Stream: Internet Engineering Task Force (IETF)
RFC: 0000
Category: Standards Track
Published: October 2021
ISSN: 2070-1721
Authors:
S. Burleigh
JPL, Calif. Inst. Of Technology
K. Fall
Roland Computing Services
E. Birrane
APL, Johns Hopkins University

RFC 0000
Bundle Protocol Version 7

Abstract

This document presents a specification for the Bundle Protocol, adapted from the experimental Bundle Protocol specification developed by the Delay-Tolerant Networking Research Group of the Internet Research Task Force and documented in RFC 5050.

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

Since the publication of the Bundle Protocol Specification (Experimental RFC 5050 [RFC5050]) in 2007, the Delay-Tolerant Networking (DTN) Bundle Protocol has been implemented in multiple programming languages and deployed to a wide variety of computing platforms. This implementation and deployment experience has identified opportunities for making the protocol simpler, more capable, and easier to use. The present document, standardizing the Bundle Protocol (BP), is adapted from RFC 5050 in that context, reflecting lessons learned. Significant changes from the Bundle Protocol specification defined in RFC 5050 are listed in Appendix A.

This document describes version 7 of BP.

Delay-Tolerant Networking is a network architecture providing communications in and/or through highly stressed environments. Stressed networking environments include those with intermittent connectivity, large and/or variable delays, and high bit error rates. To provide its services, BP may be viewed as sitting at the application layer of some number of constituent networks, forming a store-carry-forward overlay network. Key capabilities of BP include:

- Ability to use physical motility for the movement of data
- Ability to move the responsibility for error control from one node to another
- Ability to cope with intermittent connectivity, including cases where the sender and receiver are not concurrently present in the network
- Ability to take advantage of scheduled, predicted, and opportunistic connectivity, whether bidirectional or unidirectional, in addition to continuous connectivity

- Late binding of overlay network endpoint identifiers to underlying constituent network addresses

For descriptions of these capabilities and the rationale for the DTN architecture, see [ARCH] and [SIGC].

BP's location within the standard protocol stack is as shown in Figure 1. BP uses underlying "native" transport and/or network protocols for communications within a given constituent network. The layer at which those underlying protocols are located is here termed the "convergence layer" and the interface between the bundle protocol and a specific underlying protocol is termed a "convergence layer adapter".

Figure 1 shows three distinct transport and network protocols (denoted T1/N1, T2/N2, and T3/N3).

```
+-----------+                                         +-----------+
|   BP app  |                                         |   BP app  |
+---------v-|   +->>>>>>>>>>v-+     +->>>>>>>>>>v-+   +-^---------+
|   BP    v |   | ^    BP   v |     | ^    BP   v |   | ^   BP    |
+---------v-+   +-^---------v-+     +-^---------v-+   +-^---------+
| T1      v |   + ^  T1/T2  v |     + ^  T2/T3  v |   | ^ T3      |
+---------v-+   +-^---------v-+     +-^---------v +   +-^---------+
| N1      v |   | ^  N1/N2  v |     | ^  N2/N3  v |   | ^ N3      |
+---------v-+   +-^---------v +     +-^---------v-+   +-^---------+
|         >>>>>>>>^         >>>>>>>>>>^         >>>>>>>>^         |
+-----------+   +-------------+     +-------------+   +-----------+
|                     |                     |                     |
|<---- A network ---->|                     |<---- A network ---->|
|                     |                     |                     |
```

Figure 1: The Bundle Protocol in the Protocol Stack Model

This document describes the format of the protocol data units (called "bundles") passed between entities participating in BP communications.

The entities are referred to as "bundle nodes". This document does not address:

- Operations in the convergence layer adapters that bundle nodes use to transport data through specific types of internets. (However, the document does discuss the services that must be provided by each adapter at the convergence layer.)
- The bundle route computation algorithm.
- Mechanisms for populating the routing or forwarding information bases of bundle nodes.
- The mechanisms for securing bundles en route.
- The mechanisms for managing bundle nodes.

Note that implementations of the specification presented in this document will not be interoperable with implementations of RFC 5050.

2. Conventions used in this document

The key words "**MUST**", "**MUST NOT**", "**REQUIRED**", "**SHALL**", "**SHALL NOT**", "**SHOULD**", "**SHOULD NOT**", "**RECOMMENDED**", "**NOT RECOMMENDED**", "**MAY**", and "**OPTIONAL**" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

3. Service Description

3.1. Definitions

Bundle - A bundle is a protocol data unit of BP, so named because negotiation of the parameters of a data exchange may be impractical in a delay-tolerant network: it is often better practice to "bundle" with a unit of application data all metadata that might be needed in order to make the data immediately usable when delivered to the application. Each bundle comprises a sequence of two or more "blocks" of protocol data, which serve various purposes.

Block - A bundle protocol block is one of the protocol data structures that together constitute a well-formed bundle.

Application Data Unit (ADU) - An application data unit is the unit of data whose conveyance to the bundle's destination is the purpose for the transmission of some bundle that is not a fragment (as defined below).

Bundle payload - A bundle payload (or simply "payload") is the content of the bundle's payload block. The terms "bundle content", "bundle payload", and "payload" are used interchangeably in this document. For a bundle that is not a fragment (as defined below), the payload is an application data unit.

Partial payload - A partial payload is a payload that comprises either the first N bytes or the last N bytes of some other payload of length M, such that $0 < N < M$. Note that every partial payload is a payload and therefore can be further subdivided into partial payloads.

Fragment - A fragment, a.k.a. "fragmentary bundle", is a bundle whose payload block contains a partial payload.

Bundle node - A bundle node (or, in the context of this document, simply a "node") is any entity that can send and/or receive bundles. Each bundle node has three conceptual components, defined below, as shown in Figure 2: a "bundle protocol agent", a set of zero or more "convergence layer adapters", and an "application agent". ("CL1 PDUs" are the PDUs of the convergence-layer protocol used in network 1.)

```
+-----------------------------------------------------------+
|Node                                                       |
|                                                           |
| +-------------------------------------------------------+ |
| |Application Agent                                      | |
| |                                                       | |
| | +--------------------------+ +----------------------+ | |
| | |Administrative element    | |Application-specific  | | |
| | |                          | |element               | | |
| | |                          | |                      | | |
| | +--------------------------+ +----------------------+ | |
| |                ^                          ^           | |
| |           Admin|records        Application|data       | |
| |                |                          |           | |
| +----------------v--------------------------v-----------+ |
|                               ^                           |
|                               | ADUs                      |
|                               |                           |
| +-----------------------------v-------------------------+ |
| |Bundle Protocol Agent                                  | |
| |                                                       | |
| |                                                       | |
| +-------------------------------------------------------+ |
|        ^                 ^                        ^       |
|        | Bundles         | Bundles        Bundles |       |
|        |                 |                        |       |
| +------v-----+     +-----v------+          +-----v-----+ |
| |CLA 1       |     |CLA 2       |          |CLA n      | |
| |            |     |            |   . . .  |           | |
| |            |     |            |          |           | |
+-+------------+-----+------------+----------+-----------+-+
         ^                 ^                        ^
      CL1|PDUs          CL2|PDUs                 CLn|PDUs
         |                 |                        |
  +------v-----+     +-----v------+          +-----v-----+
   Network 1          Network 2                Network n
```

Figure 2: Components of a Bundle Node

Bundle protocol agent - The bundle protocol agent (BPA) of a node is the node component that offers the BP services and executes the procedures of the bundle protocol.

Convergence layer adapter - A convergence layer adapter (CLA) is a node component that sends and receives bundles on behalf of the BPA, utilizing the services of some "native" protocol stack that is supported in one of the networks within which the node is functionally located.

Application agent - The application agent (AA) of a node is the node component that utilizes the BP services to effect communication for some user purpose. The application agent in turn has two elements, an administrative element and an application-specific element.

Application-specific element - The application-specific element of an AA is the node component that constructs, requests transmission of, accepts delivery of, and processes units of user application data.

Administrative element - The administrative element of an AA is the node component that constructs and requests transmission of administrative records (defined below), including status reports, and accepts delivery of and processes any administrative records that the node receives.

Administrative record - A BP administrative record is an application data unit that is exchanged between the administrative elements of nodes' application agents for some BP administrative purpose. The only administrative record defined in this specification is the status report, discussed later.

Bundle endpoint - A bundle endpoint (or simply "endpoint") is a set of zero or more bundle nodes that all identify themselves for BP purposes by some common identifier, called a "bundle endpoint ID" (or, in this document, simply "endpoint ID"; endpoint IDs are described in detail in Section 4.2.5.1 below).

Singleton endpoint - A singleton endpoint is an endpoint that always contains exactly one member.

Registration - A registration is the state machine characterizing a given node's membership in a given endpoint. Any single registration has an associated delivery failure action as defined below and must at any time be in one of two states: Active or Passive. Registrations are local; information about a node's registrations is not expected to be available at other nodes, and the Bundle Protocol does not include a mechanism for distributing information about registrations.

Delivery - A bundle is considered to have been delivered at a node subject to a registration as soon as the application data unit that is the payload of the bundle, together with any relevant metadata (an implementation matter), has been presented to the node's application agent in a manner consistent with the state of that registration.

Deliverability - A bundle is considered "deliverable" subject to a registration if and only if (a) the bundle's destination endpoint is the endpoint with which the registration is associated, (b) the bundle has not yet been delivered subject to this registration, and (c) the bundle has not yet been "abandoned" (as defined below) subject to this registration.

Abandonment - To abandon a bundle subject to some registration is to assert that the bundle is not deliverable subject to that registration.

Delivery failure action - The delivery failure action of a registration is the action that is to be taken when a bundle that is "deliverable" subject to that registration is received at a time when the registration is in the Passive state.

Destination - The destination of a bundle is the endpoint comprising the node(s) at which the bundle is to be delivered (as defined above).

Transmission - A transmission is an attempt by a node's BPA to cause copies of a bundle to be delivered to one or more of the nodes that are members of some endpoint (the bundle's destination) in response to a transmission request issued by the node's application agent.

Forwarding - To forward a bundle to a node is to invoke the services of one or more CLAs in a sustained effort to cause a copy of the bundle to be received by that node.

Discarding - To discard a bundle is to cease all operations on the bundle and functionally erase all references to it. The specific procedures by which this is accomplished are an implementation matter.

Retention constraint - A retention constraint is an element of the state of a bundle that prevents the bundle from being discarded. That is, a bundle cannot be discarded while it has any retention constraints.

Deletion - To delete a bundle is to remove unconditionally all of the bundle's retention constraints, enabling the bundle to be discarded.

3.2. Discussion of BP concepts

Multiple instances of the same bundle (the same unit of DTN protocol data) might exist concurrently in different parts of a network -- possibly differing in some blocks -- in the memory local to one or more bundle nodes and/or in transit between nodes. In the context of the operation of a bundle node, a bundle is an instance (copy), in that node's local memory, of some bundle that is in the network.

The payload for a bundle forwarded in response to a bundle transmission request is the application data unit whose location is provided as a parameter to that request. The payload for a bundle forwarded in response to reception of a bundle is the payload of the received bundle.

In the most familiar case, a bundle node is instantiated as a single process running on a general-purpose computer, but in general the definition is meant to be broader: a bundle node might alternatively be a thread, an object in an object-oriented operating system, a special-purpose hardware device, etc.

The manner in which the functions of the BPA are performed is wholly an implementation matter. For example, BPA functionality might be coded into each node individually; it might be implemented as a shared library that is used in common by any number of bundle nodes on a single computer; it might be implemented as a daemon whose services are invoked via inter-process or network communication by any number of bundle nodes on one or more computers; it might be implemented in hardware.

