# Delay-Tolerant Networking for Space Flight Operations: Design and Development

Scott C. Burleigh[*]
*Jet Propulsion Laboratory, California Institute of Technology, Pasadena, California, 91109*

**Interplanetary Overlay Network (ION) is an implementation of the Delay-Tolerant Networking (DTN) architecture that is specifically intended to be usable for interplanetary communications. As such, its design differs in several ways from that of other implementations including DTN2, the reference implementation of the DTN Bundle Protocol. This paper briefly reviews the constraints on interplanetary communication that argue against the suitability not only of most off-the-shelf Internet technology but also of many DTN implementations. It then describes the operational components of ION, including its implementations of both the Bundle Protocol and the Licklider Transmission Protocol, noting the ways in which they address those constraints. Features of the system are explained with reference to their counterparts in the Internet architecture. The paper concludes with some notes on ION testing experience to date and plans for additional development in the future.**

## I. Introduction

DELAY-TOLERANT Networking[1] (DTN) 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.

Although research in DTN 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[2].

Interplanetary Overlay Network (ION) is an implementation of the DTN architecture that is specifically intended to be usable for interplanetary communications. As such, its design differs in several ways from that of other implementations of the DTN architecture.

This paper briefly reviews the constraints on interplanetary communication that argue against the suitability not only of most off-the-shelf Internet technology but also of many DTN implementations. It then describes the operational components of ION, including its implementations of both the Bundle Protocol[3] and the Licklider Transmission Protocol[4], noting the ways in which they address those constraints. Features of the system are explained with reference to their counterparts in the Internet architecture. The paper concludes with some notes on ION testing experience to date and plans for additional development in the future.

## II. Implementations of DTN Protocols

### A. Bundle Protocol

The DTN Bundle Protocol (BP) performs routing and forwarding functions within a delay-tolerant network in a manner that is roughly analogous to the operation of the Internet Protocol (IP) in the Internet.

The reference implementation of BP is an open-source code base often referred to as "DTN2", which can be freely downloaded from the web site of DTN Research Group (DTNRG) at http://www.dtnrg.org/wiki/Code. DTN2 is written in C++ and TCL. It has been widely used in DTN research projects[†] and is well supported by an active

[*] Principal Engineer, Systems Engineering Section, 4800 Oak Grove Drive m/s 301-490, Pasadena CA 91109
[†] See http://www.dtnrg.org/wiki/Docs#head-182f5222a6cce62677fcc9feff26639222bc00fe, "Projects Using the DTN reference implementation."

user community, with extensive documentation and at least two extant simulators. Support for TCP communication at the convergence layer is included.

"DASM" (http://www.symob.net/dasm.htm) is an implementation of BP designed for embedded use in mobile phones based on the Symbian operating system. It is adapted from the DTN2 code base and includes support for Bluetooth communication at the "convergence layer" underlying BP.

An implementation of BP in the C# language for the .NET environment has been developed at the Georgia Institute of Technology[‡]. It includes support for TCP communication at the convergence layer.

An implementation of BP in Java (http://irg.cs.ohiou.edu/ocp/bundling.html) has been developed at Ohio University. It includes support for Licklider Transmission Protocol (LTP) communication at the convergence layer.

### B. Licklider Transmission Protocol

The Licklider Transmission Protocol (LTP) provides automated retransmission of data that was lost or corrupted in transit between two BP nodes communicating over a medium in which this reliability is not otherwise available – for example, over a radio link between spacecraft that are separated by distances on the order of light minutes.

The reference implementation of the LTP is a Java code base developed and supported at Ohio University, available for download at http://irg.cs.ohiou.edu/ocp/ltp.html.

A second implementation of LTP, written in C++, has been developed at Trinity College Dublin, Ireland (https://down.dsg.cs.tcd.ie/ltplib). It has been in use for water quality monitoring research since June of 2005, providing a standardized retransmission mechanism for otherwise unreliable UDP/IP traffic.

An experimental implementation of LTP intended for use in space flight missions was developed by the Johns Hopkins University Applied Physics Laboratory and was tested in a simulated Mars environment at the NASA Jet Propulsion Laboratory in 2006[§].

## III. Flight Environment Constraints on a 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.

### A. 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.

### B. 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

[‡] Private communication with Jon Olson, jsolson@damogran.org.

[§] Private communication with Chris Krupiarz, Christopher.Krupiarz@jhuapl.edu.

