# ION IMPLEMENTATION OF THE DTN ARCHITECTURE

# Day 3 Agenda – Afternoon

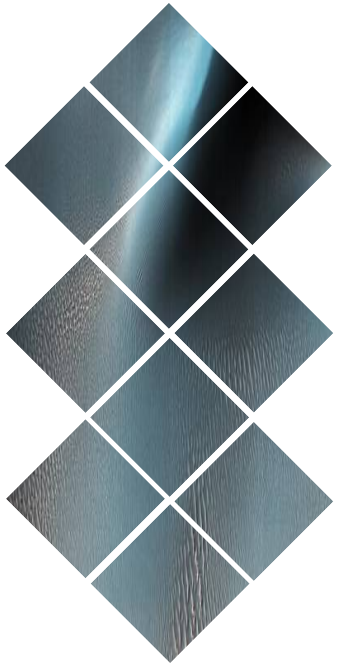
## Key Topics

- Configuration
- Recovery
- Security
- Tuning
- Troubleshooting
- Applications

# Reference materials

For more detail on the architecture, structure, functions, and modules of ION, see the “ION Design Guide” in the top-level directory of the ION download (from the SourceForge download site, https://sourceforge.net/projects/ion-dtn/).

For more detail on ION configuration, tuning, and troubleshooting, see the “ION Deployment Guide” in the same directory.



# Network Management

# Compile-time configuration (1 of 2)

ION can be extensively tailored for specific environments by means of compile-time switches. Details are provided in the Deployment Guide, but here is a partial summary:

- PRIVATE_SYMTAB – configure local symbol table
- FSWLOGGER – provide overriding message logging function for flight software
- FSWCLOCK – provide overriding time reporting function for flight software
- FSWWDNAME – provide overriding working directory name function for flight software
- FSWSYMTAB – provide overriding symbol table access function for flight software
- GDSLOGGER – provide overriding message logging function for ground software
- ION_OPS_ALLOC – overriding percentage of heap reserved for ION operations
- ION_OPS_MARGIN – overriding percentage of heap reserved for margin

# Compile-time configuration (2 of 2)

- HEAP_PTRS – enables heap object access by pointer
- NO_SDR_TRACE – disables heap space allocation (i.e., leak) tracing
- NO_PSM_TRACE – disables working memory allocation tracing
- IN_FLIGHT – disables core file production on unrecoverable error
- ERRMSGS_BUFSIZE – sets size of buffer for composing ION error messages
- SPACE_ORDER – indicates number of bits in a pointer
- NO_SDRMGT – limits SDR system to managing transactions
- DOS_PATH_DELIMITER – sets the character used as path delimeter
- ION_NOSTATS – disables the logging of bundle processing statistics
- CGR_DEBUG – controls logging of debug messages from CGR
- ION_BANDWIDTH_RESERVED – turns on starvation control in bundle transmission

# Runtime configuration (1 of 2)

Most of the parameters that configure an ION node are declared at the time the node is created: administration utility programs as listed below are passed configuration files as noted. See the *man* pages for these utility and file names for details.

- ionadmin(ionrc, ionconfig)
- ionsecadmin(ionsecrc)
- ltpadmin(ltprc)
- bpadmin(bprc)
- ipnadmin(ipnrc)
- dtn2admin(dtn2rc)
- cfdpadmin(cfdprc)
- bsspadmin(bssprc)
- dtpcadmin(dtpcrc)

# Runtime configuration (2 of 2)

Many node configuration parameters can also be revised after the node has been instantiated – in some cases, even while the node is running:

- BP convergence-layer elements can be stopped and restarted.
- BP endpoints can be added and removed.
- Security rules can be added and removed.
- Contacts can be added and removed.
- Etc.

# Multi-node Operation

It's possible to run multiple ION nodes on the same computer – even in the same core.

- A separate directory must be created for each node, containing all configuration files for that node.
- The ionconfig variables wmKey and sdrName must have different values for all nodes.
- Environment variable ION_NODE_LIST_DIR – giving the location of the ion_nodes file that indicates which nodes are resident in which directories – must be defined in the environment of every ION task.
- See the Deployment Guide for details.

# Network Management

A new, comprehensive system for network monitoring and control – superseding the ION administration utilities – has been developed and is included with ION 3.6.2.

