXI Mission

Jay Wyatt, Scott Burleigh, Ross Jones, Leigh Torgerson, Steve Wissler

Jet Propulsion Laboratory
California Institute of Technology
4800 Oak Grove Drive
Pasadena, California, 91109

{E.Jay.Wyatt, Scott.C.Burleigh, Ross.M.Jones, Jordan.L.Torgerson, Steven.S.Wissler}@jpl.nasa.gov

Abstract— In October and November of 2008, the Jet Propulsion Laboratory installed and tested essential elements of Delay/Disruption Tolerant Networking (DTN) technology on the Deep Impact spacecraft. This experiment, called Deep Impact Network Experiment (DINET), was performed in close cooperation with the EPOXI project which has responsibility for the spacecraft. During DINET some 300 images were transmitted from the JPL nodes to the spacecraft. Then they were automatically forwarded from the spacecraft back to the JPL nodes, exercising DTN's bundle origination, transmission, acquisition, dynamic route computation, congestion control, prioritization, custody transfer, and automatic retransmission procedures, both on the spacecraft and on the ground, over a period of 27 days. All transmitted bundles were successfully received, without corruption. The DINET experiment demonstrated DTN readiness for operational use in space missions. This activity was part of a larger NASA space DTN development program to mature DTN to flight readiness for a wide variety of mission types by the end of 2011. This paper describes the DTN protocols, the flight demo implementation, validation metrics which were created for the experiment, and validation results.

Keywords- DTN, EPOXI, networking, protocols

I. INTRODUCTION

Disruption-Tolerant Networking (DTN; a.k.a. Delay-Tolerant Networking) is a communication architecture that is designed to provide automated data communication services in networks characterized by frequent and lengthy episodes of partitioning, lengthy signal propagation delays, and/or heterogeneity in protocol support below the application layer.

Research into DTN has culminated in the publication of Internet experimental RFCs (Requests For Comments) describing the overall architecture of DTN technology (RFC 4838), the core DTN Bundle Protocol (RFC 5050), and the Licklider Transmission Protocol for automatic retransmission of data lost in transit (RFC 5326), with others in progress. Although this research has been

substantially motivated by its applicability to such problem domains as sensor-based networks with scheduled intermittent connectivity, terrestrial wireless networks that cannot ordinarily maintain end-to-end connectivity, and underwater acoustic networks, the original driver for the research was the emerging need to provide capable network services in support of space flight operations.

Historically, communications in spacecraft mission operations have been managed by the spacecraft team. Transmission and reception episodes are individually configured, started, and ended by command. Reliability over deep space links is achieved by management: on loss of data, we command retransmission. Even the relaying of data from Mars rovers through Mars orbiters is managed: we send transmission commands to the rovers, and later we send transmission commands to the orbiters that received the data from the rovers.

An alternative approach would be to implement an automatic space data communications network, similar in capability to the Internet. The Internet protocols themselves, however, are generally unsuitable for this purpose because they rely on timely and continuous end-to-end delivery of data and acknowledgments: communication links to and from spacecraft are often subject to interruption and, for deep-space missions, signal propagation delays may be very large.

DTN is an alternative network architecture that is designed to address these problems. DTN runs as an "overlay" above the Internet where possible, but it runs directly over link-layer protocols, taking the place of the IP network protocol where necessary. That is, a TCP connection within an IP-based network may be one "link" of a DTN end-to-end data path; a deep-space R/F transmission may be another. Reliability is achieved by retransmission between relay points within the network, not end-to-end retransmission. There is no reliance on end-to-end acknowledgment. Route computation has temporal as well as topological elements, e.g., a schedule of planned contacts. Lengthy signal propagation delays don't compromise the accuracy of route computation. Forwarding at each router is automatic but not necessarily immediate: store-and-forward rather than "bent

pipe”, so link interruption doesn’t prevent the eventual delivery of data.

Our discussion of flight validation results will rest on an understanding of the following features of the DTN architecture:

- Priority. The forwarding of DTN network protocol data units, termed *bundles*, is informed by user-assigned priority markings: bundles may be assigned High, Medium, or Low priority, with higher-priority bundles forwarded in preference to lower-priority bundles wherever forwarding opportunities are constrained.
- Dynamic Routing. Traffic flow within a delay-tolerant network, as in the Internet, is more efficient if routers can automatically select different forwarding paths at different times, depending on nodes’ anticipated ability to forward data on a timely basis. Note, though, that Internet techniques for route computation are not well suited for operating a network over interplanetary distances where changes in topology may occur more rapidly than they can be reported.
- Automated Forwarding. Automatic initiation and termination of data transmission as transmission opportunities arise is a core capability of DTN.
- Custody Transfer. Performing retransmission of lost data at relay points rather than end-to-end enables each relay point along a bundle’s end-to-end path that receives the bundle to *take custody* of it, i.e. to relieve the prior retransmission point (source or relay) of any further responsibility for transmitting the bundle.
- Delay Tolerant Retransmission. The need to retransmit lost data is normally signaled by the receiver, upon detection of a gap in the received data. But in the event that insufficient data is received to enable gap detection – or in the event that the signal itself is lost in transmission – the only way to initiate retransmission is to detect the lapse of a timer prior to arrival of a positive acknowledgment. Computing accurate intervals for retransmission timers is especially challenging in a delay-tolerant network, as link interruption may defer transmission of acknowledgments. Human operators readily compute these intervals based on their knowledge of contact schedules; DTN must do the same in order to automate retransmission.
- Flow and Congestion Control. Automated forwarding and retransmission rely on the availability of data storage resources, so rates of data transmission and reception must be controlled in order to prevent exhaustion of those resources and failure of network operations. A fully automated network must accomplish this resource conservation automatically.