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.

## IV. ION Design Principles

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

### A. 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.

ION is designed to exploit this opportunity as fully as possible. In particular, virtually all inter-task communication in ION follows this model:

- The sending task takes a mutual exclusion semaphore (*mutex*) protecting a linked list in shared memory (either DRAM or non-volatile memory), appends a data item to the list, releases the mutex, and gives a “signal” semaphore associated with the list to announce that the list is now non-empty.
- The receiving task, which is already pended on the linked list’s associated signal semaphore, resumes execution when the semaphore is given. It takes the associated mutex, extracts the next data item from the list, releases the mutex, and proceeds to operate on the data item from the sending task.

Semaphore operations are typically extremely fast, as is the storage and retrieval of data in memory, so this inter-task communication model is suitably efficient for flight software.

### B. 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.

### C. 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 (see Fig. 1). 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.

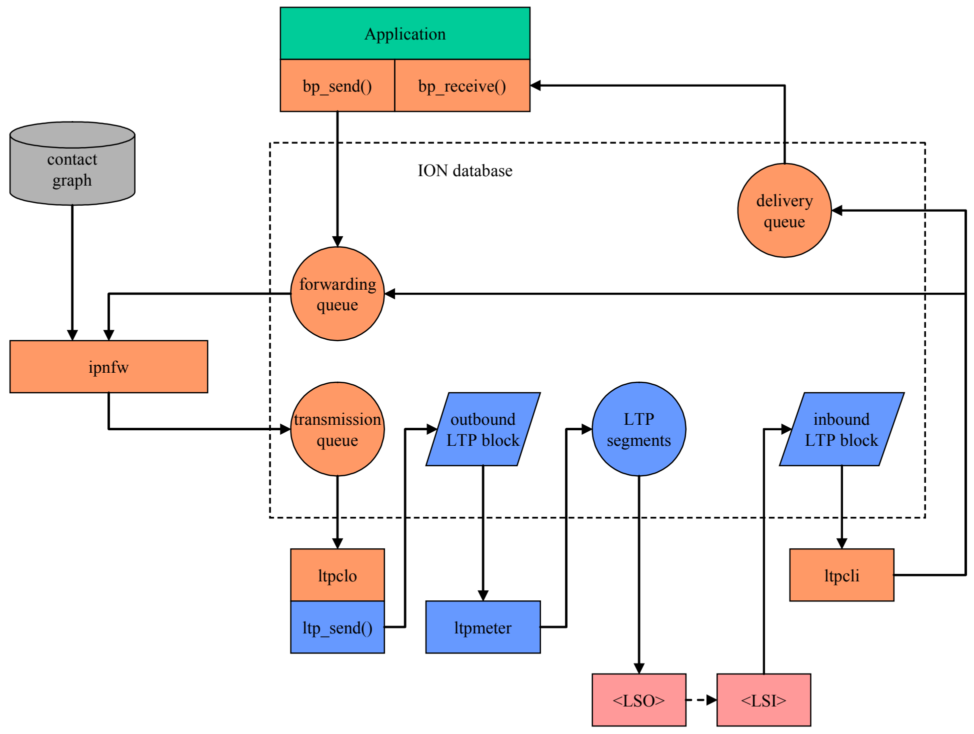

**Figure 1. Overview of distributed processing in ION.** *Applications invoke BP library functions, which access bundle queues in the database; so does the forwarding (route computation) daemon. BP "convergence-layer" adapter processes in turn invoke LTP library functions that access LTP blocks queues in the database, enabling them to bridge between BP and LTP in an open fashion. So does the LTP "meter" daemon, which provides segmentation services and flow control. One layer lower in the stack, LTP "link service" input and output adapter processes invoke low-level communication capabilities such as CCSDS packet transmission .*

## D. 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.

## V. 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.

### A. 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:

#### *1. 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.

#### *2. 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.

#### *3. 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 both. 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.

#### *4. 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.

#### *5. 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.

#### *6. 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.

#### *7. 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.

### B. Licklider Transmission Protocol (LTP)

The ION implementation of LTP conforms fully to the LTP specification as developed by the DTN Research Group, 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.

## C. 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 a system for computing dynamic routes through time-varying network topology assembled from scheduled, bounded communication opportunities. This system, called "Contact Graph Routing," is the subject of a separate SpaceOps 2008 paper.

In short, 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.