Every CLA implements its own thin layer of protocol, interposed between BP and the (usually "top") protocol(s) of the underlying native protocol stack; this "CL protocol" may only serve to multiplex and de-multiplex bundles to and from the underlying native protocol, or it may offer additional CL-specific functionality. The manner in which a CLA sends and receives bundles, as well as the definitions of CLAs and CL protocols, are beyond the scope of this specification.

Note that the administrative element of a node's application agent may itself, in some cases, function as a convergence-layer adapter. That is, outgoing bundles may be "tunneled" through encapsulating bundles:

- An outgoing bundle constitutes a byte array. This byte array may, like any other, be presented to the bundle protocol agent as an application data unit that is to be transmitted to some endpoint.
- The original bundle thus forms the payload of an encapsulating bundle that is forwarded using some other convergence-layer protocol(s).
- When the encapsulating bundle is received, its payload is delivered to the peer application agent administrative element, which then instructs the bundle protocol agent to dispatch that original bundle in the usual way.

The purposes for which this technique may be useful (such as cross-domain security) are beyond the scope of this specification.

The only interface between the BPA and the application-specific element of the AA is the BP service interface. But between the BPA and the administrative element of the AA there is a (conceptual) private control interface in addition to the BP service interface. This private control interface enables the BPA and the administrative element of the AA to direct each other to take action under specific circumstances.

In the case of a node that serves simply as a BP "router", the AA may have no application-specific element at all. The application-specific elements of other nodes' AAs may perform arbitrarily complex application functions, perhaps even offering multiplexed DTN communication services to a number of other applications. As with the BPA, the manner in which the AA performs its functions is wholly an implementation matter.

Singletons are the most familiar sort of endpoint, but in general the endpoint notion is meant to be broader. For example, the nodes in a sensor network might constitute a set of bundle nodes that are all registered in a single common endpoint and will all receive any data delivered at that endpoint. **Note** too that any given bundle node might be registered in multiple bundle endpoints and receive all data delivered at each of those endpoints.

Recall that every node, by definition, includes an application agent which in turn includes an administrative element, which exchanges administrative records with the administrative elements of other nodes. As such, every node is permanently, structurally registered in the singleton endpoint at which administrative records received from other nodes are delivered. Registration in no other endpoint can ever be assumed to be permanent. This endpoint, termed the node's "administrative endpoint", is therefore uniquely and permanently associated with the node, and for this reason the ID of a node's administrative endpoint additionally serves as the "node ID" (see Section 4.2.5.2 below) of the node.

The destination of every bundle is an endpoint, which may or may not be singleton. The source of every bundle is a node, identified by node ID. Note, though, that the source node ID asserted in a given bundle may be the null endpoint ID (as described later) rather than the ID of the source node; bundles for which the asserted source node ID is the null endpoint ID are termed "anonymous" bundles.

Any number of transmissions may be concurrently undertaken by the bundle protocol agent of a given node.

When the bundle protocol agent of a node determines that a bundle must be forwarded to a node (either to a node that is a member of the bundle's destination endpoint or to some intermediate forwarding node) in the course of completing the successful transmission of that bundle, the bundle protocol agent invokes the services of one or more CLAs in a sustained effort to cause a copy of the bundle to be received by that node.

Upon reception, the processing of a bundle that has been received by a given node depends on whether or not the receiving node is registered in the bundle's destination endpoint. If it is, and if the payload of the bundle is non-fragmentary (possibly as a result of successful payload reassembly from fragmentary payloads, including the original payload of the newly received bundle), then the bundle is normally delivered to the node's application agent subject to the registration characterizing the node's membership in the destination endpoint.

The bundle protocol does not natively ensure delivery of a bundle to its destination. Data loss along the path to the destination node can be minimized by utilizing reliable convergence-layer protocols between neighbors on all segments of the end-to-end path, but for end-to-end bundle delivery assurance it will be necessary to develop extensions to the bundle protocol and/or application-layer mechanisms.

The bundle protocol is designed for extensibility. Bundle protocol extensions, documented elsewhere, may extend this specification by:

- defining additional blocks;
- defining additional administrative records;
- defining additional bundle processing flags;
- defining additional block processing flags;
- defining additional types of bundle status reports;
- defining additional bundle status report reason codes;
- defining additional mandates and constraints on processing that conformant bundle protocol agents must perform at specified points in the inbound and outbound bundle processing cycles.

3.3. Services Offered by Bundle Protocol Agents

The BPA of each node is expected to provide the following services to the node's application agent:

- commencing a registration (registering the node in an endpoint);

- terminating a registration;
- switching a registration between Active and Passive states;
- transmitting a bundle to an identified bundle endpoint;
- canceling a transmission;
- polling a registration that is in the Passive state;
- delivering a received bundle.

Note that the details of registration functionality are an implementation matter and are beyond the scope of this specification.

4. Bundle Format

4.1. Bundle Structure

The format of bundles **SHALL** conform to the Concise Binary Object Representation (CBOR [RFC8949]).

Cryptographic verification of a block is possible only if the sequence of octets on which the verifying node computes its hash - the canonicalized representation of the block - is identical to the sequence of octets on which the hash declared for that block was computed. To ensure that blocks are always in canonical representation when they are transmitted and received, the CBOR representations of the values of all fields in all blocks must conform to the rules for Canonical CBOR as specified in [RFC8949].

Each bundle **SHALL** be a concatenated sequence of at least two blocks, represented as a CBOR indefinite-length array. The first block in the sequence (the first item of the array) **MUST** be a primary bundle block in CBOR representation as described below; the bundle **MUST** have exactly one primary bundle block. The primary block **MUST** be followed by one or more canonical bundle blocks (additional array items) in CBOR representation as described in Section 4.3.2 below. Every block following the primary block **SHALL** be the CBOR representation of a canonical block. The last such block **MUST** be a payload block; the bundle **MUST** have exactly one payload block. The payload block **SHALL** be followed by a CBOR "break" stop code, terminating the array.

(Note that, while CBOR permits considerable flexibility in the encoding of bundles, this flexibility must not be interpreted as inviting increased complexity in protocol data unit structure.)

Associated with each block of a bundle is a block number. The block number uniquely identifies the block within the bundle, enabling blocks (notably bundle security protocol blocks) to reference other blocks in the same bundle without ambiguity. The block number of the primary block is implicitly zero; the block numbers of all other blocks are explicitly stated in block headers as noted below. Block numbering is unrelated to the order in which blocks are sequenced in the bundle. The block number of the payload block is always 1.

An implementation of the Bundle Protocol **MAY** discard any sequence of bytes that does not conform to the Bundle Protocol specification.

An implementation of the Bundle Protocol **MAY** accept a sequence of bytes that does not conform to the Bundle Protocol specification (e.g., one that represents data elements in fixed-length arrays rather than indefinite-length arrays) and transform it into conformant BP structure before processing it. Procedures for accomplishing such a transformation are beyond the scope of this specification.

4.2. BP Fundamental Data Structures

4.2.1. CRC Type

CRC type is an unsigned integer type code for which the following values (and no others) are valid:

- 0 indicates "no CRC is present."
- 1 indicates "a standard X-25 CRC-16 is present." [CRC16]
- 2 indicates "a standard CRC32C (Castagnoli) CRC-32 is present." [RFC4960]

CRC type **SHALL** be represented as a CBOR unsigned integer.

For examples of CRC32C CRCs, see Appendix A.4 of [RFC7143].

Note that more robust protection of BP data integrity, as needed, may be provided by means of Block Integrity Blocks as defined in the Bundle Security Protocol [BPSEC]).

4.2.2. CRC

CRC **SHALL** be omitted from a block if and only if the block's CRC type code is zero.

When not omitted, the CRC **SHALL** be represented as a CBOR byte string of two bytes (that is, CBOR additional information 2, if CRC type is 1) or of four bytes (that is, CBOR additional information 4, if CRC type is 2); in each case the sequence of bytes **SHALL** constitute an unsigned integer value (of 16 or 32 bits, respectively) in network byte order.

4.2.3. Bundle Processing Control Flags

Bundle processing control flags assert properties of the bundle as a whole rather than of any particular block of the bundle. They are conveyed in the primary block of the bundle.

The following properties are asserted by the bundle processing control flags:

- The bundle is a fragment. (Boolean)
- The bundle's payload is an administrative record. (Boolean)
- The bundle must not be fragmented. (Boolean)
- Acknowledgment by the user application is requested. (Boolean)
- Status time is requested in all status reports. (Boolean)
- Flags requesting types of status reports (all Boolean):
  - Request reporting of bundle reception.
  - Request reporting of bundle forwarding.

- ◦ Request reporting of bundle delivery.
- ◦ Request reporting of bundle deletion.

If the bundle processing control flags indicate that the bundle's application data unit is an administrative record, then all status report request flag values **MUST** be zero.

If the bundle's source node is omitted (i.e., the source node ID is the ID of the null endpoint, which has no members as discussed below; this option enables anonymous bundle transmission), then the bundle is not uniquely identifiable and all bundle protocol features that rely on bundle identity must therefore be disabled: the "Bundle must not be fragmented" flag value **MUST** be 1 and all status report request flag values **MUST** be zero.

Bundle processing control flags that are unrecognized **MUST** be ignored, as future definitions of additional flags might not be integrated simultaneously into the Bundle Protocol implementations operating at all nodes.

The bundle processing control flags **SHALL** be represented as a CBOR unsigned integer item, the value of which **SHALL** be processed as a bit field indicating the control flag values as follows (note that bit numbering in this instance is reversed from the usual practice, beginning with the low-order bit instead of the high-order bit, in recognition of the potential definition of additional control flag values in the future):

Bit 0 (the low-order bit, 0x000001): bundle is a fragment.

Bit 1 (0x000002): payload is an administrative record.

Bit 2 (0x000004): bundle must not be fragmented.

Bit 3 (0x000008): reserved.

Bit 4 (0x000010): reserved.

Bit 5 (0x000020): user application acknowledgement is requested.

Bit 6 (0x000040): status time is requested in all status reports.

Bit 7 (0x000080): reserved.

Bit 8 (0x000100): reserved.

Bit 9 (0x000200): reserved.

Bit 10(0x000400): reserved.

Bit 11(0x000800): reserved.

Bit 12(0x001000): reserved.

Bit 13(0x002000): reserved.

Bit 14(0x004000): bundle reception status reports are requested.

Bit 15(0x008000): reserved.

Bit 16(0x010000): bundle forwarding status reports are requested.

Bit 17(0x020000): bundle delivery status reports are requested.

Bit 18(0x040000): bundle deletion status reports are requested.

Bits 19-20 reserved.

Bits 21-63 unassigned.

4.2.4. Block Processing Control Flags

The block processing control flags assert properties of canonical bundle blocks. They are conveyed in the header of the block to which they pertain.

Block processing control flags that are unrecognized **MUST** be ignored, as future definitions of additional flags might not be integrated simultaneously into the Bundle Protocol implementations operating at all nodes.

The block processing control flags **SHALL** be represented as a CBOR unsigned integer item, the value of which **SHALL** be processed as a bit field indicating the control flag values as follows (note that bit numbering in this instance is reversed from the usual practice, beginning with the low-order bit instead of the high-order bit, for agreement with the bit numbering of the bundle processing control flags):

Bit 0(the low-order bit, 0x01): block must be replicated in every fragment.

Bit 1(0x02): transmission of a status report is requested if block can't be processed.

Bit 2(0x04): bundle must be deleted if block can't be processed.

Bit 3(0x08): reserved.

Bit 4(0x10): block must be removed from bundle if it can't be processed.

Bit 5(0x20): reserved.

Bit 6 (0x40): reserved.

Bits 7-63 unassigned.