Interplanetary Overlay Network (ION) is an implementation of the DTN architecture that is specifically intended to be usable for interplanetary communications. As such, a key milestone in its development has been validation in operation on-board a functioning spacecraft. The Deep Impact Network experiment provided an opportunity not only to validate the software in flight but also to apply metrics by which the operational suitability of the software could be objectively assessed.

II. DINET OVERVIEW

The Deep Impact Network Experiment (DINET) was a technology validation experiment of JPL’s implementation of Delay-Tolerant Networking (DTN) protocols. The DINET development produced a version of JPL’s implementation of Delay-Tolerant Networking protocols in flight and ground software that is now at technology readiness level (TRL) 8. The DINET software (SW) is of sufficient quality that future flight projects can easily use it at low risk. DINET was implemented on the Deep Impact spacecraft and was closely coordinated with the EPOXI project. DINET operations were performed during the EPOXI spacecraft team “stand down” after Extrasolar Planet Observation and Characterization (EPOCH) operations and before the start of development for DIXI operations (i.e., during October and November 2008). DINET developments and operations were on a non-interference basis with EPOXI to the maximum extent possible. DINET was sponsored by NASA Office of Space Operations / Space Communications and Navigation (OSO/SCAN) via JPL DSN office Space Networking and Mission Automation. The total cost of DINET was \$1.4M, which included support for the EPOXI spacecraft team and their contractor Ball Aerospace and Technology Corporation.

III. DTN IMPLEMENTATION

A DTN implementation intended to function in an interplanetary network environment – specifically, aboard interplanetary research spacecraft separated from Earth and one another by vast distances – must operate successfully within two general classes of design constraints: link constraints and processor constraints.

Link constraints

All communications among interplanetary spacecraft are, obviously, wireless. Less obviously, those wireless links are generally slow and are usually asymmetric.

The electrical power provided to on-board radios is limited and antennae are relatively small, so signals are weak. This limits the speed at which data can be transmitted intelligibly

from an interplanetary spacecraft to Earth, usually to some rate on the order of 256 Kbps to 6 Mbps.

The electrical power provided to transmitters on Earth is certainly much greater, but the sensitivity of receivers on spacecraft is again constrained by limited power and antenna mass allowances. Because historically the volume of command traffic that had to be sent to spacecraft was far less than the volume of telemetry the spacecraft were expected to return, spacecraft receivers have historically been engineered for even lower data rates from Earth to the spacecraft, on the order of 1 to 2 Kbps.

As a result, the cost per octet of data transmission or reception is high and the links are heavily subscribed. Economical use of transmission and reception opportunities is therefore important, and transmission is designed to enable useful information to be obtained from brief communication opportunities: units of transmission are typically small, and the immediate delivery of even a small part (carefully delimited) of a large data object may be preferable to deferring delivery of the entire object until all parts have been acquired.

Processor constraints

The computing capability aboard a robotic interplanetary spacecraft is typically quite different from that provided by an engineering workstation on Earth. In part this is due, again, to the limited available electrical power and limited mass allowance within which a flight computer must operate. But these factors are exacerbated by the often intense radiation environment of deep space. In order to minimize errors in computation and storage, flight processors must be radiation-hardened and both dynamic memory and non-volatile storage (typically flash memory) must be radiation-tolerant. The additional engineering required for these adaptations takes time and is not inexpensive, and the market for radiation-hardened spacecraft computers is relatively small; for these reasons, the latest advances in processing technology are typically not available for use on interplanetary spacecraft, so flight computers are invariably slower than their Earth-bound counterparts. As a result, the cost per processing cycle is high and processors are heavily subscribed; economical use of processing resources is very important.

The nature of interplanetary spacecraft operations imposes a further constraint. These spacecraft are wholly robotic and are far beyond the reach of mission technicians; hands-on repairs are out of the question. Therefore the processing performed by the flight computer must be highly reliable, which in turn generally means that it must be highly predictable. Flight software is typically required to meet “hard” real-time processing deadlines, for which purpose it must be run within a hard real-time operating system (RTOS).

One other implication of the requirement for high reliability in flight software is that the dynamic allocation of system memory may be prohibited except in certain well-understood states, such as at system start-up. Unrestrained dynamic allocation of system memory introduces a degree of unpredictability into the overall flight system that can threaten the reliability of the computing environment and jeopardize the health of the vehicle.

1) ION Design Principles

The design of the ION implementation of DTN reflects several core principles that are intended to address these constraints.

Shared memory

Since ION must run on flight processors, it had to be designed to function successfully within an RTOS. Many real-time operating systems improve processing determinism by omitting the support for protected-memory models that is provided by Unix-like operating systems: all tasks have direct access to all regions of system memory. (In effect, all tasks operate in kernel mode rather than in user mode.) ION therefore had to be designed with no expectation of memory protection. But universally shared access to all memory can be viewed not only as a hazard but also as an opportunity. Placing a data object in shared memory is an extremely efficient means of passing data from one software task to another.