It's called nm, an implementation of the emerging Asynchronous Management Protocol (AMP) standard.

However, graphical user interface support for nm is not yet complete and there is currently little documentation, so we won't cover it now.

nm will be discussed in a future edition of the ION course, or possibly an additional course.

# Transaction reversibility

**Every ION node can be individually configured for support of ION transaction reversibility:**

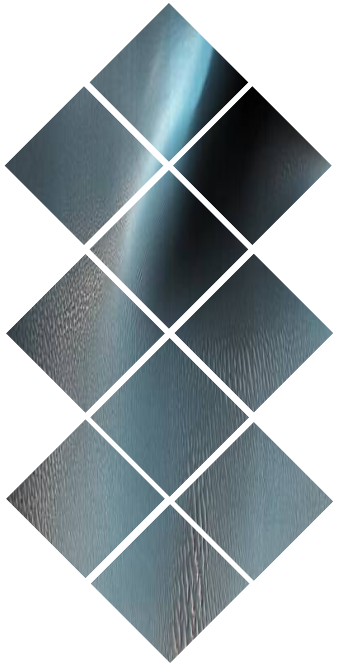
- The **configFlags** argument passed to ionadmin sets this switch.
- When transaction reversibility is disabled, any ION transaction failure or cancellation results in an "unrecoverable error" log message and (normally) a core dump. This is convenient during software development.
- When transaction reversibility is enabled, any ION transaction failure or cancellation causes all heap modification operations performed during the current transaction to be reversed and triggers execution of the **ionrestart** utility program. **ionrestart** automatically shuts down the node and restarts it, so that working memory is brought into agreement with heap state.
- Transaction reversibility also enables **recovery from an unplanned node shutdown**, as in a hardware power-on reset.

# Security

ION provides support for DTN security in multiple modules:

- A central security database management system, including key management, is included in the ici module (the ionsecadmin utility and ionsecrc configuration files). This general facility provides services to the security adaptations in other ION modules.
- Adaptations for some simple LTP security are included in the ltp module.
- Adaptations for bundle security are included in the bp module.
- Some simple authentication mechanisms are also built into the brs (Bundle Relay Server) convergence-layer protocol in the bp module.

Like network management, security in ION may be a large enough topic to merit its own separate course. We won't try to cover it here.



# Tuning

# What to tune for

**The fundamental purpose of ION is to maximize the rate at which important information is delivered to network user applications.**

ION's design itself helps to achieve that purpose, e.g.:

- The protocol designs attempt to minimize overhead that wastes bandwidth.
- Retransmission mechanisms attempt to minimize data loss.
- Routing mechanisms attempt to minimize end-to-end delivery latency.
- Rate control and congestion control attempt to maximize bandwidth utilization.
- "Blocking" resource allocation mechanisms exert indirect control on data rates.
- Bundle class of service enables preferential forwarding of more important data.
- "Critical" bundles are forwarded over all paths to the destination concurrently.

But operational configuration decisions can affect the rate of important information delivery in either positive or negative ways.

# Data triage (1 of 2)

When links are under-utilized, the rate of important delivery may not be as high as it might be – so ION attempts to convey as much data as possible during every contact opportunity.

This means, though, that sometimes more data will be presented to the links than they can convey.

In the Internet, data rates can be adjusted instantly, cooperatively, by means of flow control protocols.

But in DTN, in the general case, precise flow control is impossible because round-trip communication can take a long time. So ION must simply discard excess traffic – proactively when possible and reactively when necessary.

The key factor in these decisions is the bundle's Time To Live (TTL).

| Proactive data triage | Reactive data triage |
|---|---|
| Proactive data triage occurs when ION determines that it cannot compute a route that will deliver a given bundle to its final destination prior to expiration of the bundle's TTL. So a bundle may be peremptorily discarded because its TTL is too short, given the backlog of bundles awaiting transmission to the neighboring node that is first on the path to the destination. | The destruction of bundles due to TTL expiration prior to successful delivery to the final destination occurs when the network conveys bundles at lower net rates than were projected during route computation.<br>Example of causes: Contacts may be shortened by the configuration of ION itself, or high R/F interference or underestimated acknowledgment round-trip times may cause an unexpectedly high volume of retransmission traffic. |

# Data triage (2 of 2)

To tune an ION-based network, try to encourage a modicum of proactive data triage and as little reactive data triage as possible:

1. Estimate convergence-layer protocol overhead as accurately as possible in BP configuration.
2. Synchronize nodes' clocks as accurately as possible. Try to keep clock error as close to zero as possible.
3. Set LTP session limit and block size threshold generously to assure that LTP does not constrain data flow to rates below those supported by BP rate control.
4. Set ranges (one-way light times) and queuing delays as accurately as possible.
5. Communicate changes in configuration (especially contacts and ranges) to all nodes as far as possible in advance of the time they take effect.
6. Provide all nodes with as much storage capacity as possible for queues of bundles awaiting transmission.