For each bundle whose bundle processing control flags indicate that the bundle's application data unit is an administrative record, or whose source node ID is the null endpoint ID as defined below, the value of the "Transmit status report if block can't be processed" flag in every canonical block of the bundle **MUST** be zero.

4.2.5. Identifiers

4.2.5.1. Endpoint ID

The destinations of bundles are bundle endpoints, identified by text strings termed "endpoint IDs" (see Section 3.1). Each endpoint ID (EID) is a Uniform Resource Identifier (URI; [URI]). As such, each endpoint ID can be characterized as having this general structure:

< scheme name > : < scheme-specific part, or "SSP" >

The scheme identified by the < scheme name > in an endpoint ID is a set of syntactic and semantic rules that fully explain how to parse and interpret the SSP. Each scheme that may be used to form a BP endpoint ID must be added to the registry of URI scheme code numbers for the Bundle Protocol as maintained by IANA as described in Section 10; association of a unique URI scheme code number with each scheme name in this registry helps to enable compact representation of endpoint IDs in bundle blocks. Note that the set of allowable schemes is effectively unlimited. Any scheme conforming to [URIREG] may be added to the URI scheme code number registry and thereupon used in a bundle protocol endpoint ID.

Each entry in the URI scheme code number registry **MUST** contain a reference to a scheme code number definition document, which defines the manner in which the scheme-specific part of any URI formed in that scheme is parsed and interpreted and **MUST** be encoded, in CBOR representation, for transmission as a BP endpoint ID. The scheme code number definition document may also contain information as to (a) which convergence-layer protocol(s) may be used to forward a bundle to a BP destination endpoint identified by such an ID, and (b) how the ID of the convergence-layer protocol endpoint to use for that purpose can be inferred from that destination endpoint ID.

Note that, although endpoint IDs are URIs, implementations of the BP service interface may support expression of endpoint IDs in some internationalized manner (e.g., Internationalized Resource Identifiers (IRIs); see [RFC3987]).

Each BP endpoint ID (EID) **SHALL** be represented as a CBOR array comprising two items.

The first item of the array **SHALL** be the code number identifying the endpoint ID's URI scheme, as defined in the registry of URI scheme code numbers for the Bundle Protocol. Each URI scheme code number **SHALL** be represented as a CBOR unsigned integer.

The second item of the array **SHALL** be the applicable CBOR representation of the scheme-specific part (SSP) of the EID, defined as noted in the references(s) for the URI scheme code number registry entry for the EID's URI scheme.

4.2.5.1.1. The "dtn" URI scheme

The "dtn" scheme supports the identification of BP endpoints by arbitrarily expressive character strings. It is specified as follows:

Scheme syntax:

This specification uses the Augmented Backus-Naur Form (ABNF) notation of [RFC5234].

```
dtn-uri = "dtn:" ("none" / dtn-hier-part)

dtn-hier-part = "//" node-name name-delim demux ; a path-rootless

node-name = 1*(ALPHA/DIGIT/"-"/"."/"_") reg-name

name-delim = "/"

demux = *VCHAR
```

Scheme semantics: URIs of the dtn scheme are used as endpoint identifiers in the Delay-Tolerant Networking (DTN) Bundle Protocol (BP) as described in the present document.

The endpoint ID "dtn:none" identifies the "null endpoint", the endpoint that by definition never has any members.

All BP endpoints identified by all other dtn-scheme endpoint IDs for which the first character of demux is a character other than '~' (tilde) are singleton endpoints. All BP endpoints identified by dtn-scheme endpoint IDs for which the first character **is** '~' (tilde) are **not** singleton endpoints.

A dtn-scheme endpoint ID for which the demux is of length zero **MAY** identify the administrative endpoint for the node identified by node-name, and as such may serve as a node ID. No dtn-scheme endpoint ID for which the demux is of non-zero length may do so.

Note that these syntactic rules impose constraints on dtn-scheme endpoint IDs that were not imposed by the original specification of the dtn scheme as provided in [RFC5050]. It is believed that the dtn-scheme endpoint IDs employed by BP applications conforming to [RFC5050] are in most cases unlikely to be in violation of these rules, but the developers of such applications are advised of the potential for compromised interoperation.

Encoding considerations: For transmission as a BP endpoint ID, the scheme-specific part of a URI of the dtn scheme **SHALL** be represented as a CBOR text string unless the EID's SSP is "none", in which case the SSP **SHALL** be represented as a CBOR unsigned integer with the value zero. For all other purposes, URIs of the dtn scheme are encoded exclusively in US-ASCII characters.

Interoperability considerations: none.

Security considerations:

Reliability and consistency: none of the BP endpoints identified by the URIs of the dtn scheme are guaranteed to be reachable at any time, and the identity of the processing entities operating on those endpoints is never guaranteed by the Bundle Protocol itself. Bundle authentication as defined by the Bundle Security Protocol is required for this purpose.

Malicious construction: malicious construction of a conformant dtn-scheme URI is limited to the malicious selection of node names and the malicious selection of demux strings. That is, a maliciously constructed dtn-scheme URI could be used to direct a bundle to an endpoint that might be damaged by the arrival of that bundle or, alternatively, to declare a false source for a bundle and thereby cause incorrect processing at a node that receives the bundle. In both cases (and indeed in all bundle processing), the node that receives a bundle should verify its authenticity and validity before operating on it in any way.

Back-end transcoding: the limited expressiveness of URIs of the dtn scheme effectively eliminates the possibility of threat due to errors in back-end transcoding.

Rare IP address formats: not relevant, as IP addresses do not appear anywhere in conformant dtn-scheme URIs.

Sensitive information: because dtn-scheme URIs are used only to represent the identities of Bundle Protocol endpoints, the risk of disclosure of sensitive information due to interception of these URIs is minimal. Examination of dtn-scheme URIs could be used to support traffic analysis; where traffic analysis is a plausible danger, bundles should be conveyed by secure convergence-layer protocols that do not expose endpoint IDs.

Semantic attacks: the simplicity of dtn-scheme URI syntax minimizes the possibility of misinterpretation of a URI by a human user.

4.2.5.1.2. The "ipn" URI scheme

The "ipn" scheme supports the identification of BP endpoints by pairs of unsigned integers, for compact representation in bundle blocks. It is specified as follows:

Scheme syntax: This specification uses the Augmented Backus-Naur Form (ABNF) notation of [RFC5234], including the core ABNF syntax rule for DIGIT defined by that specification.

```
ipn-uri = "ipn:" ipn-hier-part

ipn-hier-part = node-nbr nbr-delim service-nbr ; a path-rootless

node-nbr = 1*DIGIT

nbr-delim = "."

service-nbr = 1*DIGIT
```

Scheme semantics: URIs of the ipn scheme are used as endpoint identifiers in the Delay-Tolerant Networking (DTN) Bundle Protocol (BP) as described in the present document.

All BP endpoints identified by ipn-scheme endpoint IDs are singleton endpoints.

An ipn-scheme endpoint ID for which service-nbr is zero **MAY** identify the administrative endpoint for the node identified by node-nbr, and as such may serve as a node ID. No ipn-scheme endpoint ID for which service-nbr is non-zero may do so.

Encoding considerations: For transmission as a BP endpoint ID, the scheme-specific part of a URI of the ipn scheme the SSP **SHALL** be represented as a CBOR array comprising two items. The first item of this array **SHALL** be the EID's node number (a number that identifies the node) represented as a CBOR unsigned integer. The second item of this array **SHALL** be the EID's service number (a number that identifies some application service) represented as a CBOR unsigned integer. For all other purposes, URIs of the ipn scheme are encoded exclusively in US-ASCII characters.

Interoperability considerations: none.

Security considerations:

Reliability and consistency: none of the BP endpoints identified by the URIs of the ipn scheme are guaranteed to be reachable at any time, and the identity of the processing entities operating on those endpoints is never guaranteed by the Bundle Protocol itself. Bundle authentication as defined by the Bundle Security Protocol [BPSEC] is required for this purpose.

Malicious construction: malicious construction of a conformant ipn-scheme URI is limited to the malicious selection of node numbers and the malicious selection of service numbers. That is, a maliciously constructed ipn-scheme URI could be used to direct a bundle to an endpoint that might be damaged by the arrival of that bundle or, alternatively, to declare a false source for a bundle and thereby cause incorrect processing at a node that receives the bundle. In both cases (and indeed in all bundle processing), the node that receives a bundle should verify its authenticity and validity before operating on it in any way.

Back-end transcoding: the limited expressiveness of URIs of the ipn scheme effectively eliminates the possibility of threat due to errors in back-end transcoding.

Rare IP address formats: not relevant, as IP addresses do not appear anywhere in conformant ipn-scheme URIs.

Sensitive information: because ipn-scheme URIs are used only to represent the identities of Bundle Protocol endpoints, the risk of disclosure of sensitive information due to interception of these URIs is minimal. Examination of ipn-scheme URIs could be used to support traffic analysis; where traffic analysis is a plausible danger, bundles should be conveyed by secure convergence-layer protocols that do not expose endpoint IDs.

Semantic attacks: the simplicity of ipn-scheme URI syntax minimizes the possibility of misinterpretation of a URI by a human user.

4.2.5.2. Node ID

For many purposes of the Bundle Protocol it is important to identify the node that is operative in some context.

As discussed in Section 3.1 above, nodes are distinct from endpoints; specifically, an endpoint is a set of zero or more nodes. But rather than define a separate namespace for node identifiers, we instead use endpoint identifiers to identify nodes as discussed in Section 3.2 above. Formally:

- Every node is, by definition, permanently registered in the singleton endpoint at which administrative records are delivered to its application agent's administrative element, termed the node's "administrative endpoint".
- As such, the EID of a node's administrative endpoint **SHALL** uniquely identify that node.
- A "node ID" is an EID that identifies the administrative endpoint of a node.

4.2.6. DTN Time

A DTN time is an unsigned integer indicating the number of milliseconds that have elapsed since the DTN Epoch, 2000-01-01 00:00:00 +0000 (UTC). DTN time is not affected by leap seconds.

Each DTN time **SHALL** be represented as a CBOR unsigned integer item. Implementers need to be aware that DTN time values conveyed in CBOR representation in bundles will nearly always exceed (2^{32} - 1); the manner in which a DTN time value is represented in memory is an implementation matter. The DTN time value zero indicates that the time is unknown.

4.2.7. Creation Timestamp

Each bundle's creation timestamp **SHALL** be represented as a CBOR array comprising two items.

The first item of the array, termed "bundle creation time", **SHALL** be the DTN time at which the transmission request was received that resulted in the creation of the bundle, represented as a CBOR unsigned integer.

The second item of the array, termed the creation timestamp's "sequence number", **SHALL** be the latest value (as of the time at which the transmission request was received) of a monotonically increasing positive integer counter managed by the source node's bundle protocol agent, represented as a CBOR unsigned integer. The sequence counter **MAY** be reset to zero whenever the current time advances by one millisecond.

For nodes that lack accurate clocks, it is recommended that bundle creation time be set to zero and that the counter used as the source of the bundle sequence count never be reset to zero.

Note that, in general, the creation of two distinct bundles with the same source node ID and bundle creation timestamp may result in unexpected network behavior and/or suboptimal performance. The combination of source node ID and bundle creation timestamp serves to identify a single transmission request, enabling it to be acknowledged by the receiving application (provided the source node ID is not the null endpoint ID).

4.2.8. Block-type-specific Data

Block-type-specific data in each block (other than the primary block) **SHALL** be the applicable CBOR representation of the content of the block. Details of this representation are included in the specification defining the block type.

4.3. Block Structures

This section describes the primary block in detail and non-primary blocks in general. Rules for processing these blocks appear in Section 5 of this document.

Note that supplementary DTN protocol specifications (including, but not restricted to, the Bundle Security Protocol [BPSEC]) may require that BP implementations conforming to those protocols construct and process additional blocks.