Zero-copy procedures

Given ION’s orientation toward the shared memory model, a further strategy for processing efficiency offers itself: if the data item appended to a linked list is merely a pointer to a large data object, rather than a copy, then we can further reduce processing overhead by eliminating the cost of byte-for-byte copying of large objects. Moreover, in the event that multiple software elements need to access the same large object at the same time, we can provide each such software element with a pointer to the object rather than its own copy (maintaining a count of references to assure that the object is not destroyed until all elements have relinquished their pointers). This serves to reduce somewhat the amount of memory needed for ION operations.

Highly distributed processing

The efficiency of inter-task communications based on shared memory makes it practical to distribute ION processing among multiple relatively simple pipelined tasks rather than localize it in a single, somewhat more complex daemon. This strategy has a number of advantages:

- The simplicity of each task reduces the sizes of the software modules, making them easier to understand

and maintain, and thus it can somewhat reduce the incidence of errors.

- The scope of the ION operating stack can be adjusted incrementally at run time, by spawning or terminating instances of configurable software elements, without increasing the size or complexity of any single task and without requiring that the stack as a whole be halted and restarted in a new configuration. In theory, a module could even be upgraded with new functionality and integrated into the stack without interrupting operations.
- The clear interfaces between tasks simplify the implementation of flow control measures to prevent uncontrolled resource consumption.

Portability

Designs based on these kinds of principles are foreign to many software developers, who may be far more comfortable in development environments supported by protected memory. It is typically much easier, for example, to develop software in a Linux environment than in VxWorks 5.4. However, the Linux environment is not the only one in which ION software must ultimately run.

Consequently, ION has been designed for easy portability. POSIX™ API functions are widely used, and differences in operating system support that are not concealed by the POSIX abstractions are encapsulated in two small modules of platform-sensitive ION code. The bulk of the ION software runs, without any source code modification whatsoever, equally well in Linux™ (Red Hat®, Fedora™, and Ubuntu™, so far), Solaris® 9, OS/X®, VxWorks® 5.4, and RTEMS™, on both 32-bit and 64-bit processors. Developers may compile and test ION modules in whatever environment they find most convenient. Moreover, there is no need to maintain separate versions of the implementation for flight and ground. This reduces cost and the risk of error in software maintenance.

2) *ION Software Elements*

The following elements of ION software, conforming to these principles, implement the DTN architecture in a manner that we believe will be suitable for interplanetary network applications.

Interplanetary Communication Infrastructure (ICI)

The ICI package in ION provides a number of core services that, from ION's point of view, implement what amounts to an extended POSIX-accessible operating system. ICI services include the following:

Platform

The platform system contains operating-system-sensitive code that enables ICI to present a single, consistent

programming interface to those common operating system services that multiple ION modules utilize. For example, the platform system implements a standard semaphore abstraction that may invisibly be mapped to underlying POSIX semaphores, SVR4 IPC semaphores, or VxWorks semaphores, depending on which operating system the package is compiled for. The platform system also implements a standard shared-memory abstraction, enabling software running on operating systems both with and without memory protection to participate readily in ION's shared-memory-based computing environment.

Personal Space Management (PSM)

Although sound flight software design may prohibit the uncontrolled dynamic management of system memory, private management of assigned, fixed blocks of system memory is standard practice. Often that private management amounts to merely controlling the reuse of fixed-size rows in static tables, but such techniques can be awkward and may not make the most efficient use of available memory. The ICI package provides an alternative, called PSM, which performs high-speed dynamic allocation and recovery of variable-size memory objects *within* an assigned memory block of fixed size.

Memmgr

The static allocation of privately-managed blocks of system memory for different purposes implies the need for multiple memory management regimes, and in some cases a program that interacts with multiple software elements may need to participate in the private shared-memory management regimes of all. ICI's memmgr system enables multiple memory managers – for multiple privately-managed blocks of system memory – to coexist within ION and be concurrently available to ION software elements.

Lyst

The lyst system is a comprehensive, powerful, and efficient system for managing doubly-linked lists in private memory. It is the model for a number of other list management systems supported by ICI; as noted earlier, linked lists are heavily used in ION inter-task communication.

Smlist

Smlist is another doubly-linked list management service. It differs from lyst in that the lists it manages reside in shared (rather than private) DRAM, so operations on them must be semaphore-protected to prevent race conditions.

Simple Data Recorder (SDR)

SDR is a system for managing non-volatile storage, built on exactly the same model as PSM. Put another way, SDR is a small and simple “persistent object” system or “object database”. It enables straightforward management of linked lists (and other data structures of arbitrary complexity) in non-volatile storage, nominally within a single file whose

size is pre-defined and fixed. SDR includes a transaction mechanism that protects database integrity by ensuring that the failure of any database operation will cause all other operations undertaken within the same transaction to be backed out. The intent of the system is to assure retention of coherent protocol engine state even in the event of an unplanned flight computer reboot in the midst of communication activity.

Zero-Copy Objects (ZCO)

ION's zero-copy objects system leverages the SDR system's storage flexibility to let user application data be encapsulated in any number of layers of protocol without copying the successively augmented protocol data unit from one layer to the next. It also implements a reference counting system that enables protocol data to be processed by multiple software elements concurrently – e.g., a bundle may be both delivered to a local endpoint and, at the same time, queued for forwarding to another node – without requiring that distinct copies of the data be provided to each element.

