Architecture of ION Interplanetary Overlay Network

Scott Burleigh, IPNSIG

Architecture of ION

- *Interplanetary Overlay Network* (ION) is a DTN implementation that is specifically designed for use in resource-constrained embedded systems, such as interplanetary robotic spacecraft.
- ION development began in June 2003; first announced in November 2003.
- The first use of ION in space flight was during the Deep Impact Network (DINET) experiment in October-November 2008.
- ION's architecture is very different from that of other Bundle Protocol implementations. Why is that?

Target Environment

- The design of ION drew on experience from working with the Mars Pathfinder mission in 1996-1997.
- The platform for that mission was an early version of VxWorks running on a RAD6000 flight computer.
- ION was designed to enable DTN to run on that platform and successor deep-space flight mission platforms, such as the RAD750 flight computers used for the Mars Exploration Rovers and more advanced architectures used in the Mars Surface Laboratory (Curiosity) and the Perseverance rover.
- This required:
  - Conformance to several flight mission environment constraints.
  - Portability, to enable development on a developer-friendly platform followed by deployment **without modification** on flight mission platforms.

Flight Environment Constraints (1 of 3)

- Link constraints: wireless links enabling interplanetary network communication are generally slow and are usually asymmetric.
  - Limited electrical power, relatively small antennae.
    - So signals are weak. This limits transmission from the spacecraft to rates on the order of .25 Mbps to 6 Mbps.
    - Additionally, reception sensitivity is limited. Rates of transmission to the spacecraft are typically even lower, on the order of 1 or 2 Kbps.
  - So the cost per octet of data is on the links is high, and the links are heavily subscribed.
  - Economical use of reception and transmission opportunities is important.

Flight Environment Constraints (2 of 3)

- Processor constraints:
  - Limited electrical power, limited mass allowance.
  - Intense radiation environment, mandating radiation-hardening, which is time-consuming and expensive.
  - Relatively small market, limiting incentive to do radiation-hardening engineering for the latest advances in processor technology.
  - So flight processors are always slower than engineering workstations.
  - So the cost per processing cycle is high and the processors are heavily subscribed.
  - Economical use of processing resources is important.

Flight Environment Constraints (3 of 3)

- Hands-on repair is impossible, so reliability is key.
  - Predictability enhances reliability, so flight software usually must meet hard real-time deadlines. So real-time operating systems (RTOS) are used: all software runs in “kernel” (rather than “user”) mode, typically with no memory protection.
  - Dynamic allocation of system memory is difficult to predict, so it is typically **prohibited** except in certain well-understood spacecraft states, e.g., start-up.

Constraints on DTN Implementations in Space

	Terrestrial DTN	**DTN for Space Flight**
Links	Ethernet or WiFi Fast, cheap, symmetrical	Directed, highly attenuated Relatively slow, very expensive, asymmetrical Must use reception/transmission contacts efficiently.
CPU, memory	Commodity generic chips Fast, cheap	Limited-production radiation-hardened chips Relatively slow, very expensive Must use processing resources efficiently.
Resource management	Reboots are easy. Dynamic management of memory is routine.	Hands-on repair is impossible; must minimize risk. Dynamic memory management is unpredictable. Fixed memory allocation is provided at startup.
Operating System	Commercial O/S with memory protection; tasks run in user space.	Real-time O/S, normally no memory protection – all tasks run in kernel space. Must be RTOS-compatible.

ION's Design

- The design of ION uniquely addresses these constraints.
  - Built-in private dynamic management of memory allocated at startup.
  - High-speed shared direct access to built-in object database.
  - System-wide transaction mechanism, for safety:
    - Ensures mutual exclusion, preventing lockouts and race conditions.
    - Enables reversal of all database updates made within the current transaction in case of software failure.
  - Zero-copy objects, for processing and storage economy.
  - Written in C, for processing economy and (relatively) small footprint.
  - Portable among POSIX operating systems, including RTOS.
    - Currently available for Linux, Solaris, OS/X, Windows.
    - Support for FreeBSD, VxWorks, RTEMS, Android is not current but could be recovered quickly.

ION's Design goals

- Reliable conveyance of data over a delay-tolerant network
- Built on this capability: reliable data streaming, reliable file delivery, and reliable distribution of short messages to multiple recipients (subscribers) residing in such a network
- Inexpensive management of traffic through such a network
- Inexpensive facilities for monitoring the performance of the network
- Robustness against node failure
- Portability across heterogeneous computing platforms
- High speed with low overhead
- Easy integration with heterogeneous underlying communication infrastructure, ranging from Internet to dedicated spacecraft communication links