4.3.1. Primary Bundle Block

The primary bundle block contains the basic information needed to forward bundles to their destinations.

Each primary block **SHALL** be represented as a CBOR array; the number of elements in the array **SHALL** be 8 (if the bundle is not a fragment and the block has no CRC), 9 (if the block has a CRC and the bundle is not a fragment), 10 (if the bundle is a fragment and the block has no CRC), or 11 (if the bundle is a fragment and the block has a CRC).

The primary block of each bundle **SHALL** be immutable. The CBOR- encoded values of all fields in the primary block **MUST** remain unchanged from the time the block is created to the time it is delivered.

The fields of the primary bundle block **SHALL** be as follows, listed in the order in which they **MUST** appear:

Version: An unsigned integer value indicating the version of the bundle protocol that constructed this block. The present document describes version 7 of the bundle protocol. Version number **SHALL** be represented as a CBOR unsigned integer item.

Bundle Processing Control Flags: The Bundle Processing Control Flags are discussed in Section 4.2.3. above.

CRC Type: CRC Type codes are discussed in Section 4.2.1. above. The CRC Type code for the primary block **MAY** be zero if the bundle contains a BPsec [BPSEC] Block Integrity Block whose target is the primary block; otherwise the CRC Type code for the primary block **MUST** be non-zero.

Destination EID: The Destination EID field identifies the bundle endpoint that is the bundle's destination, i.e., the endpoint that contains the node(s) at which the bundle is to be delivered.

Source node ID: The Source node ID field identifies the bundle node at which the bundle was initially transmitted, except that Source node ID may be the null endpoint ID in the event that the bundle's source chooses to remain anonymous.

Report-to EID: The Report-to EID field identifies the bundle endpoint to which status reports pertaining to the forwarding and delivery of this bundle are to be transmitted.

Creation Timestamp: The creation timestamp comprises two unsigned integers that, together with the source node ID and (if the bundle is a fragment) the fragment offset and payload length, serve to identify the bundle. See Section 4.2.7 above for the definition of this field.

Lifetime: The lifetime field is an unsigned integer that indicates the time at which the bundle's payload will no longer be useful, encoded as a number of milliseconds past the creation time. (For high-rate deployments with very brief disruptions, fine-grained expression of bundle lifetime may be useful.) When a bundle's age exceeds its lifetime, bundle nodes need no longer retain or forward the bundle; the bundle **SHOULD** be deleted from the network.

If the asserted lifetime for a received bundle is so lengthy that retention of the bundle until its expiration time might degrade operation of the node at which the bundle is received, or if the bundle protocol agent of that node determines that the bundle must be deleted in order to prevent network performance degradation (e.g., the bundle appears to be part of a denial-of-service attack), then that bundle protocol agent **MAY** impose a temporary overriding lifetime of shorter duration; such overriding lifetime **SHALL NOT** replace the lifetime asserted in the bundle but **SHALL** serve as the bundle's effective lifetime while the bundle resides at that node. Procedures for imposing lifetime overrides are beyond the scope of this specification.

For bundles originating at nodes that lack accurate clocks, it is recommended that bundle age be obtained from the Bundle Age extension block (see Section 4.4.2 below) rather than from the difference between current time and bundle creation time. Bundle lifetime **SHALL** be represented as a CBOR unsigned integer item.

Fragment offset: If and only if the Bundle Processing Control Flags of this Primary block indicate that the bundle is a fragment, fragment offset **SHALL** be present in the primary block. Fragment offset **SHALL** be represented as a CBOR unsigned integer indicating the offset from the start of the original application data unit at which the bytes comprising the payload of this bundle were located.

Total Application Data Unit Length: If and only if the Bundle Processing Control Flags of this Primary block indicate that the bundle is a fragment, total application data unit length **SHALL** be present in the primary block. Total application data unit length **SHALL** be represented as a CBOR unsigned integer indicating the total length of the original application data unit of which this bundle's payload is a part.

CRC: A CRC **SHALL** be present in the primary block unless the bundle includes a BPsec [BPSEC] Block Integrity Block whose target is the primary block, in which case a CRC **MAY** be present in the primary block. The length and nature of the CRC **SHALL** be as indicated by the CRC type. The CRC **SHALL** be computed over the concatenation of all bytes (including CBOR "break" characters) of the primary block including the CRC field itself, which for this purpose **SHALL** be temporarily populated with all bytes set to zero.

4.3.2. Canonical Bundle Block Format

Every block other than the primary block (all such blocks are termed "canonical" blocks) **SHALL** be represented as a CBOR array; the number of elements in the array **SHALL** be 5 (if CRC type is zero) or 6 (otherwise).

The fields of every canonical block **SHALL** be as follows, listed in the order in which they **MUST** appear:

- Block type code, an unsigned integer. Bundle block type code 1 indicates that the block is a bundle payload block. Block type codes 2 through 9 are explicitly reserved as noted later in this specification. Block type codes 192 through 255 are not reserved and are available for private and/or experimental use. All other block type code values are reserved for future use.
- Block number, an unsigned integer as discussed in Section 4.1 above. Block number **SHALL** be represented as a CBOR unsigned integer.
- Block processing control flags as discussed in Section 4.2.4 above.
- CRC type as discussed in Section 4.2.1 above.
- Block-type-specific data represented as a single definite-length CBOR byte string, i.e., a CBOR byte string that is not of indefinite length. For each type of block, the block-type-specific data byte string is the serialization, in a block-type-specific manner, of the data conveyed by that type of block; definitions of blocks are required to define the manner in which block-type-specific data are serialized within the block-type-specific data field. For the Payload Block in particular (block type 1), the block-type-specific data field, termed the "payload", **SHALL** be an application data unit, or some contiguous extent thereof, represented as a definite-length CBOR byte string.
- If and only if the value of the CRC type field of this block is non-zero, a CRC. If present, the length and nature of the CRC **SHALL** be as indicated by the CRC type and the CRC **SHALL** be computed over the concatenation of all bytes of the block (including CBOR "break" characters) including the CRC field itself, which for this purpose **SHALL** be temporarily populated with all bytes set to zero.

4.4. Extension Blocks

"Extension blocks" are all blocks other than the primary and payload blocks. Three types of extension blocks are defined below. All implementations of the Bundle Protocol specification (the present document) **MUST** include procedures for recognizing, parsing, and acting on, but not necessarily producing, these types of extension blocks.

The specifications for additional types of extension blocks must indicate whether or not BP implementations conforming to those specifications must recognize, parse, act on, and/or produce blocks of those types. As not all nodes will necessarily instantiate BP implementations that conform to those additional specifications, it is possible for a node to receive a bundle that includes extension blocks that the node cannot process. The values of the block processing control flags indicate the action to be taken by the bundle protocol agent when this is the case.

No mandated procedure in this specification is unconditionally dependent on the absence or presence of any extension block. Therefore any bundle protocol agent **MAY** insert or remove any extension block in any bundle, subject to all mandates in the Bundle Protocol specification and all extension block specifications to which the node's BP implementation conforms. Note that removal of an extension block will probably disable one or more elements of bundle processing that were intended by the BPA that inserted that block. In particular, note that removal of an extension block that is one of the targets of a BPsec security block may render the bundle unverifiable.

The following extension blocks are defined in the current document.

4.4.1. Previous Node

The Previous Node block, block type 6, identifies the node that forwarded this bundle to the local node (i.e., to the node at which the bundle currently resides); its block-type-specific data is the node ID of that forwarder node which **SHALL** take the form of a node ID represented as described in Section 4.2.5.2. above. If the local node is the source of the bundle, then the bundle **MUST NOT** contain any Previous Node block. Otherwise the bundle **SHOULD** contain one (1) occurrence of this type of block and **MUST NOT** contain more than one.

4.4.2. Bundle Age

The Bundle Age block, block type 7, contains the number of milliseconds that have elapsed between the time the bundle was created and time at which it was most recently forwarded. It is intended for use by nodes lacking access to an accurate clock, to aid in determining the time at which a bundle's lifetime expires. The block-type-specific data of this block is an unsigned integer containing the age of the bundle in milliseconds, which **SHALL** be represented as a CBOR unsigned integer item. (The age of the bundle is the sum of all known intervals of the bundle's residence at forwarding nodes, up to the time at which the bundle was most recently forwarded, plus the summation of signal propagation time over all episodes of transmission between forwarding nodes. Determination of these values is an implementation matter.) If the bundle's creation time is zero, then the bundle **MUST** contain exactly one (1) occurrence of this type of block; otherwise, the bundle **MAY** contain at most one (1) occurrence of this type of block. A bundle **MUST NOT** contain multiple occurrences of the bundle age block, as this could result in processing anomalies.

4.4.3. Hop Count

The Hop Count block, block type 10, contains two unsigned integers, hop limit and hop count. A "hop" is here defined as an occasion on which a bundle was forwarded from one node to another node. Hop limit **MUST** be in the range 1 through 255. The hop limit value **SHOULD NOT** be changed at any time after creation of the Hop Count block; the hop count value **SHOULD** initially be zero and **SHOULD** be increased by 1 on each hop.

The hop count block is mainly intended as a safety mechanism, a means of identifying bundles for removal from the network that can never be delivered due to a persistent forwarding error. Hop count is particularly valuable as a defense against routing anomalies that might cause a bundle to be forwarded in a cyclical "ping-pong" fashion between two nodes. When a bundle's hop count exceeds its hop limit, the bundle **SHOULD** be deleted for the reason "hop limit exceeded", following the bundle deletion procedure defined in Section 5.10.

Procedures for determining the appropriate hop limit for a bundle are beyond the scope of this specification.

The block-type-specific data in a hop count block **SHALL** be represented as a CBOR array comprising two items. The first item of this array **SHALL** be the bundle's hop limit, represented as a CBOR unsigned integer. The second item of this array **SHALL** be the bundle's hop count, represented as a CBOR unsigned integer. A bundle **MAY** contain one occurrence of this type of block but **MUST NOT** contain more than one.

5. Bundle Processing

The bundle processing procedures mandated in this section and in Section 6 govern the operation of the Bundle Protocol Agent and the Application Agent administrative element of each bundle node. They are neither exhaustive nor exclusive. Supplementary DTN protocol specifications (including, but not restricted to, the Bundle Security Protocol [BPSEC]) may augment, override, or supersede the mandates of this document.

5.1. Generation of Administrative Records

All transmission of bundles is in response to bundle transmission requests presented by nodes' application agents. When required to "generate" an administrative record (such as a bundle status report), the bundle protocol agent itself is responsible for causing a new bundle to be transmitted, conveying that record. In concept, the bundle protocol agent discharges this responsibility by directing the administrative element of the node's application agent to construct the record and request its transmission as detailed in Section 6 below. In practice, the manner in which administrative record generation is accomplished is an implementation matter, provided the constraints noted in Section 6 are observed.

Status reports are relatively small bundles. Moreover, even when the generation of status reports is enabled the decision on whether or not to generate a requested status report is left to the discretion of the bundle protocol agent. Nonetheless, note that requesting status reports for any single bundle might easily result in the generation of (1 + (2 *(N-1))) status report bundles, where N is the number of nodes on the path from the bundle's source to its destination, inclusive. That is, the requesting of status reports for large numbers of bundles could result in an unacceptable increase in the bundle traffic in the network. For this reason, the generation of status reports **MUST** be disabled by default and enabled only when the risk of excessive network traffic is deemed acceptable. Mechanisms that could assist in assessing and mitigating this risk, such as pre-placed agreements authorizing the generation of status reports under specified circumstances, are beyond the scope of this specification.