Licklider Transmission Protocol (LTP)

The ION implementation of LTP conforms fully to RFC 5326, but it also provides two additional features that enhance functionality without affecting interoperability with other implementations:

- The service data units – nominally bundles – passed to LTP for transmission may be aggregated into larger blocks before segmentation. By controlling block size we can control the volume of acknowledgment traffic generated as blocks are received, for improved accommodation of highly asynchronous data rates.
- The maximum number of transmission sessions that may be concurrently managed by LTP (a protocol control parameter), multiplied by the maximum block size, constitutes a transmission “window” – the basis for a delay-tolerant, non-conversational flow control service over interplanetary links

In the ION stack, LTP serves effectively the same role that is performed by TCP in the Internet architecture, providing flow control and retransmission-based reliability.

All LTP session state is safely retained in an SDR database for rapid recovery from a spacecraft or software fault.

Bundle Protocol (BP)

The ION implementation of BP conforms fully to RFC 5050, including support for the following standard capabilities:

- Prioritization of data flows
- Bundle reassembly from fragments
- Flexible status reporting

- Custody transfer, including re-forwarding of custodial bundles upon failure of nominally reliable convergence-layer transmission

The system also provides two additional features that enhance functionality without affecting interoperability with other implementations:

- Rate control provides support for congestion forecasting and avoidance.
- Bundle headers are encoded into compressed form before issuance, to reduce protocol overhead and improve link utilization.

In addition, ION BP includes an implementation of Contact Graph Routing (CGR), a system for computing dynamic routes through time-varying network topology assembled from scheduled, bounded communication opportunities. However, the details of CGR are beyond the scope of this paper.

To summarize, BP serves effectively the same role that is performed by IP in the Internet architecture, providing route computation, forwarding, congestion avoidance, and control over quality of service. Together, the BP/LTP combination offers capabilities comparable to TCP/IP in the Internet.

All bundle transmission state is safely retained in an SDR database for rapid recovery from a spacecraft or software fault.

3) ION implementation architecture

The ION implementation of BP/LTP is designed to work well within the constraints of the spacecraft flight software environment, emphasizing safety and efficiency. Figure 1 provides an overview of ION's architecture.

A few notes on this main line data flow:

- For simplicity, the data flow depicted here is a “loopback” flow in which a single BP “node” is shown sending data to itself (a useful configuration for test purposes). In order to depict typical operations over a network we would need two instances of this node diagram, such that the <LSO> task of one node is shown sending data to the <LSI> task of the other and vice versa.
- A BP application or application service (such as Remote AMS) that has access to the local BP node – for our purposes, the “sender” – invokes the bp_send function to send a unit of application data to a remote counterpart. The destination of the application data unit is expressed as a BP endpoint ID (EID). The application data unit is encapsulated in a bundle and is queued for forwarding.
- The forwarder task identified by the “scheme” portion of the bundle's destination EID removes the bundle from the

forwarding queue and computes a route to the destination EID.

- The output task for LTP transmission to the selected proximate node removes the bundle from the transmission queue and invokes the `ltp_send` function to append it to a *block* that is being assembled for transmission to the proximate node. (Because LTP acknowledgment traffic is issued on a per-block basis, we can limit the amount of acknowledgment traffic on the network by aggregating multiple bundles into a single block rather than transmitting each bundle in its own block.)
- The *ltpmeter* task for the selected proximate node divides the aggregated block into multiple segments and enqueues them for transmission by underlying link-layer transmission software, such as an implementation of the CCSDS AOS protocol.
- Underlying link-layer software at the sending node transmits the segments to its counterpart at the proximate node (the receiver), where they are used to reassemble the transmission block.
- The receiving node's input task for LTP reception extracts the bundles from the reassembled block and dispatches them: each bundle whose final destination is some other node is queued for forwarding, just like bundles created by local applications, while each bundle whose final destination is the local node is queued for delivery to whatever application "opens" the BP endpoint identified by the bundle's final destination endpoint ID.
- The destination application or application service at the receiving node opens the appropriate BP endpoint and invokes the `bp_receive` function to remove the bundle from the associated delivery queue and extract the original application data unit, which it can then process.

The DTN protocols are at relatively high layers of the communication protocol "stack" and rely on the support of communication software at lower layers to effect, for example, signal radiation and acquisition. Existing EPOXI operational software provides this support but is not designed to interact with the ION software, and vice versa.

An additional increment of DINET software, called Deep Impact Adaptation Software (DIAS), is therefore needed to act as an intermediary between ION and the operational software currently residing on the spacecraft and in the DI ground data system. The DIAS system enables the exchange of data between ION modules and DI operational software modules, thereby indirectly enabling the flow of DINET data, without requiring significant modification of DI flight or ground software.

The fundamental design decision underlying the DIAS design is simple. To minimize modification of DI operational software, we merely replace DI's implementation of the CCSDS File Delivery Protocol

(CFDP) with a CFDP simulator, called "PX". DI operational software, both in flight and on the ground, continues to invoke the CFDP protocol data unit (PDU) transmission and reception functions exactly as it does now, but the PDUs that are transmitted and received are neither produced nor consumed by CFDP protocol engines. Instead those PDUs are artificially produced and consumed by the PX system, which simply encapsulates *segments* of DTN data in bogus CFDP file data segment PDUs (FPDUs). In effect, we "tunnel" DTN traffic through underlying CFDP.