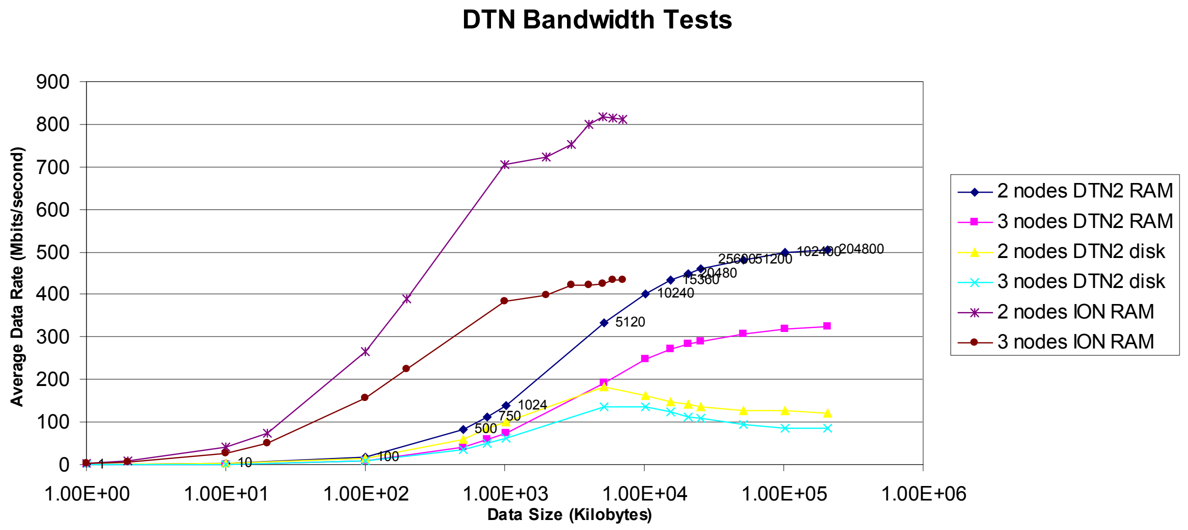

**Figure 2. Data transmission throughput measured for two implementations of Bundle Protocol.** *Average throughput for non-custodial transmission of bundles of various sizes was measured on a gigabit Ethernet.*

## VI. Test Results and Future Work

The results of preliminary performance benchmark tests of ION's implementation of Bundle Protocol versus the DTN2 reference implementation are shown in Fig. 2. For relatively small (less than 100KB) bundles such as we expect to be conveying in spacecraft operations, ION's measured throughput proved to be several times higher than that of DTN2. This appears to validate the optimization strategies underlying the ION design.

ION has reached a level of maturity that has enabled JPL to begin work on a flight demonstration of ION on the Deep Impact flyby spacecraft, currently planned for late in 2008. Additional development is planned, however:

- Implementations of the CCSDS File Delivery Protocol (CFDP) and Asynchronous Message Service (AMS) that utilize underlying ION communication capabilities are needed. An AMS implementation has been developed but not yet tested in flight. Work on an ION-enabled CFDP implementation has not yet started.
- Rough clock synchronization, on the order of one second, among all nodes in any DTN-based network is needed in order to assure the correct operation of the bundle expiration mechanism. A general solution to this problem in interplanetary space has yet to be devised.
- Security measures for communication in DTN-based networks have been designed and are approaching standardization. ION will need to include implementations of those standards.

## Acknowledgments

The research described in this paper was carried out at the Jet Propulsion Laboratory, California Institute of Technology, under a contract with the National Aeronautics and Space Administration.

## References

[1]Cerf, V., et al., "Delay-Tolerant Network Architecture," IETF RFC 4838, informational, April 2007, URL: http://www.ietf.org/rfc/rfc4838.txt.

[2]Burleigh, S., et. al., "Delay-Tolerant Networking – An Approach to Interplanetary Internet," *IEEE Communications Magazine*, Vol. 41, No. 6, 2003.

[3]Scott, K., Burleigh, S., "Bundle Protocol Specification," IETF RFC 5050, experimental, November 2007, URL: http://www.ietf.org/rfc/rfc5050.txt.

[4]Farrell, S., Cahill, V., Geraghty, D., Humphreys, I., and McDonald, P., "When TCP Breaks: Delay- and Disruption-Tolerant Networking," *IEEE Internet Computing*, Vol. 10, No. 4, 2006, pp. 72-78.