# LTP tuning (1 of 2)

LTP configuration exerts a heavy influence on ION performance.

Unfortunately, tuning LTP in ION is something of a black art. There are a large number of configuration parameters, and the settings of those parameters can affect throughput in non-obvious ways.

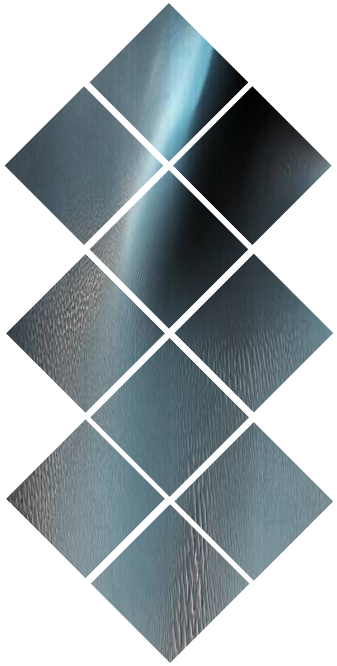
To help with this network management task, an Excel spreadsheet named ION-LTP-configuration.xls is provided. This spreadsheet constitutes a rough model of the characteristics of a single LTP "span", i.e., the relationship between a single pair of LTP engines (BP nodes): it indicates how changing the values of various span parameters will affect several operational figures of merit.

The use of the spreadsheet is itself less than obvious, so documentation is provided in a file named ION-LTP-configuration.pdf.

# LTP tuning (2 of 2)

In general:

- A large block size means a lot of LTP segments per session (good for a high-rate return, low-rate forward link situation).
- A small block size means the number of segments per session is smaller. LTP protocol will complete the block transfer more quickly because the number of segment retries is generally smaller.
- A good starting point is to set block aggregation size threshold to the number of bytes that will typically be transmitted in one second, so that blocks are typically clocked out about once per second. The maximum number of export sessions then should be at least the total number of seconds in the round-trip time for traffic on this LTP span, to prevent transmission from being blocked due to inability to start another session while waiting for the LTP acknowledgment that can end one of the current sessions.
- When bit error rates are high, LTP performance can be improved by reducing segment size and/or block aggregation size threshold.



# Troubleshooting

# What could possibly go wrong?

ION is designed to fit within a tight set of constraints, over a very wide range of deployment environments.

To make this possible, ION operates within a large number of variable configuration parameters that make it adaptable on many dimensions.

This makes instantiating a new ION-based network almost always possible, but often difficult. It's easy to make a mistake.

The good news is that once you've got ION running well in your environment, it should provide good service indefinitely without requiring much attention.

# The ion.log file

Whenever anything weird happens while you are using ION:

**Always, always look at the ion.log file first.**

The time-tagged messages printed in the ion.log file will very often give you at least a clue as to what failed.

# “Watch” characters (1 of 2)

Several ION modules can be configured to print “watch” characters to standard output (your console) as various significant events in DTN traffic flow occur.

The stream of watch characters appearing in stdout can give you a succinct, coarse, but comprehensive trace of activity in the nodes of your network.

For details, see the man pages for bprc, ltprc, cfdprc, bssprc, etc. Keith will present an example in the next hands-on segment of the class.