Notes on administrative record terminology:

- A "bundle reception status report" is a bundle status report with the "reporting node received bundle" flag set to 1.
- A "bundle forwarding status report" is a bundle status report with the "reporting node forwarded the bundle" flag set to 1.
- A "bundle delivery status report" is a bundle status report with the "reporting node delivered the bundle" flag set to 1.

- A "bundle deletion status report" is a bundle status report with the "reporting node deleted the bundle" flag set to 1.

5.2. Bundle Transmission

The steps in processing a bundle transmission request are:

Step 1: Transmission of the bundle is initiated. An outbound bundle **MUST** be created per the parameters of the bundle transmission request, with the retention constraint "Dispatch pending". The source node ID of the bundle **MUST** be either the null endpoint ID, indicating that the source of the bundle is anonymous, or else the EID of a singleton endpoint whose only member is the node of which the BPA is a component.

Step 2: Processing proceeds from Step 1 of Section 5.4.

5.3. Bundle Dispatching

(Note that this procedure is initiated only following completion of Step 4 of Section 5.6.)

The steps in dispatching a bundle are:

Step 1: If the bundle's destination endpoint is an endpoint of which the node is a member, the bundle delivery procedure defined in Section 5.7 **MUST** be followed and for the purposes of all subsequent processing of this bundle at this node the node's membership in the bundle's destination endpoint **SHALL** be disavowed; specifically, even though the node is a member of the bundle's destination endpoint, the node **SHALL NOT** undertake to forward the bundle to itself in the course of performing the procedure described in Section 5.4.

Step 2: Processing proceeds from Step 1 of Section 5.4.

5.4. Bundle Forwarding

The steps in forwarding a bundle are:

Step 1: The retention constraint "Forward pending" **MUST** be added to the bundle, and the bundle's "Dispatch pending" retention constraint **MUST** be removed.

Step 2: The bundle protocol agent **MUST** determine whether or not forwarding is contraindicated (that is, rendered inadvisable) for any of the reasons listed in the IANA registry of Bundle Status Report Reason Codes (see Section 10.5 below), whose initial contents are listed in Table 2. In particular:

- The bundle protocol agent **MAY** choose either to forward the bundle directly to its destination node(s) (if possible) or to forward the bundle to some other node(s) for further forwarding. The manner in which this decision is made may depend on the scheme name in the destination endpoint ID and/or on other state but in any case is beyond the scope of this document; one possible mechanism is described in [SABR]. If the BPA elects to forward the bundle to some other node(s) for further forwarding but finds it impossible to select any node(s) to forward the bundle to, then forwarding is contraindicated.

- Provided the bundle protocol agent succeeded in selecting the node(s) to forward the bundle to, the bundle protocol agent **MUST** subsequently select the convergence layer adapter(s) whose services will enable the node to send the bundle to those nodes. The manner in which specific appropriate convergence layer adapters are selected is beyond the scope of this document; the TCP convergence-layer adapter [TCPCL] **MUST** be implemented when some or all of the bundles forwarded by the bundle protocol agent must be forwarded via the Internet but may not be appropriate for the forwarding of any particular bundle. If the agent finds it impossible to select any appropriate convergence layer adapter(s) to use in forwarding this bundle, then forwarding is contraindicated.

Step 3: If forwarding of the bundle is determined to be contraindicated for any of the reasons listed in the IANA registry of Bundle Status Report Reason Codes (see Section 10.5 below), then the Forwarding Contraindicated procedure defined in Section 5.4.1 **MUST** be followed; the remaining steps of Section 5.4 are skipped at this time.

Step 4: For each node selected for forwarding, the bundle protocol agent **MUST** invoke the services of the selected convergence layer adapter(s) in order to effect the sending of the bundle to that node. Determining the time at which the bundle protocol agent invokes convergence layer adapter services is a BPA implementation matter. Determining the time at which each convergence layer adapter subsequently responds to this service invocation by sending the bundle is a convergence-layer adapter implementation matter. Note that:

- If the bundle has a Previous Node block, as defined in Section 4.4.1 above, then that block **MUST** be removed from the bundle before the bundle is forwarded.
- If the bundle protocol agent is configured to attach Previous Node blocks to forwarded bundles, then a Previous Node block containing the node ID of the forwarding node **MUST** be inserted into the bundle before the bundle is forwarded.
- If the bundle has a bundle age block, as defined in Section 4.4.2 above, then at the last possible moment before the CLA initiates conveyance of the bundle via the CL protocol the bundle age value **MUST** be increased by the difference between the current time and the time at which the bundle was received (or, if the local node is the source of the bundle, created).

Step 5: When all selected convergence layer adapters have informed the bundle protocol agent that they have concluded their data sending procedures with regard to this bundle, processing may depend on the results of those procedures.

If completion of the data sending procedures by all selected convergence layer adapters has not resulted in successful forwarding of the bundle (an implementation-specific determination that is beyond the scope of this specification), then the bundle protocol agent **MAY** choose (in an implementation-specific manner, again beyond the scope of this specification) to initiate another attempt to forward the bundle. In that event, processing proceeds from Step 4. The minimum number of times a given node will initiate another forwarding attempt for any single bundle in this event (a number which may be zero) is a node configuration parameter that must be exposed to other nodes in the network to the extent that this is required by the operating environment.

If completion of the data sending procedures by all selected convergence layer adapters HAS resulted in successful forwarding of the bundle, or if it has not but the bundle protocol agent does not choose to initiate another attempt to forward the bundle, then:

- If the "request reporting of bundle forwarding" flag in the bundle's status report request field is set to 1, and status reporting is enabled, then a bundle forwarding status report **SHOULD** be generated, destined for the bundle's report-to endpoint ID. The reason code on this bundle forwarding status report **MUST** be "no additional information".
- If any applicable bundle protocol extensions mandate generation of status reports upon conclusion of convergence-layer data sending procedures, all such status reports **SHOULD** be generated with extension-mandated reason codes.
- The bundle's "Forward pending" retention constraint **MUST** be removed.

5.4.1. Forwarding Contraindicated

The steps in responding to contraindication of forwarding are:

Step 1: The bundle protocol agent **MUST** determine whether or not to declare failure in forwarding the bundle. Note: this decision is likely to be influenced by the reason for which forwarding is contraindicated.

Step 2: If forwarding failure is declared, then the Forwarding Failed procedure defined in Section 5.4.2 **MUST** be followed.

Otherwise, when - at some future time - the forwarding of this bundle ceases to be contraindicated, processing proceeds from Step 4 of Section 5.4.

5.4.2. Forwarding Failed

The steps in responding to a declaration of forwarding failure are:

Step 1: The bundle protocol agent **MAY** forward the bundle back to the node that sent it, as identified by the Previous Node block, if present. This forwarding, if performed, **SHALL** be accomplished by performing Step 4 and Step 5 of Section 5.4 where the sole node selected for forwarding **SHALL** be the node that sent the bundle.

Step 2: If the bundle's destination endpoint is an endpoint of which the node is a member, then the bundle's "Forward pending" retention constraint **MUST** be removed. Otherwise, the bundle **MUST** be deleted: the bundle deletion procedure defined in Section 5.10 **MUST** be followed, citing the reason for which forwarding was determined to be contraindicated.

5.5. Bundle Expiration

A bundle expires when the bundle's age exceeds its lifetime as specified in the primary bundle block or as overridden by the bundle protocol agent. Bundle age **MAY** be determined by subtracting the bundle's creation timestamp time from the current time if (a) that timestamp time is not zero and (b) the local node's clock is known to be accurate; otherwise bundle age **MUST** be obtained from the Bundle Age extension block. Bundle expiration **MAY** occur at any point in the processing of a bundle. When a bundle expires, the bundle protocol agent **MUST** delete the bundle for the reason "lifetime expired" (when the expired lifetime is the lifetime as specified in the primary block) or "traffic pared" (when the expired lifetime is a lifetime override as imposed by the bundle protocol agent): the bundle deletion procedure defined in Section 5.10 **MUST** be followed.

5.6. Bundle Reception

The steps in processing a bundle that has been received from another node are:

Step 1: The retention constraint "Dispatch pending" **MUST** be added to the bundle.

Step 2: If the "request reporting of bundle reception" flag in the bundle's status report request field is set to 1, and status reporting is enabled, then a bundle reception status report with reason code "No additional information" **SHOULD** be generated, destined for the bundle's report-to endpoint ID.

Step 3: CRCs **SHOULD** be computed for every block of the bundle that has an attached CRC. If any block of the bundle is malformed according to this specification (including syntactically invalid CBOR), or if any block has an attached CRC and the CRC computed for this block upon reception differs from that attached CRC, then the bundle protocol agent **MUST** delete the bundle for the reason "Block unintelligible". The bundle deletion procedure defined in Section 5.10 **MUST** be followed and all remaining steps of the bundle reception procedure **MUST** be skipped.

Step 4: For each block in the bundle that is an extension block that the bundle protocol agent cannot process:

- If the block processing flags in that block indicate that a status report is requested in this event, and status reporting is enabled, then a bundle reception status report with reason code "Block unsupported" **SHOULD** be generated, destined for the bundle's report-to endpoint ID.
- If the block processing flags in that block indicate that the bundle must be deleted in this event, then the bundle protocol agent **MUST** delete the bundle for the reason "Block unsupported"; the bundle deletion procedure defined in Section 5.10 **MUST** be followed and all remaining steps of the bundle reception procedure **MUST** be skipped.
- If the block processing flags in that block do NOT indicate that the bundle must be deleted in this event but do indicate that the block must be discarded, then the bundle protocol agent **MUST** remove this block from the bundle.

- If the block processing flags in that block indicate neither that the bundle must be deleted nor that that the block must be discarded, then processing continues with the next extension block that the bundle protocol agent cannot process, if any; otherwise, processing proceeds from step 5.

Step 5: Processing proceeds from Step 1 of Section 5.3.

5.7. Local Bundle Delivery

The steps in processing a bundle that is destined for an endpoint of which this node is a member are:

Step 1: If the received bundle is a fragment, the application data unit reassembly procedure described in Section 5.9 **MUST** be followed. If this procedure results in reassembly of the entire original application data unit, processing of the fragmentary bundle whose payload has been replaced by the reassembled application data unit (whether this bundle or a previously received fragment) proceeds from Step 2; otherwise, the retention constraint "Reassembly pending" **MUST** be added to the bundle and all remaining steps of this procedure **MUST** be skipped.

Step 2: Delivery depends on the state of the registration whose endpoint ID matches that of the destination of the bundle:

- An additional implementation-specific delivery deferral procedure **MAY** optionally be associated with the registration.
- If the registration is in the Active state, then the bundle **MUST** be delivered automatically as soon as it is the next bundle that is due for delivery according to the BPA's bundle delivery scheduling policy, an implementation matter.
- If the registration is in the Passive state, or if delivery of the bundle fails for some implementation-specific reason, then the registration's delivery failure action **MUST** be taken. Delivery failure action **MUST** be one of the following:
    - defer delivery of the bundle subject to this registration until (a) this bundle is the least recently received of all bundles currently deliverable subject to this registration and (b) either the registration is polled or else the registration is in the Active state, and also perform any additional delivery deferral procedure associated with the registration; or
    - abandon delivery of the bundle subject to this registration (as defined in Section 3.1).

Step 3: As soon as the bundle has been delivered, if the "request reporting of bundle delivery" flag in the bundle's status report request field is set to 1 and bundle status reporting is enabled, then a bundle delivery status report **SHOULD** be generated, destined for the bundle's report-to endpoint ID. Note that this status report only states that the payload has been delivered to the application agent, not that the application agent has processed that payload.

5.8. Bundle Fragmentation

It may at times be advantageous for bundle protocol agents to reduce the sizes of bundles in order to forward them. This might be the case, for example, if a node to which a bundle is to be forwarded is accessible only via intermittent contacts and no upcoming contact is long enough to enable the forwarding of the entire bundle.