IV. EPOXI OPERATIONS

The EPOXI (Deep Impact extended mission) spacecraft was a unique opportunity to demonstrate the DINET technology. The EPOXI spacecraft has a backup flight computer which is always on and available for communications.

- 1) The prime flight computer controls all spacecraft functions, even while communication with the ground is through the backup flight computer.
- 2) The EPOXI spacecraft already had the CFDP protocol implemented for file transfer to and from the ground.
- 3) The EPOXI project was able to benefit from the DINET effort and had the personnel available for flight software implementation and test, as well as spacecraft operations during the DINET experiment.

It was critical to the EPOXI project that the DINET experiment posed minimal risk to the EPOXI mission. The EPOXI flight team worked closely with the DINET team to design an implementation approach that minimized risk to the spacecraft. The DINET software was installed on the backup software partition on the backup flight computer. Once the backup flight computer was booted with the DINET software, the boot configuration was restored to the original EPOXI software load. In the event of a spacecraft problem requiring a flight computer side swap, the backup computer with the DIINET software would be re-booted as prime, with the original EPOXI software running.

EPOXI operations during the month of DINET operations was performed by uploading a sequence to the spacecraft, which would switch the telemetry source to the SDST between the prime and backup flight computers, corresponding to the contact graph intervals installed during the DINET software upload. This made spacecraft operations "hands-off" during the DINET operational passes. The EPOXI flight control team would switch the updated DINET ground system software into place during the DINET operational passes and turn over the data-link to the DINET operations team.

V. EXPERIMENT DESIGN

The basic topology of DINET is shown in Figure 2 (i.e., two surface assets, a relay orbiter, and Earth). The surface assets

are designated Mars and Phobos, and the Deep Impact (DI) spacecraft fills the role of the relay orbiter.

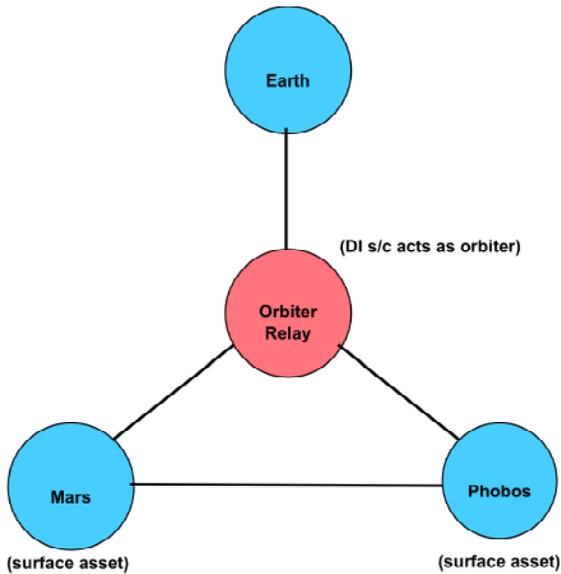


Figure 2 DINET Topology

Figure 3 shows how this topology was implemented during the experiment. The ION software with the DTN protocols was resident in each of the eleven network nodes.

The 4-week period of DINET operations was divided into two configurations (*a* and *b*) of four tracking passes each. Configuration *a* had no injection of artificial data loss. During configuration *b*, 3.125% of all LTP segments were randomly discarded upon reception at the DI spacecraft and at each of the three DSOT nodes. On the fourth tracking pass of each segment, the contact between Phobos and EPOXI was omitted. A brief “cross-link” contact between Phobos and Mars was scheduled for a time shortly before the 4th tracking pass of each experiment, providing an alternate path for data from Phobos.

VI. DINET OPERATIONS

An essential component of the DINET project is the Experiment Operations Center (EOC). The EOC served to:

- Produce Experiment Payload Data: Input JPEG image files as a single file per bundle, Mark bundle priority, Meter output to specified data rate.
- Consume Experiment Payload Data: Store in local file system at node upon reception, Display image upon reception.
- Consume Software Diagnostic Messages (ION logs): ION log messages transmitted to EOC software via TCP/IP socket, Received messages parsed & stored in a SQL database.

- Consume Protocol Diagnostic Messages (BSRs): BSRs transmitted from ION nodes to EOC software via the ION stack, Received messages parsed & stored in an SQL database.
- EOC Bundle Network Configured / Monitored With GUI: BSRs transmitted from ION nodes to EOC software via the ION stack, Received messages parsed & stored in a SQL database, Messages displayed through the GUI in real time.

The EOC generated and received the test communications traffic as well as “out-of-DTN band” command and control traffic of the DTN experiment, stored DTN flight test information in a database, provided display systems for monitoring DTN operations status and statistics (e.g., bundle throughput), and supported query and analyses of the data collected.

The DINET EOC was located within the JPL Protocol Technology Lab (PTL). The PTL provides connectivity to other NASA centers and external entities, and is itself a node in the larger DTN Experiment Network (DEN). The DINET EOC is envisioned to become a general tool in this broader context of experimental testing of DTN across a geographically dispersed user community.

VII. FLIGHT VALIDATION RESULTS

Four specific performance evaluation metrics were created to aid in the flight validation of DTN. This section explains those metrics and reports on DINET’s performance against them.