# “Watch” characters (2 of 2)

BP Watch Characters:

- a new bundle is queued for forwarding
- b bundle is queued for transmission
- c bundle is popped from its transmission queue
- m custody acceptance signal is received
- w custody of bundle is accepted
- x custody of bundle is refused
- y bundle is accepted upon arrival
- z bundle is queued for delivery to an application
- ~ bundle is abandoned (discarded) on attempt to forward it
- ! bundle is destroyed due to TTL expiration
- & custody refusal signal is received
- # bundle is queued for re-forwarding due to CL protocol failure
- j bundle is placed in "limbo" for possible future re-forwarding
- k bundle is removed from "limbo" and queued for re-forwarding
- $ bundle's custodial retransmission timeout interval expired

LTP Watch Characters

- d bundle appended to block for next session
- e segment of block is queued for transmission
- f block has been fully segmented for transmission
- g segment popped from transmission queue
- h positive ACK received for block, session ended
- s segment received
- t block has been fully received
- @ negative ACK received for block, segments retransmitted
- = unacknowledged checkpoint was retransmitted
- + unacknowledged report segment was retransmitted
- { export session canceled locally (by sender)
- } import session canceled by remote sender
- [ import session canceled locally (by receiver)
- ] export session canceled by remote receiver

# sdrwatch, psmwatch

ION's SDR heap storage and working memory are monolithic memory partitions for which dynamic allocation is privately managed by the SDR and PSM systems.

This means that industry standard memory debugging tools like *valgrind* have no visibility into ION memory management. Yet we sometimes still need to debug memory utilization.

This is why the *sptrace* module was developed for ION. sptrace tracks all memory allocation and freeing for a single memory partition managed by SDR or PSM, enabling detection of **leaks** and other **memory management errors**.

The *sdrwatch* and *psmwatch* utilities are the developers' tools for examining the data collected by sptrace. They are not especially slick, but they do work. See the man pages for details on how they can be used.

# Other troubleshooting tools

When you are developing software that uses or extends ION, you may sometimes encounter errors that you can't understand and correct simply by looking at error messages and watch characters.

ION is developed exclusively on Linux machines because historically Unix-derived operating systems have provided the most powerful debugging tools. That may be changing, but for now ION development is still mostly old-school.

To make life easy on yourself, develop on a Linux box and become familiar with the sorcerous genius of *gdb*.

# “Wrong profile for this SDR”

While you are experimenting with instantiating new DTN nodes, ***always erase your previously created node completely before trying to create a new one***.

ION will retain all of the state of that old node until you explicitly erase it. If the old node isn’t completely erased, either you will pick up the old node state or else your attempt to re-create the new node will fail because its parameters conflict with the old configuration of that node.

To erase the previously created node completely:

- Do a clean shutdown if possible, using “ionstop” or a similar script. These scripts use ION admin utilities to cleanly stop all daemons.
- Run the “killm” script. This script will try to kill any leftover running daemons, in the event that your node ended by an unrecoverable error rather than a clean shutdown.
- Run the “ipcs” command to make sure that all shared-memory semaphores have been destroyed by killm.
- If any shared-memory semaphores remain, use “ps” to locate leftover running ION daemons and use “kill -9” to terminate them, then try again.

# “Can’t find ION security database”

This message is just a warning, but it’s annoying.

To make it go away, always pass the “1” command to ionsecadmin during instantiation of any new node.

# LAN firewalls

If you find that no data can flow between DTN nodes over your local area network (using TCP/IP or UDP/IP at the convergence layer or LTP link service layer), make sure you don't have a LAN communication problem:

- Might a firewall setting be preventing traffic between the LTP or UDP ports you're using?
- Might the be a divergence in DNS name resolution due to a difference in /etc/host files?

# Clock disagreement

Remember that BP relies on time-to-live expiration to clear undelivered bundles out of the network, releasing buffer space for more traffic.

This means that if the clocks of two nodes don't report the same time, bundles issued by one node with a given expected expiration time might be deleted immediately upon reception by the other node – which thinks that expiration time has already passed.

Possible ways that clocks can disagree:

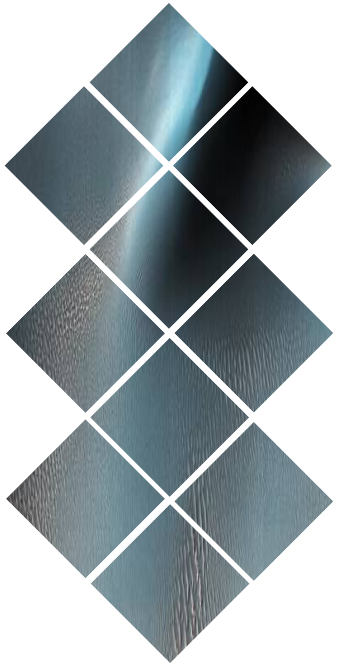
- Clocks don't always keep time accurately. The time reported by a clock may drift.
- One machine's clock might be on the UTC time scale while the other's is on local time.
- The two machines' clocks might be configured for different time zones.
- One machine's clock might be adjusted for Daylight Savings Time while the other's is not.