The size of a bundle can be reduced by "fragmenting" the bundle. To fragment a bundle whose payload is of size M is to replace it with two "fragments" - new bundles with the same source node ID and creation timestamp as the original bundle - whose payloads **MUST** be the first N and the last (M - N) bytes of the original bundle's payload, where 0 < N < M.

Note that fragments are bundles and therefore may themselves be fragmented, so multiple episodes of fragmentation may in effect replace the original bundle with more than two fragments. (However, there is only one "level" of fragmentation, as in IP fragmentation.)

Any bundle whose primary block's bundle processing flags do NOT indicate that it must not be fragmented **MAY** be fragmented at any time, for any purpose, at the discretion of the bundle protocol agent. NOTE, however, that some combinations of bundle fragmentation, replication, and routing might result in unexpected traffic patterns.

Fragmentation **SHALL** be constrained as follows:

- The concatenation of the payloads of all fragments produced by fragmentation **MUST** always be identical to the payload of the fragmented bundle (that is, the bundle that is being fragmented). Note that the payloads of fragments resulting from different fragmentation episodes, in different parts of the network, may be overlapping subsets of the fragmented bundle's payload.
- The primary block of each fragment **MUST** differ from that of the fragmented bundle, in that the bundle processing flags of the fragment **MUST** indicate that the bundle is a fragment and both fragment offset and total application data unit length must be provided. Additionally, the CRC of the primary block of the fragmented bundle, if any, **MUST** be replaced in each fragment by a new CRC computed for the primary block of that fragment.
- The payload blocks of fragments will differ from that of the fragmented bundle as noted above.
- If the fragmented bundle is not a fragment or is the fragment with offset zero, then all extension blocks of the fragmented bundle **MUST** be replicated in the fragment whose offset is zero.
- Each of the fragmented bundle's extension blocks whose "Block must be replicated in every fragment" flag is set to 1 **MUST** be replicated in every fragment.
- Beyond these rules, rules for the replication of extension blocks in the fragments must be defined in the specifications for those extension block types.

5.9. Application Data Unit Reassembly

Note that the bundle fragmentation procedure described in Section 5.8 above may result in the replacement of a single original bundle with an arbitrarily large number of fragmentary bundles. In order to be delivered at a destination node, the original bundle's payload must be reassembled from the payloads of those fragments.

The "material extents" of a received fragment's payload are all continuous sequences of bytes in that payload that do not overlap with the material extents of the payloads of any previously received fragments with the same source node ID and creation timestamp. If the concatenation - as informed by fragment offsets and payload lengths - of the material extents of the payloads of this fragment and all previously received fragments with the same source node ID and creation timestamp as this fragment forms a continuous byte array whose length is equal to the total application data unit length noted in the fragment's primary block, then:

- This byte array -- the reassembled application data unit -- **MUST** replace the payload of that fragment whose material extents include the extent at offset zero. Note that this will enable delivery of the reconstituted original bundle as described in Step 1 of Section 5.7.
- The "Reassembly pending" retention constraint **MUST** be removed from every other fragment with the same source node ID and creation timestamp as this fragment.

Note: reassembly of application data units from fragments occurs at the nodes that are members of destination endpoints as necessary; an application data unit **MAY** also be reassembled at some other node on the path to the destination.

5.10. Bundle Deletion

The steps in deleting a bundle are:

Step 1: If the "request reporting of bundle deletion" flag in the bundle's status report request field is set to 1, and if status reporting is enabled, then a bundle deletion status report citing the reason for deletion **SHOULD** be generated, destined for the bundle's report-to endpoint ID.

Step 2: All of the bundle's retention constraints **MUST** be removed.

5.11. Discarding a Bundle

As soon as a bundle has no remaining retention constraints it **MAY** be discarded, thereby releasing any persistent storage that may have been allocated to it.

5.12. Canceling a Transmission

When requested to cancel a specified transmission, where the bundle created upon initiation of the indicated transmission has not yet been discarded, the bundle protocol agent **MUST** delete that bundle for the reason "transmission cancelled". For this purpose, the procedure defined in Section 5.10 **MUST** be followed.

6. Administrative Record Processing

6.1. Administrative Records

Administrative records are standard application data units that are used in providing some of the features of the Bundle Protocol. One type of administrative record has been defined to date: bundle status reports. Note that additional types of administrative records may be defined by supplementary DTN protocol specification documents.

Every administrative record consists of:

- Record type code (an unsigned integer for which valid values are as defined below).
- Record content in type-specific format.

Valid administrative record type codes are defined as follows:

Value	Meaning
1	Bundle status report.
(other)	Reserved for future use.

Table 1: Administrative Record Type Codes

Each BP administrative record **SHALL** be represented as a CBOR array comprising two items.

The first item of the array **SHALL** be a record type code, which **SHALL** be represented as a CBOR unsigned integer.

The second element of this array **SHALL** be the applicable CBOR representation of the content of the record. Details of the CBOR representation of administrative record type 1 are provided below. Details of the CBOR representation of other types of administrative record type are included in the specifications defining those records.

6.1.1. Bundle Status Reports

The transmission of "bundle status reports" under specified conditions is an option that can be invoked when transmission of a bundle is requested. These reports are intended to provide information about how bundles are progressing through the system, including notices of receipt, forwarding, final delivery, and deletion. They are transmitted to the Report-to endpoints of bundles.

Each bundle status report **SHALL** be represented as a CBOR array. The number of elements in the array **SHALL** be either 6 (if the subject bundle is a fragment) or 4 (otherwise).

The first item of the bundle status report array **SHALL** be bundle status information represented as a CBOR array of at least 4 elements. The first four items of the bundle status information array shall provide information on the following four status assertions, in this order:

- Reporting node received bundle.
- Reporting node forwarded the bundle.
- Reporting node delivered the bundle.
- Reporting node deleted the bundle.

Each item of the bundle status information array **SHALL** be a bundle status item represented as a CBOR array; the number of elements in each such array **SHALL** be either 2 (if the value of the first item of this bundle status item is 1 AND the "Report status time" flag was set to 1 in the bundle processing flags of the bundle whose status is being reported) or 1 (otherwise). The first item of the bundle status item array **SHALL** be a status indicator, a Boolean value indicating whether or not the corresponding bundle status is asserted, represented as a CBOR Boolean value. The second item of the bundle status item array, if present, **SHALL** indicate the time (as reported by the local system clock, an implementation matter) at which the indicated status was asserted for this bundle, represented as a DTN time as described in Section 4.2.6. above.

The second item of the bundle status report array **SHALL** be the bundle status report reason code explaining the value of the status indicator, represented as a CBOR unsigned integer. Valid status report reason codes are registered in the IANA "Bundle Status Report Reason Codes" registry in the Bundle Protocol Namespace (see Section 10.5). The initial contents of that registry are listed in Table 2 below, but the list of status report reason codes provided here is neither exhaustive nor exclusive; supplementary DTN protocol specifications (including, but not restricted to, the Bundle Security Protocol [BPSEC]) may define additional reason codes.

Value	Meaning
0	No additional information.
1	Lifetime expired.
2	Forwarded over unidirectional link.
3	Transmission canceled.
4	Depleted storage.
5	Destination endpoint ID unavailable.
6	No known route to destination from here.
7	No timely contact with next node on route.
8	Block unintelligible.

Value	Meaning
9	Hop limit exceeded.
10	Traffic pared (e.g., status reports).
11	Block unsupported.
(other)	Reserved for future use.

Table 2: Status Report Reason Codes

The third item of the bundle status report array **SHALL** be the source node ID identifying the source of the bundle whose status is being reported, represented as described in Section 4.2.5.1.1. above.

The fourth item of the bundle status report array **SHALL** be the creation timestamp of the bundle whose status is being reported, represented as described in Section 4.2.7. above.

The fifth item of the bundle status report array **SHALL** be present if and only if the bundle whose status is being reported contained a fragment offset. If present, it **SHALL** be the subject bundle's fragment offset represented as a CBOR unsigned integer item.

The sixth item of the bundle status report array **SHALL** be present if and only if the bundle whose status is being reported contained a fragment offset. If present, it **SHALL** be the length of the subject bundle's payload represented as a CBOR unsigned integer item.

Note that the forwarding parameters (such as lifetime, applicable security measures, etc.) of the bundle whose status is being reported **MAY** be reflected in the parameters governing the forwarding of the bundle that conveys a status report, but this is an implementation matter. Bundle protocol deployment experience to date has not been sufficient to suggest any clear guidance on this topic.

6.2. Generation of Administrative Records

Whenever the application agent's administrative element is directed by the bundle protocol agent to generate an administrative record, the following procedure must be followed:

Step 1: The administrative record must be constructed. If the administrative record references a bundle and the referenced bundle is a fragment, the administrative record **MUST** contain the fragment offset and fragment length.

Step 2: A request for transmission of a bundle whose payload is this administrative record **MUST** be presented to the bundle protocol agent.

7. Services Required of the Convergence Layer

7.1. The Convergence Layer

The successful operation of the end-to-end bundle protocol depends on the operation of underlying protocols at what is termed the "convergence layer"; these protocols accomplish communication between nodes. A wide variety of protocols may serve this purpose, so long as each convergence layer protocol adapter provides a defined minimal set of services to the bundle protocol agent. This convergence layer service specification enumerates those services.

7.2. Summary of Convergence Layer Services

Each convergence layer protocol adapter is expected to provide the following services to the bundle protocol agent:

- sending a bundle to a bundle node that is reachable via the convergence layer protocol;
- notifying the bundle protocol agent of the disposition of its data sending procedures with regard to a bundle, upon concluding those procedures;
- delivering to the bundle protocol agent a bundle that was sent by a bundle node via the convergence layer protocol.

The convergence layer service interface specified here is neither exhaustive nor exclusive. That is, supplementary DTN protocol specifications (including, but not restricted to, the Bundle Security Protocol [BPSEC]) may expect convergence layer adapters that serve BP implementations conforming to those protocols to provide additional services such as reporting on the transmission and/or reception progress of individual bundles (at completion and/or incrementally), retransmitting data that were lost in transit, discarding bundle-conveying data units that the convergence layer protocol determines are corrupt or inauthentic, or reporting on the integrity and/or authenticity of delivered bundles.

In addition, bundle protocol relies on the capabilities of protocols at the convergence layer to minimize congestion in the store-carry-forward overlay network. The potentially long round-trip times characterizing delay-tolerant networks are incompatible with end-to- end reactive congestion control mechanisms, so convergence-layer protocols **MUST** provide rate limiting or congestion control.

8. Implementation Status

[NOTE to the RFC Editor: please remove this section before publication, as well as the reference to RFC 7942.]

This section records the status of known implementations of the protocol defined by this specification at the time of posting of this Internet-Draft, and is based on a proposal described in RFC 7942. The description of implementations in this section is intended to assist the IETF in its decision processes in progressing drafts to RFCs. Please note that the listing of any individual implementation here does not imply endorsement by the IETF. Furthermore, no effort has been spent to verify the information presented here that was supplied by IETF contributors. This is not intended as, and must not be construed to be, a catalog of available implementations or their features. Readers are advised to note that other implementations may exist.

According to RFC 7942, "this will allow reviewers and working groups to assign due consideration to documents that have the benefit of running code, which may serve as evidence of valuable experimentation and feedback that have made the implemented protocols more mature. It is up to the individual working groups to use this information as they see fit".

At the time of this writing, there are six known implementations of the current document.