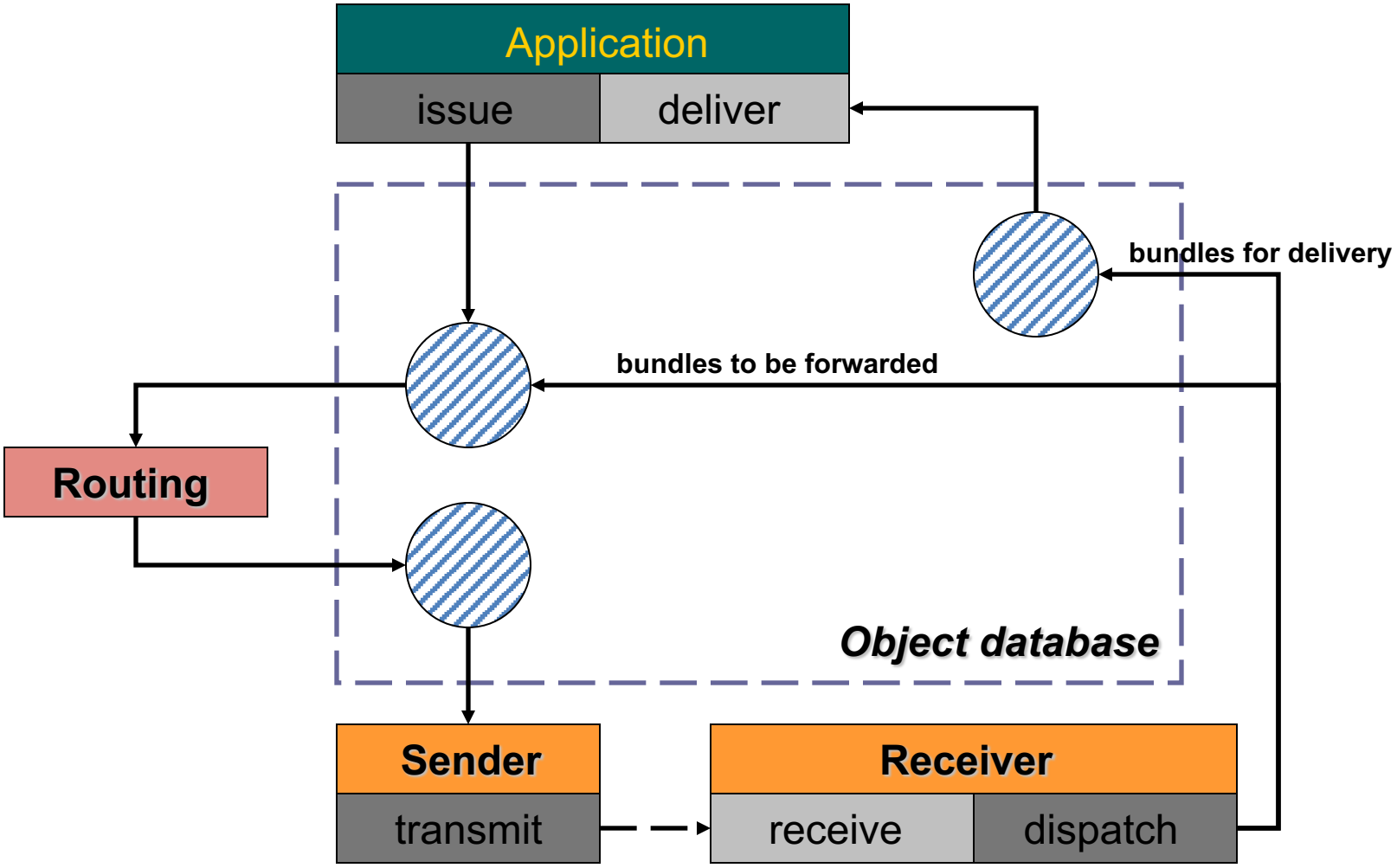
ION node architecture

- ION is database-centric rather that daemon-centric.
  - Each node is a single SDR database.
- Bundle protocol API is local functions in shared libraries, rather than inter-process communication channels.
- Multiple independent processes – daemons and applications, as peers – share direct access to the node state (database and shared memory) concurrently.

ION node architecture (cont'd)