# Missing ranges

ION nodes typically use Contact Graph Routing to compute routes to bundles' destinations and thereby determine which neighboring nodes to forward those bundles to.

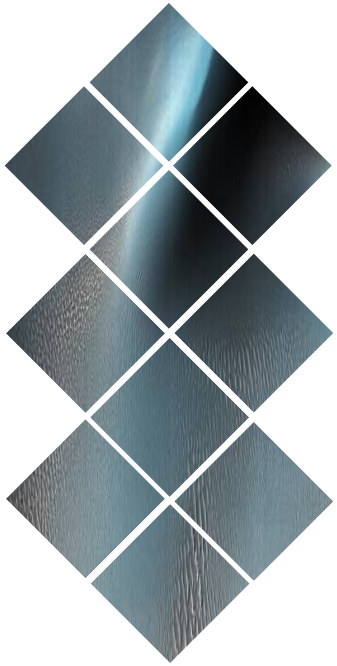
But CGR can't compute routes if information is missing in the contact plans. It needs not only the anticipated contacts but also all ranges (that is, distances expressed in light seconds) between nodes – because the time required to travel from one node to the next is a key factor in selecting between contacts while computing a route.

- Note that ION doesn't expect the range between nodes to be symmetrical: that is, it's fine for a bundle sent from node A to reach node B after 3 seconds but for the reply from B to A to take 11 seconds (due to additional queuing delay at B, for example).
- When a range command cites two node numbers A and B where B (the second) is *numerically larger* than A, the same value is implied for the reverse direction (B to A) unless overridden.
- A range command citing node numbers A and B where B (the second) is *numerically smaller* than A overrides the default assumption. It does not imply anything about the B→A range!



# Applications

- CFDP
- AMS
- Bundle Streaming Service
- DTPC



# CCSDS FILE DELIVERY PROTOCOL (CFDP)

# CCSDS File Delivery Protocol (CFDP)

The ION implementation of CFDP offers support for the following standard capabilities:

- Segmentation of files on user-specified record boundaries
- Transmission of file segments in protocol data units that are conveyed by an underlying Unitdata Transfer service
- Reassembly of files from received segments
- User-specified fault handling procedures
- Operations on remote file systems

**Remember!** All CFDP transaction state is safely retained in the ION heap for rapid recovery from a spacecraft or software fault.

# CCSDS File Delivery Protocol (CFDP)

## *Compile time options*

Defining the following macro, by setting a parameter that is provided to the C compiler, will alter the functionality of CFDP.

**TargetFFS:** Setting this option adapts CFDP for use with the TargetFFS flash file system on the VxWorks operating system. TargetFFS locks one or more system semaphores so long as a file is kept open.



# CCSDS File Delivery Protocol (CFDP)

## *Building CFDP*

1. The “bp” module has to be built for the platform on which CFDP will run.
2. Edit the Makefile in ion/cfdp:
   - Make sure PLATFORMS is set to the name of platform on which you plan to run BP.
   - Set OPT to the directory containing the bin, lib, include, etc. directories used for building ici.
3. Then:
   - cd ion/cfdp
   - make
   - make install

# CCSDS File Delivery Protocol (CFDP)

## *Running CFDP*

**The executable programs used in operation of the CFDP component of ION include:**

- **The cfdpadmin protocol configuration utility, invoked at node startup time and as needed thereafter.**
- **The cfdpclock background daemon, which effects scheduled CFDP events such as check timer expirations. The cfdpclock task also effects CFDP transaction cancellations, by canceling the bundles encapsulating the transaction's protocol data units.**
- **The bputa UT-layer input/output task, which handles transmission of CFDP PDUs encapsulated in bundles.**