The first known implementation is microPCN (https://upcn.eu/). According to the developers:

The Micro Planetary Communication Network (uPCN) is a free software project intended to offer an implementation of Delay-Tolerant Networking protocols for POSIX operating systems (well, and for Linux) plus for the ARM Cortex STM32F4 microcontroller series. More precisely it currently provides an implementation of

- the Bundle Protocol (BP, RFC 5050),
- version 6 of the Bundle Protocol version 7 specification draft,
- the DTN IP Neighbor Discovery (IPND) protocol, and
- a routing approach optimized for message-ferry micro LEO satellites.

uPCN is written in C and is built upon the real-time operating system FreeRTOS. The source code of uPCN is released under the "BSD 3-Clause License".

The project depends on an execution environment offering link layer protocols such as AX.25. The source code uses the USB subsystem to interact with the environment.

The second known implementation is PyDTN, developed by X-works, s.r.o (https://x-works.sk/). The final third of the implementation was developed during the IETF 101 Hackathon. According to the developers, PyDTN implements bundle coding/decoding and neighbor discovery. PyDTN is written in Python and has been shown to be interoperable with uPCN.

The third known implementation is "Terra" (https://github.com/RightMesh/Terra/), a Java implementation developed in the context of terrestrial DTN. It includes an implementation of a "minimal TCP" convergence layer adapter.

The fourth and fifth known implementations are products of cooperating groups at two German universities:

- An implementation written in Go, licensed under GPLv3, is focused on being easily extensible suitable for research. It is maintained at the University of Marburg and can be accessed from https://github.com/dtn7/dtn7-go.
- An implementation written in Rust, licensed under the MIT/Apache license, is intended for environments with limited resources or demanding safety and/or performance

requirements. It is maintained at the Technical University of Darmstadt and can be accessed at https://github.com/dtn7/dtn7-rs/.

The sixth known implementation is the "bpv7" module in version 4.0.0 of the Interplanetary Overlay Network (ION) software maintained at the Jet Propulsion Laboratory, California Institute of Technology, for the U.S. National Aeronautics and Space Administration (NASA).

9. Security Considerations

The bundle protocol security architecture and the available security services are specified in an accompanying document, the Bundle Security Protocol (BPsec) specification [BPSEC]. Whenever Bundle Protocol security services (as opposed to the security services provided by overlying application protocols or underlying convergence-layer protocols) are required, those services **SHALL** be provided by BPsec rather than by some other mechanism with the same or similar scope.

A Bundle Protocol Agent (BPA) which sources, cryptographically verifies, and/or accepts a bundle **MUST** implement support for BPsec. Use of BPsec for a particular Bundle Protocol session is optional.

The BPsec extensions to Bundle Protocol enable each block of a bundle (other than a BPsec extension block) to be individually authenticated by a signature block (Block Integrity Block, or BIB) and also enable each block of a bundle other than the primary block (and the BPsec extension blocks themselves) to be individually encrypted by a Block Confidentiality Block (BCB).

Because the security mechanisms are extension blocks that are themselves inserted into the bundle, the protections they afford apply while the bundle is at rest, awaiting transmission at the next forwarding opportunity, as well as in transit.

Additionally, convergence-layer protocols that ensure authenticity of communication between adjacent nodes in BP network topology **SHOULD** be used where available, to minimize the ability of unauthenticated nodes to introduce inauthentic traffic into the network. Convergence-layer protocols that ensure confidentiality of communication between adjacent nodes in BP network topology **SHOULD** also be used where available, to minimize exposure of the bundle's primary block and other clear-text blocks, thereby offering some defense against traffic analysis.

In order to provide authenticity and/or confidentiality of communication between BP nodes, the convergence-layer protocol requires as input the name(s) of the expected communication peer (s). These must be supplied by the convergence-layer adapter. Details of the means by which the CLA determines which CL endpoint name(s) must be provided to the CL protocol are out of scope for this specification. Note, though, that when the CL endpoint names are a function of BP endpoint IDs, the correctness and authenticity of that mapping will be vital to the overall security properties that the CL provides to the system.

Note that, while the primary block must remain in the clear for routing purposes, the Bundle Protocol could be protected against traffic analysis to some extent by using bundle-in-bundle encapsulation [BIBE] to tunnel bundles to a safe forward distribution point: the encapsulated bundle could form the payload of an encapsulating bundle, and that payload block could be encrypted by a BCB.

Note that the generation of bundle status reports is disabled by default because malicious initiation of bundle status reporting could result in the transmission of extremely large numbers of bundles, effecting a denial of service attack. Imposing bundle lifetime overrides would constitute one defense against such an attack.

Note also that the reception of large numbers of fragmentary bundles with very long lifetimes could constitute a denial of service attack, occupying storage while pending reassembly that will never occur. Imposing bundle lifetime overrides would, again, constitute one defense against such an attack.

This protocol makes use of absolute timestamps for several purposes. Provisions are included for nodes without accurate clocks to retain most of the protocol functionality, but nodes that are unaware that their clock is inaccurate may exhibit unexpected behavior.

10. IANA Considerations

The Bundle Protocol includes fields requiring registries managed by IANA.

10.1. Bundle Block Types

The current "Bundle Block Types" registry in the Bundle Protocol Namespace is augmented by adding a column identifying the version of the Bundle Protocol (Bundle Protocol Version) that applies to the new values. IANA has added the following values, as described in Section 4.3.1, to the "Bundle Block Types" registry. The current values in the "Bundle Block Types" registry have the Bundle Protocol Version set to the value "6", as shown below.

Bundle Protocol Version	Value	Description	Reference
none	0	Reserved	[RFC6255]
6,7	1	Bundle Payload Block	[RFC5050], RFC 0000
6	2	Bundle Authentication Block	[RFC6257]
6	3	Payload Integrity Block	[RFC6257]
6	4	Payload Confidentiality Block	[RFC6257]
6	5	Previous-Hop Insertion Block	[RFC6259]

Bundle Protocol Version	Value	Description	Reference
7	6	Previous node (proximate sender)	RFC 0000
7	7	Bundle age (in milliseconds)	RFC 0000
6	8	Metadata Extension Block	[RFC6258]
6	9	Extension Security Block	[RFC6257]
7	10	Hop count (#prior xmit attempts)	RFC 0000
7	11-191	Unassigned	
6,7	192-255	Reserved for Private and/or Experimental Use	[RFC5050], RFC 0000

Table 3

10.2. Primary Bundle Protocol Version

IANA has added the following value to the "Primary Bundle Protocol Version" registry in the Bundle Protocol Namespace.

Value	Description	Reference
7	Assigned	RFC 0000

Table 4

Values 8-255 (rather than 7-255) are now Unassigned.

10.3. Bundle Processing Control Flags

The current "Bundle Processing Control Flags" registry in the Bundle Protocol Namespace is augmented by adding a column identifying the version of the Bundle Protocol (Bundle Protocol Version) that applies to the new values. IANA has added the following values, as described in Section 4.2.3, to the "Bundle Processing Control Flags" registry. The current values in the "Bundle Processing Control Flags" registry have the Bundle Protocol Version set to the value 6 or "6, 7", as shown below.

Bundle Protocol Version	Bit Position (right to left)	Description	Reference
6,7	0	Bundle is a fragment	[RFC5050], RFC 0000

Bundle Protocol Version	Bit Position (right to left)	Description	Reference
6,7	1	Application data unit is an administrative record	[RFC5050], RFC 0000
6,7	2	Bundle must not be fragmented	[RFC5050], RFC 0000
6	3	Custody transfer is requested	[RFC5050]
6	4	Destination endpoint is singleton	[RFC5050]
6,7	5	Acknowledgement by application is requested	[RFC5050], RFC 0000
7	6	Status time requested in reports	RFC 0000
6	7	Class of service, priority	[RFC5050]
6	8	Class of service, priority	[RFC5050]
6	9	Class of service, reserved	[RFC5050]
6	10	Class of service, reserved	[RFC5050]
6	11	Class of service, reserved	[RFC5050]
6	12	Class of service, reserved	[RFC5050]
6	13	Class of service, reserved	[RFC5050]
6,7	14	Request reporting of bundle reception	[RFC5050], RFC 0000
6	15	Request reporting of custody acceptance	[RFC5050]
6,7	16	Request reporting of bundle forwarding	[RFC5050], RFC 0000
6,7	17	Request reporting of bundle delivery	[RFC5050], RFC 0000
6,7	18	Request reporting of bundle deletion	[RFC5050], RFC 0000
6,7	19	Reserved	[RFC5050], RFC 0000

Bundle Protocol Version	Bit Position (right to left)	Description	Reference
6,7	20	Reserved	[RFC5050], RFC 0000
	21-63	Unassigned	

Table 5: Bundle Processing Control Flags Registry

10.4. Block Processing Control Flags

The current "Block Processing Control Flags" registry in the Bundle Protocol Namespace is augmented by adding a column identifying the version of the Bundle Protocol (Bundle Protocol Version) that applies to the related BP version. The current values in the "Block Processing Control Flags" registry have the Bundle Protocol Version set to the value 6 or "6, 7", as shown below.

Bundle Protocol Version	Bit Version Position (right to left)	Description	Reference
6,7	0	Block must be replicated in every fragment	[RFC5050], RFC 0000
6,7	1	Transmit status report if block can't be processed	[RFC5050], RFC 0000
6,7	2	Delete bundle if block can't be processed	[RFC5050], RFC 0000
6	3	Last block	[RFC5050]
6,7	4	Discard block if it can't be processed	[RFC5050], RFC 0000
6	5	Block was forwarded without being processed	[RFC5050]
6	6	Block contains an EID reference field	[RFC5050]
	7-63	Unassigned	

Table 6: Block Processing Control Flags Registry

10.5. Bundle Status Report Reason Codes

The current "Bundle Status Report Reason Codes" registry in the Bundle Protocol Namespace is augmented by adding a column identifying the version of the Bundle Protocol (Bundle Protocol Version) that applies to the new values. IANA has added the following values, as described in Section 6.1.1, to the "Bundle Status Report Reason Codes" registry. The current values in the "Bundle Status Report Reason Codes" registry have the Bundle Protocol Version set to the value 6 or 7 or "6, 7", as shown below.

Bundle Protocol Version	Value	Description	Reference
6,7	0	No additional information	[RFC5050], RFC 0000
6,7	1	Lifetime expired	[RFC5050], RFC 0000
6,7	2	Forwarded over unidirectional link	[RFC5050], RFC 0000
6,7	3	Transmission canceled	[RFC5050], RFC 0000
6,7	4	Depleted storage	[RFC5050], RFC 0000
6,7	5	Destination endpoint ID unavailable	[RFC5050], RFC 0000
6,7	6	No known route to destination from here	[RFC5050], RFC 0000
6,7	7	No timely contact with next node on route	[RFC5050], RFC 0000
6,7	8	Block unintelligible	[RFC5050], RFC 0000
7	9	Hop limit exceeded	RFC 0000
7	10	Traffic pared	RFC 0000
7	11	Block unsupported	RFC 0000
	12-254	Unassigned	

Bundle Protocol Version	Value	Description	Reference
6,7	255	Reserved	[RFC6255], RFC 0000

Table 7: Bundle Status Report Reason Codes Registry

10.6. Bundle Protocol URI Scheme Types

The Bundle Protocol has a URI scheme type field - an unsigned integer of indefinite length - for which IANA has created, and will maintain, a new "Bundle Protocol URI Scheme Type" registry in the Bundle Protocol Namespace. The "Bundle Protocol URI Scheme Type" registry governs a namespace of unsigned integers. Initial values for the "Bundle Protocol URI Scheme Type" registry are given below.

The registration policy for this registry is: Standards Action. The allocation should only be granted for a standards-track RFC approved by the IESG.

The value range is: unsigned integer.

Each assignment consists of a URI scheme type name and its associated description, a reference to the document that defines the URI scheme, and a reference to the document that defines the use of this URI scheme in BP endpoint IDs (including the CBOR representation of those endpoint IDs in transmitted bundles).