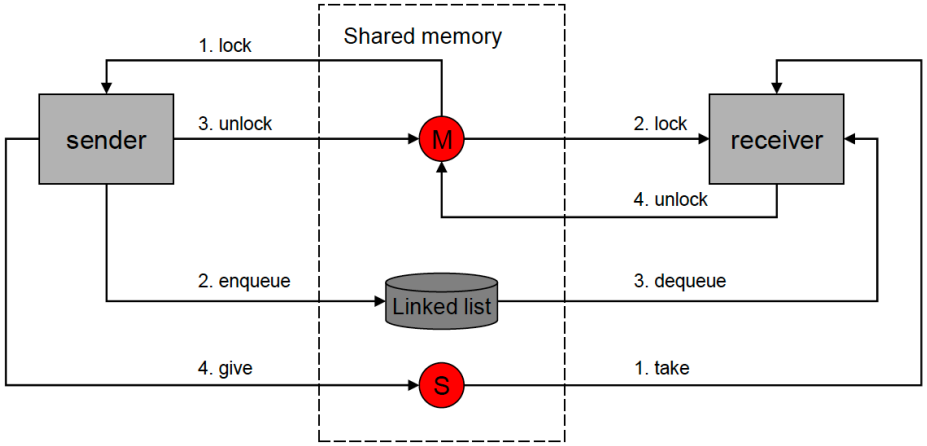
- Separate process for each scheme-specific forwarder.
  - Forwarder is tailored to the characteristics (endpoint naming, topology) of the environment implied by the scheme name.
- Separate process for each convergence-layer input and output.
  - No assumption of duplex connectivity.
- Schemes (forwarders) and convergence-layer adapter points can be added, stopped, reconfigured, restarted, removed while the node is running.

ION Design Overview



ION's Design Principles (1 of 2)

- Use shared memory.
  - Often there's no protected memory, so we have no option.
  - But this can be turned to advantage: shared memory is a highly efficient way to pass data between flight software tasks.



ION's Design Principles (2 of 2)

- Zero-copy procedures: leverage shared memory to minimize processing overhead.
  - Encapsulation in layers of protocol overhead (headers and trailers) can be done by reference rather than by copy.
  - The same data object can be shared by multiple tasks, provided reference counting prevents premature deletion.
- Portability: this is an unfamiliar programming model, so we must make it easy to develop in an environment with good programming support (e.g., Linux) and then deploy – without change – in the target RTOS environment

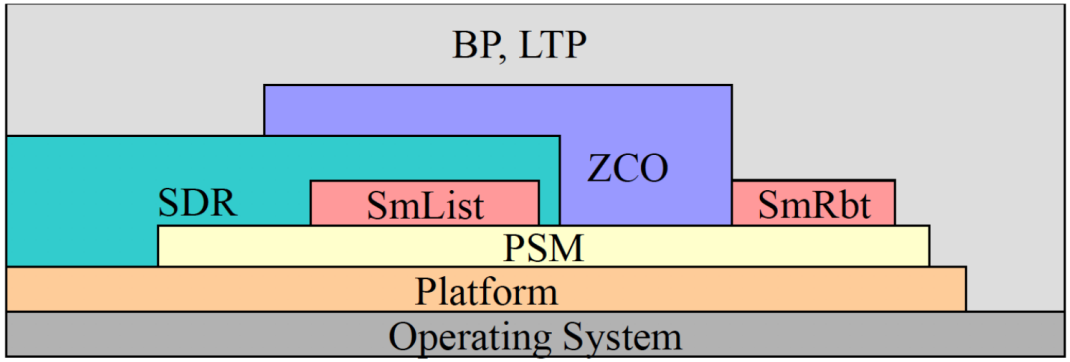
ION Modules Overview (1 of 2)

- The ION distribution comprises the following software modules:
  - bp (Bundle Protocol), a core DTN protocol that provides delay-tolerant forwarding of data through a network in which continuous end-to-end connectivity is never assured, including support for delay-tolerant dynamic routing.
  - ltp (Licklider Transmission Protocol), a core DTN protocol that provides transmission reliability based on delay-tolerant acknowledgments, timeouts, and retransmissions.
  - dgr (Datagram Retransmission), an alternative implementation of LTP that is designed for use in the Internet. It enables data to be transmitted via UDP with reliability comparable to that provided by TCP.
  - ici (Interplanetary Communication Infrastructure), a set of general-purpose libraries providing common functionality to the other modules.

ION Modules Overview (2 of 2)

- cfdp (CCSDS File Delivery Protocol), application-layer service which utilizes underlying DTN protocols for file transfer. CFDP performs the segmentation, transmission, reception, reassembly, and delivery of files in a delay-tolerant manner.
- ams (Asynchronous Message Service), an application-layer service which utilizes underlying DTN protocols for publication of short messages.
- bss (Bundle Streaming Service), a system for efficient data streaming over a delay-tolerant network. It includes:
  - a convergence-layer protocol (bssp) that preserves in-order arrival of all data that were never lost en route, yet ensures that all data arrive at the destination eventually, and
  - a library for building delay-tolerant streaming applications.

ION Software Structure



- BP, LTP Bundle Protocol and Licklider Transmission Protocol libraries and daemons
- ZCO Zero-copy objects capability: minimize data copying up and down the stack
- SDR Spacecraft Data Recorder: persistent object database in shared memory, using PSM and SmList
- SmList linked lists in shared memory using PSM
- SmRbt red-black trees in shared memory using PSM
- PSM Personal Space Management: memory management within a pre-allocated memory partition
- Platform common access to O.S.: shared memory, system time, IPC mechanisms
- Operating System: POSIX thread spawn/destroy, file system, time