Terms of validation

Let G_{XYZ} denote the transmission opportunity – or *contact* – from node X to node Y on DINET pass $\#Z$. The duration of G_{XYZ} in seconds, denoted by D_{XYZ} , is the end time of G_{XYZ} minus the start time of G_{XYZ} . The data rate of G_{XYZ} in bytes per second is denoted by C_{XYZ} . The raw *capacity* of XYZ , denoted by K_{XYZ} , is equal to $D_{XYZ} * C_{XYZ}$. (Note that this is ideal capacity; the actual capacity of the link will be the ideal capacity reduced by actual signal noise on XYZ . Moreover, transient outages in transmission – as were experienced during four of the eight DINET transmission opportunities – necessarily reduce the total capacity of an opportunity.)

The total data return capacity S_{72a} from the EPOXI spacecraft (node 7) to the Earth subnet (node 2) while DINET is in configuration *a* is $\sum K_{72Z}$ for $Z = 1 \rightarrow 4$. The total data return capacity S_{72b} from the EPOXI spacecraft (node 7) to the Earth subnet (node 2) while DINET is in configuration *b* is $\sum K_{72Z}$ for $Z = 5 \rightarrow 8$.

The total data return capacity S_{M7a} from the two Mars subnets (nodes 3 and 5) to the EPOXI spacecraft (node 7) while DINET is in configuration a is $\sum K_{M7Z}$ for $Z = 1 \rightarrow 4$. The total data return capacity S_{M7b} from the two Mars subnets (nodes 3 and 5) to the DI spacecraft (node 7) while DINET is in configuration b is $\sum K_{M7Z}$ for $Z = 5 \rightarrow 8$.

The EPOXI spacecraft is the bottleneck in the flow of data from the Mars subnets to the Earth subnet: the *total science data return capacity* of DINET in configuration a , S_{M2a} , is either the capacity of the transmission opportunities from the Mars subnets to EPOXI or the capacity of the transmission opportunities from EPOXI to the Earth subnet, whichever is less. That is, $S_{M2a} = S_{M7a} \perp S_{72a}$ and $S_{M2b} = S_{M7b} \perp S_{72b}$.

The volume of priority-0 science data that is received at the Earth subnet over the entire course of DINET while in configuration a is denoted by R_{0a} . Similarly, the volume of priority-1 and priority-2 science data received at the Earth subnet over the entire course of DINET while in configuration a is denoted by R_{1a} and R_{2a} . The *raw volume of science data received* at the Earth subnet over the entire course of DINET in configuration a , R_{Ta} , is the sum of these: $R_{Ta} = R_{0a} + R_{1a} + R_{2a}$. Similarly, $R_{Tb} = R_{0b} + R_{1b} + R_{2b}$.

The *urgency-weighted volume of science data received* at the Earth subnet over the entire course of DINET in configuration a , W_{Ta} , is the weighted sum: $W_{Ta} = R_{0a} + (2 * R_{1a}) + (4 * R_{2a})$. Similarly, $W_{Tb} = R_{0b} + (2 * R_{1b}) + (4 * R_{2b})$.

The *reference volume of priority-0 science data* received at the Earth subnet while DINET is in configuration a , denoted by Q_{0a} , is computed as R_{Ta} multiplied by the proportion of all image bundles that were published with priority 0 during this phase of the experiment. (This is the proportion of R_{Ta} that we would expect to be priority-0 data, that is, the expected value of R_{0a} if there were no reordering of data transmissions in the network due to priority.) Similarly, $Q_{1a} = .60 * R_{Ta}$ and $Q_{2a} = .25 * R_{Ta}$, and the same relationships can be expressed for the configuration- b phase of the experiment as well.

The *urgency-weighted reference volume of science data received* at the Earth subnet while DINET is in configuration a , V_{Ta} , is the weighted sum: $V_{Ta} = (.5 * Q_{0a}) + Q_{1a} + (2.0 * Q_{2a})$. Similarly, $V_{Tb} = (.5 * Q_{0b}) + Q_{1b} + (2.0 * Q_{2b})$.

The size of the Interplanetary Overlay Network (ION) data store at each node X , I_X , is a DINET configuration parameter. The *size of the traffic storage allocation* A_X at each node X is computed by $A_X = .6 * I_X$.

The *total unassigned space* S_{XZ} at each node X for pass Z was reported by each node at least once on each day on which there was a tracking pass.

A *path* from node A to node B while DINET is in configuration a is any series of J distinct contacts $G_{X(1)Y(1)a}$, $G_{X(2)Y(2)a}$, ..., $G_{X(J)Y(J)a}$ such that (a) $X_1 = A$, (b) $Y_J = B$, (c) X_N (where $N > 1$) = Y_{N-1} , and (d) the start time of $G_{X(N)Y(N)a}$ (where $N > 1$) \geq the stop time of $G_{X(N-1)Y(N-1)a}$. The *net capacity* for a given path is the smallest value of contact capacity among all the contacts included in that path.

A *multipath* is a set of zero or more paths from node A to node B. Formally:

- The *net capacity* for a multipath containing zero paths is zero.
- The *net capacity* for a multipath that is formed by adding a path to a multipath is the sum of the net capacities of the path and multipath **or** the smallest value of contact capacity among all contacts that are common to the path and multipath, whichever is less.