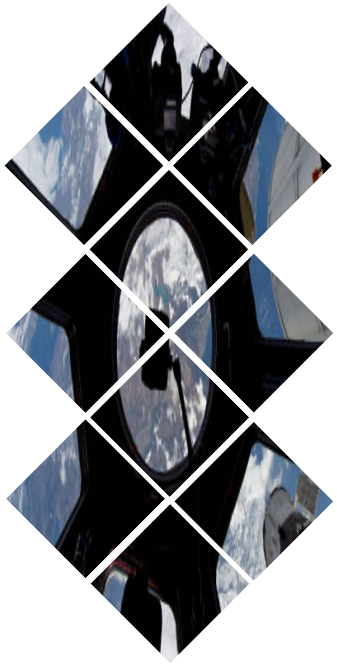
# CCSDS File Delivery Protocol (CFDP)

### *Testing CFDP*

### *Test executables*

- **cfdptest** is provided to support testing and debugging of the DGR component of ION.

**Remember:** The CFDP administration command (cfdprc) file provides the information needed to configure CFDP on a given ION node.



# THE ASYNCHRONOUS MESSAGE SERVICE (AMS)

# Asynchronous Message Service (AMS)

AMS is a **data system communications architecture** where communication relationships among modules are self-configuring → minimized complexity in the development and operations of modular data systems.

The **purpose** of AMS is **to reduce mission cost and risk** by providing standard, reusable infrastructure for the exchange of information among data system modules in a manner that is simple to use, highly automated, flexible, robust, scalable, and efficient.

**Remember!** The CCSDS AMS standard conforms fully to CCSDS 735.0-B-1

# Asynchronous Message Service (AMS)

*Compile-time options*

Defining the following macros will alter the functionality of AMS.

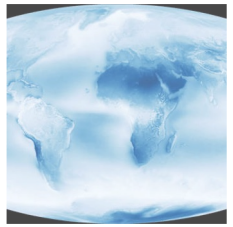
**NOEXPAT**: Setting this option adapts AMS to expect MIB information to be presented to it in "amsrc" syntax rather than in XML syntax, normally because the expat XML interpretation system is not installed. The default for AMS is now **NOEXPAT**.

**AMS_INDUSTRIAL**: Setting this option adapts AMS to an "industrial" rather than safety-critical model for memory management. By default, the memory acquired for message transmission and reception buffers in AMS is allocated from limited ION working memory, which is fixed at ION start-up time => this limits the rate at which AMS messages may be originated and acquired. When **–DAMS_INDUSTRIAL** is set at compile time, the memory acquired for message transmission and reception buffers in AMS is allocated from system memory, using the familiar **malloc()** and **free()** functions; this enables much higher message traffic rates on machines with abundant system memory.

# Asynchronous Message Service (AMS)

## *Building AMS*

1. The “bp” component has to be built for the platform on which AMS runs.
2. Edit the Makefile in ion/cfdp:
   - Just as for bp, make sure PLATFORMS is set to the name of platform on which you plan to run AMS.
   - Set OPT to the directory containing the bin, lib, include, etc. directories used for building bp.
3. Then:
   - cd ion/ams
   - make
   - make install



# Asynchronous Message Service (AMS)

## *Running AMS*

**The executable programs used in operation of the AMS component of ION include:**

- The **amsd** background daemon serves as configuration server and/or as the registrar for a single application cell.
- The **ramsgate** application module serves as the Remote AMS gateway for a single message space.
- The **amsstop** utility terminates all AMS operation throughout a single message space.
- The **amsmib** utility announces supplementary MIB information to selected subsets of AMS entities without interrupting the operation of the message space.

# Asynchronous Message Service (AMS)

## *Testing AMS*

### *Test executables*

- **amsbenchs** is a continuous source of messages.
- **amsbenchr** is a message receiver that calculates bundle transmission performance statistics.
- **amshello** is an AMS "hello, world" demo program.
- **amsshell** is a console-like application for interactively publishing, sending, and announcing text strings in messages.
- **amslog** is a console-like application for receiving messages and piping their contents to stdout.
- **amslogprt** is a pipeline program that simply prints AMS message contents piped to it from amslog.
- **amspubsub** is a pair of functions for rudimentary testing of AMS functionality in a VxWorks environment.

# THE BUNDLE STREAMING SERVICE (BSS)

# Bundle Streaming Service (BSS)

The BSS service provided in ION enables continuously generated application data units (e.g.: stream of video) to be presented to a destination application in two modes concurrently:

- In the order in which the data units were generated, with the least possible end-to-end delivery latency, but possibly with some gaps due to transient data loss or corruption.
- In the order in which the data units were generated, without gaps but in a non-real-time “playback” mode.

# Bundle Streaming Service (BSS)

## *Compile-time options*

Defining the following macro, by setting a parameter that is provided to the C compiler (e.g., –DWINDOW=10000), will alter the functionality of BSS.

**WINDOW=xx**: Setting this option changes the maximum number of seconds by which the BSS database for a BSS application may be "rewound" for replay. The default value is 86400 seconds (24 hours).

# Bundle Streaming Service (BSS)

## *Building BSS*

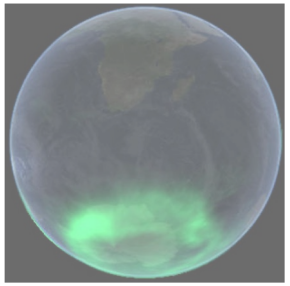
1. The “bp” component of ION has to be built for the platform on which BSS will run.
2. Edit the Makefile in ion/bss:
   - As for ici, make sure PLATFORMS is set to the name of platform on which you plan to run BSS.
   - Set OPT to the directory containing the bin, lib, include, etc. directories used for building ici.
3. Then:
   - cd ion/bss
   - make
   - make install

# Bundle Streaming Service (BSS)

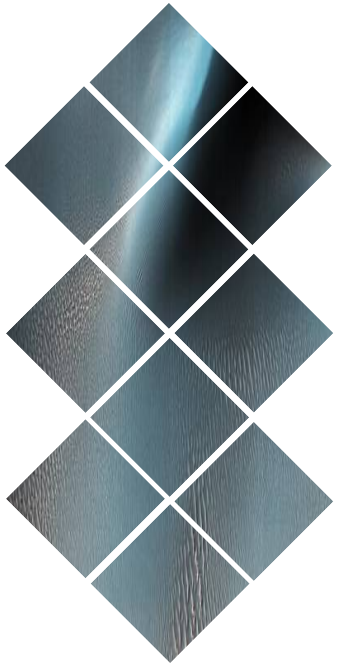
*Testing BSS on a development platform*

*Test executables*

- **bssdriver** sends a stream of data to **bsscounter** for non-interactive testing.
- **bssStreamingApp** sends a stream of data to **bssrecv** for graphical, interactive testing.



**Remember!** No additional configuration files or runtime executables are required for the operation of the BSS component of ION.



# Delay-Tolerant Payload Conditioning (DTPC)

# Delay-Tolerant Payload Conditioning (DTPC)

The DTPC service provided in ION is an application service protocol that offers delay-tolerant support for several end-to-end services to applications that may require them:

- Delivery of application data items in transmission (rather than reception) order
- Detection of reception gaps in the sequence of transmitted application data items, with end-to-end negative acknowledgment of the missing data
- End-to-end positive acknowledgment of successfully received data
- End-to-end retransmission of missing data, driven either by negative acknowledgment or timer expiration
- Suppression of duplicate application data items
- Aggregation of small application data items into large bundle payloads, to reduce bundle protocol overhead
- Application-controlled elision of redundant data items in aggregated payloads, to improve link utilization

**NOTE**, though, that these are **not really delay-tolerant services**! If your communication round-trip times are long, data delivery performance will suffer if you use DTPC.

# Delay-Tolerant Payload Conditioning (DTPC)

## *Building DTPC*

1. The “bp” component of ION has to be built for the platform on which DTPC will run.
2. Edit the Makefile in ion/dtpc:
   - As for ici, make sure PLATFORMS is set to the name of platform on which you plan to run DTPC.
   - Set OPT to the directory containing the bin, lib, include, etc. directories used for building ici.
3. Then:
   - cd ion/dtpc
   - make
   - make install

# Delay-Tolerant Payload Conditioning (DTPC)

*Testing dtpc on a development platform*

*Test executables*

- **dtpcsend** sends DTPC data items to **dtpcreceive** for performance testing.

**Remember!** No additional configuration files or runtime executables are required for the operation of the DTPC component of ION.

DTN
DELAY/DISRUPTION TOLERANT NETWORKING

## Topics discussed this afternoon:

| Theory |
| --- |
| Configuration |
| Recovery |
| Security |
| Tuning |
| Troubleshooting |
| Applications |



# Any questions?

# Thank you!

# Happy networking!