Metric 1 – Path utilization rate (U)

Path utilization rate for DINET in configuration a is given by $U_a = R_{Ta} / S_{M2a}$. It measures the effectiveness of automatic forwarding, custody transfer, and delay-tolerant retransmission.

Validation criteria:

$U_a > 90\%$. (DTN uses both high-rate and low-rate links efficiently.)

$U_b > 90\%$. (DTN remains efficient despite an increase in the rate of data loss.)

Findings:

Analysis of the DINET experiment log indicates that U_a was 76.2% and U_b was 72.4%.

Note, however, that passes 2 and 8 were underutilized due to insufficiency of offered uplink data as discussed later, so their path utilization rates don't accurately reflect protocol efficiency. Additionally, note that about 20% of available uplink capacity was consumed by link service overhead, mainly telecommand coding. When only passes 1, 3, 4, 5, 6, and 7 are considered and all non-DTN overhead is subtracted from available transmission capacity, U_a and U_b are 97.4% and 92.5% respectively. With these provisos, both validation criteria were satisfied.

Note that the increased data loss rate in configuration b was found to correlate to a reduced path utilization rate as expected.

Metric 2 – Delivery acceleration ratio (G)

The delivery acceleration ratio for configuration a is given by $G_a = W_{Ta} / V_{Ta}$. It measures the effectiveness of the priority system.

Validation criteria:

$G_a > 1.05$ (Prioritization accelerates the delivery of urgent data.)

$G_b > 1.1$ (The advantage of prioritization increases with the rate of data loss.)

Findings:

Analysis of the DINET experiment log indicates that G_a was 1.10 and G_b was 1.12. Both validation criteria were satisfied.

Metric 3 – ION node storage utilization

Retention of a stable margin of unassigned space at each node measures the effectiveness of congestion control.

Validation criteria:

The total number of bundles for which custody is refused anywhere in the network for the reason “depleted storage”, throughout each configuration, is always zero. (We never run out of storage anywhere.)

$N_{X7} = N_{X6}$ for all values of X. (Storage utilization stabilizes over the course of network operations.)

Findings:

Analysis of the DINET experiment log indicates that both validation criteria were satisfied, *except* that N_{X7} was 156,816 bytes less than N_{X6} for node 10 (only). N_{10} had remained constant from passes 4 through pass 6. We suspect that some new functionality requiring additional storage space – possibly not related to the DTN protocols – was initially exercised on node 10 after pass 6 and prior to pass 7; analysis is continuing.

Metric 4 – Multipath advantage

The *multipath advantage* M_{AB} for traffic on the multipath from A to B conferred by the addition of a path is computed as the net capacity of the augmented multipath divided by the net capacity of the multipath excluding the added path, minus 1. When routing is static, all paths must necessarily comprise contacts between the same pairs of nodes in the same sequence; this precludes the addition of other paths to the multipath, limiting total multipath capacity. Multipath advantage therefore measures the effectiveness of dynamic routing.

Validation criteria:

The multipath advantage for traffic from node 20 to node 8 that is conferred by the cross-link contacts

between nodes 10 and 6 is greater than 20%. (Dynamic routing among multiple possible paths increases the total network capacity from Phobos to Earth.)

Findings:

The computed multipath advantage for traffic from node 20 to node 8 through the entire DINET experiment is 35%. Thus, the validation criterion was satisfied. Note, however, that errors in the implementation of dynamic routing prevented the expression of this advantage in improvements in delivery acceleration ratio. This metric will be revisited in future DINET experiments.

ACKNOWLEDGMENT

The work described in this report was performed at the Jet Propulsion Laboratory, California Institute of Technology under a contract with the National Aeronautics and Space Administration (NASA). Reference herein to any specific commercial product, process or service by trade name, trademark, manufacturer, or otherwise, does not constitute or imply its endorsement by the United States Government, NASA or the Jet Propulsion Laboratory, California Institute of Technology. DINET was implemented by the following personnel: Rashied Amini, Yan Brenman, Scott Burleigh, Loren Clare, Micah Clark, Andre Girerd, Son Ho, Nuha Jawad, Ross Jones, Margaret Lam, Marisol Mercado, Amalaye Oyake, Richard Rieber, Joshua Schoolcraft, Leigh Torgerson, Shin-Ywan Wang and Jay Wyatt. The following members of the EPOXI project team were essential to the success of DINET; Steve Wissler, Richard Reiber, Greg LaBorde, Leticia Montanez and Al Nakata.

The Deep Impact Networking Experiment was sponsored by the Space Communications and Navigation Office in NASA's Space Operations Mission Directorate. NASA's Science Mission Directorate and Discovery Program provided experimental access to the EPOXI spacecraft.

The EPOXI mission team provided critical support throughout development and operations. The following members of the EPOXI project team were essential to the success of DINET; Steve Wissler, Richard Reiber, Greg LaBorde, Leticia Montanez and Al Nakata. Finally, Rich Benson provided DSN scheduling support.

REFERENCES

- [1] JPL Publication 09-2 Disruption Tolerant Network Flight Validation Report
- [2] 1. K. Scott and S. Burleigh, *Bundle Protocol Specification*, RFC 5050, Internet Society, Reston, VA, November 2007.

[3] 2. M. Ramadas, S. Burleigh and S. Farrell, *Licklider Transmission Protocol—Specification*, RFC 5326, Internet Society, Reston, VA, September 2008.

[4] 3. R.W. Clayton, P.M. Davis, X. Perez-Campos, “Seismic Structure of the Subducted Cocos Plate,”

American Geophysical Union, abstract #T32A-01, Fall Meeting 2007.

[5] 4. A. Doria, “Saami Network Connectivity: Technical Overview of SNC.” Available at www.cdt.luth.se/babylon/snc; accessed February 11, 2009.

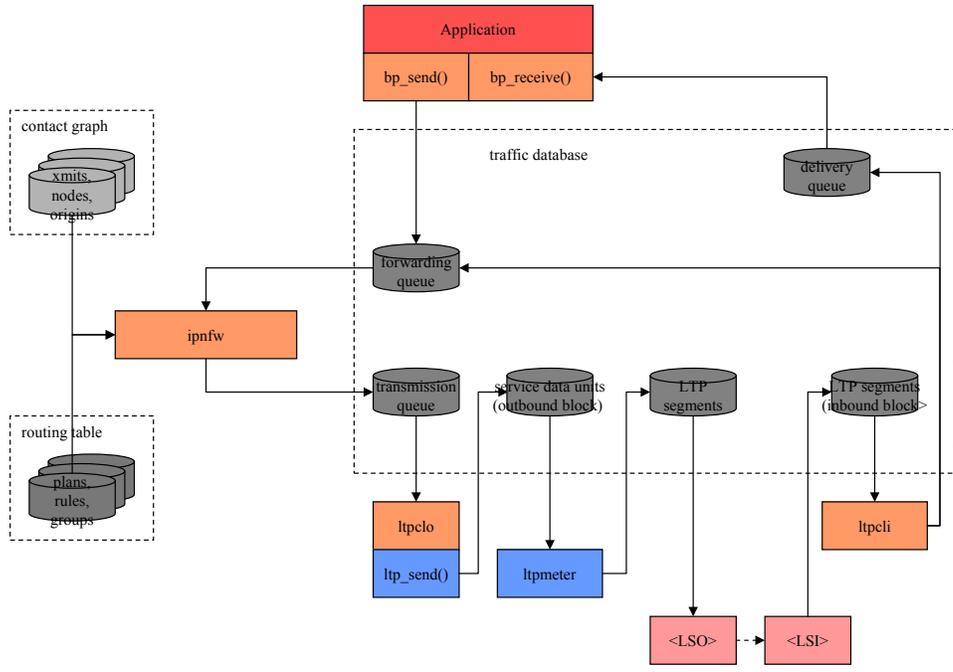


Figure 1 ION General Processing Flow

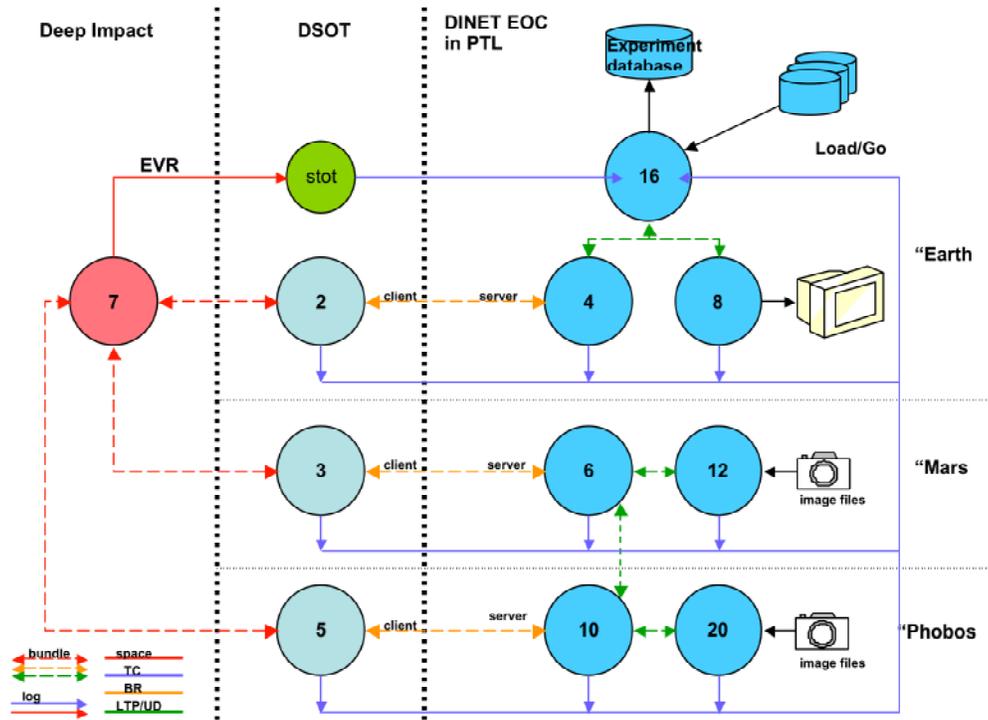


Figure 3 DINET Topology as Physically Implemented

Send images from nodes 12 to node 8 via nodes 6, 3, 7, 2, 4. Also send images from nodes 20 to node 8 via nodes 10, 5, 7, 2, 4. BRS = Bundle Relay Service; LTP = Licklider Transmission Protocol; UDP = user datagram protocol