Value	Description	BP Utilization Reference	URI Definition Reference
0	Reserved	n/a	
1	dtn	RFC 0000	RFC 0000
2	ipn	RFC 0000	[RFC6260], RFC 0000
3-254	Unassigned	n/a	
255-65535	reserved	n/a	
>65535	open for private use	n/a	
		n/a	

Table 8: Bundle Protocol URI Scheme Type Registry

10.7. URI scheme "dtn"

In the "Uniform Resource Identifier (URI) Schemes" (uri-schemes) registry, IANA has updated the registration of the URI scheme with the string "dtn" as the scheme name, as follows:

URI scheme name: "dtn"

Status: permanent

Applications and/or protocols that use this URI scheme name: the Delay-Tolerant Networking (DTN) Bundle Protocol (BP).

Contact:

Scott Burleigh

Jet Propulsion Laboratory,

California Institute of Technology

scott.c.burleigh@jpl.nasa.gov

+1 (800) 393-3353

Change controller: IETF (iesg@ietf.org)

10.8. URI scheme "ipn"

In the "Uniform Resource Identifier (URI) Schemes" (uri-schemes) registry, IANA has updated the registration of the URI scheme with the string "ipn" as the scheme name, originally documented in RFC 6260 [RFC6260], as follows.

URI scheme name: "ipn"

Status: permanent

Applications and/or protocols that use this URI scheme name: the Delay-Tolerant Networking (DTN) Bundle Protocol (BP).

Contact:

Scott Burleigh

Jet Propulsion Laboratory,

California Institute of Technology

scott.c.burleigh@jpl.nasa.gov

+1 (800) 393-3353

Change controller: IETF (iesg@ietf.org)

11. References

11.1. Normative References

[BPSEC] Birrane, E. and K. McKeever, "Bundle Protocol Security Specification", RFC YYY1, DOI 10.17487/RFCYYY1, October 2021, <https://www.rfc-editor.org/info/rfcYYY1>.

[CRC16] International Telecommunication Union, "X.25: Interface between Data Terminal Equipment (DTE) and Data Circuit-terminating Equipment (DCE) for terminals operating in the packet mode and connected to public data networks by dedicated circuit", p. 9, Section 2.2.7.4, ITU-T Recommendation X.25, October 1996, <https://www.itu.int/rec/T-REC-X.25-199610-I/>.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/info/rfc2119>.

[RFC4960] Stewart, R., Ed., "Stream Control Transmission Protocol", RFC 4960, DOI 10.17487/RFC4960, September 2007, <https://www.rfc-editor.org/info/rfc4960>.

[RFC5234] Crocker, D., Ed. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", STD 68, RFC 5234, DOI 10.17487/RFC5234, January 2008, <https://www.rfc-editor.org/info/rfc5234>.

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/info/rfc8174>.

[RFC8949] Bormann, C. and P. Hoffman, "Concise Binary Object Representation (CBOR)", STD 94, RFC 8949, DOI 10.17487/RFC8949, December 2020, <https://www.rfc-editor.org/info/rfc8949>.

[SABR] Consultative Committee for Space Data Systems, "Schedule-Aware Bundle Routing", CCSDS Recommended Standard 734.3-B-1, July 2019, <https://public.ccsds.org/Pubs/734x3b1.pdf>.

[TCPCL] Sipos, B., Demmer, M., Ott, J., and S. Perreault, "Delay-Tolerant Networking TCP Convergence Layer Protocol Version 4", RFC YYY2, DOI 10.17487/RFCYYY2, October 2021, <https://www.rfc-editor.org/info/rfcYYY2>.

[URI] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, DOI 10.17487/RFC3986, January 2005, <https://www.rfc-editor.org/info/rfc3986>.

[URIREG] Thaler, D., Ed., Hansen, T., and T. Hardie, "Guidelines and Registration Procedures for URI Schemes", BCP 35, RFC 7595, DOI 10.17487/RFC7595, June 2015, <https://www.rfc-editor.org/info/rfc7595>.

11.2. Informative References

[ARCH] Cerf, V., Burleigh, S., Hooke, A., Torgerson, L., Durst, R., Scott, K., Fall, K., and H. Weiss, "Delay-Tolerant Networking Architecture", RFC 4838, DOI 10.17487/RFC4838, April 2007, <https://www.rfc-editor.org/info/rfc4838>.

[BIBE] Burleigh, S., "Bundle-in-Bundle Encapsulation", Work in Progress, Internet-Draft, draft-ietf-dtn-bibect-03, 18 February 2020, <https://datatracker.ietf.org/doc/html/draft-ietf-dtn-bibect-03>.

[RFC3987] Duerst, M. and M. Suignard, "Internationalized Resource Identifiers (IRIs)", RFC 3987, DOI 10.17487/RFC3987, January 2005, <https://www.rfc-editor.org/info/rfc3987>.

[RFC5050] Scott, K. and S. Burleigh, "Bundle Protocol Specification", RFC 5050, DOI 10.17487/RFC5050, November 2007, <https://www.rfc-editor.org/info/rfc5050>.

[RFC6255] Blanchet, M., "Delay-Tolerant Networking Bundle Protocol IANA Registries", RFC 6255, DOI 10.17487/RFC6255, May 2011, <https://www.rfc-editor.org/info/rfc6255>.

[RFC6257] Symington, S., Farrell, S., Weiss, H., and P. Lovell, "Bundle Security Protocol Specification", RFC 6257, DOI 10.17487/RFC6257, May 2011, <https://www.rfc-editor.org/info/rfc6257>.

[RFC6258] Symington, S., "Delay-Tolerant Networking Metadata Extension Block", RFC 6258, DOI 10.17487/RFC6258, May 2011, <https://www.rfc-editor.org/info/rfc6258>.

[RFC6259] Symington, S., "Delay-Tolerant Networking Previous-Hop Insertion Block", RFC 6259, DOI 10.17487/RFC6259, May 2011, <https://www.rfc-editor.org/info/rfc6259>.

[RFC6260] Burleigh, S., "Compressed Bundle Header Encoding (CBHE)", RFC 6260, DOI 10.17487/RFC6260, May 2011, <https://www.rfc-editor.org/info/rfc6260>.

[RFC7143] Chadalapaka, M., Satran, J., Meth, K., and D. Black, "Internet Small Computer System Interface (iSCSI) Protocol (Consolidated)", RFC 7143, DOI 10.17487/RFC7143, April 2014, <https://www.rfc-editor.org/info/rfc7143>.

[SIGC] Fall, K., "A Delay-Tolerant Network Architecture for Challenged Internets", SIGCOMM 2003, DOI 10.1145/863955.863960, August 2003, <https://dl.acm.org/doi/10.1145/863955.863960>.

Appendix A. Significant Changes from RFC 5050

Points on which this draft significantly differs from RFC 5050 include the following:

- Clarify the difference between transmission and forwarding.
- Migrate custody transfer to the bundle-in-bundle encapsulation specification [BIBE].
- Introduce the concept of "node ID" as functionally distinct from endpoint ID, while having the same syntax.
- Restructure primary block, making it immutable. Add optional CRC.
- Add optional CRCs to non-primary blocks.

- Add block ID number to canonical block format (to support BPsec).
- Add definition of bundle age extension block.
- Add definition of previous node extension block.
- Add definition of hop count extension block.
- Remove Quality of Service markings.
- Change from SDNVs to CBOR representation.
- Add lifetime overrides.
- Time values are denominated in milliseconds, not seconds.

Appendix B. For More Information

Appendix C. CDDL Expression

For informational purposes, Carsten Bormann and Brian Sipos have kindly provided an expression of the Bundle Protocol specification in the Concise Data Definition Language (CDDL). That CDDL expression is presented below. Note that wherever the CDDL expression is in disagreement with the textual representation of the BP specification presented in the earlier sections of this document, the textual representation rules.

```
bpv7_start = bundle / #6.55799(bundle)

; Times before 2000 are invalid

dtn-time = uint

; CRC enumerated type

crc-type = &(

  crc-none: 0,

  crc-16bit: 1,

  crc-32bit: 2

)

; Either 16-bit or 32-bit

crc-value = (bstr .size 2) / (bstr .size 4)

creation-timestamp = [

  dtn-time, ; absolute time of creation

  sequence: uint ; sequence within the time

]

eid = $eid .within eid-structure

eid-structure = [

  uri-code: uint,

  SSP: any

]

$eid /= [

  uri-code: 1,

  SSP: (tstr / 0)

]

$eid /= [

  uri-code: 2,

  SSP: [

    nodenum: uint,

    servicenum: uint
```

```
    ]
  ]
  ; The root bundle array
  bundle = [primary-block, *extension-block, payload-block]
  primary-block = [
    version: 7,
    bundle-control-flags,
    crc-type,
    destination: eid,
    source-node: eid,
    report-to: eid,
    creation-timestamp,
    lifetime: uint,
    ? (
      fragment-offset: uint,
      total-application-data-length: uint
    ),
    ? crc-value,
  ]
  bundle-control-flags = uint .bits bundleflagbits
  bundleflagbits = &(
    reserved: 21,
    reserved: 20,
    reserved: 19,
    bundle-deletion-status-reports-are-requested: 18,
    bundle-delivery-status-reports-are-requested: 17,
    bundle-forwarding-status-reports-are-requested: 16,
    reserved: 15,
    bundle-reception-status-reports-are-requested: 14,
```

```
    reserved: 13,
    reserved: 12,
    reserved: 11,
    reserved: 10,
    reserved: 9,
    reserved: 8,
    reserved: 7,
    status-time-is-requested-in-all-status-reports: 6,
    user-application-acknowledgement-is-requested: 5,
    reserved: 4,
    reserved: 3,
    bundle-must-not-be-fragmented: 2,
    payload-is-an-administrative-record: 1,
    bundle-is-a-fragment: 0
  )
  ; Abstract shared structure of all non-primary blocks
  canonical-block-structure = [
    block-type-code: uint,
    block-number: uint,
    block-control-flags,
    crc-type,
    ; Each block type defines the content within the bytestring
    block-type-specific-data,
    ? crc-value
  ]
  block-control-flags = uint .bits blockflagbits
  blockflagbits = &(
    reserved: 7,
    reserved: 6,
```

```
    reserved: 5,

    block-must-be-removed-from-bundle-if-it-cannot-be-processed: 4,

    reserved: 3,

    bundle-must-be-deleted-if-block-cannot-be-processed: 2,

    status-report-must-be-transmitted-if-block-cannot-be-
    processed: 1,

    block-must-be-replicated-in-every-fragment: 0

  )

  block-type-specific-data = bstr / #6.24(bstr)

  ; Actual CBOR data embedded in a bytestring, with optional tag to
  indicate so.

  ; Additional plain bstr allows ciphertext data.

  embedded-cbor<Item> = (bstr .cbor Item) / #6.24(bstr .cbor Item) /
  bstr

  ; Extension block type, which does not specialize other than the
  code/number

  extension-block = $extension-block .within canonical-block-
  structure

  ; Generic shared structure of all non-primary blocks

  extension-block-use<CodeValue, BlockData> = [

    block-type-code: CodeValue,

    block-number: (uint .gt 1),

    block-control-flags,

    crc-type,

    BlockData,

    ? crc-value

  ]

  ; Payload block type

  payload-block = payload-block-structure .within canonical-block-
  structure

  payload-block-structure = [

    block-type-code: 1,
```

```
    block-number: 1,
    block-control-flags,
    crc-type,
    $payload-block-data,
    ? crc-value
  ]
  ; Arbitrary payload data, including non-CBOR bytestring
  $payload-block-data /= block-type-specific-data
  ; Administrative record as a payload data specialization
  $payload-block-data /= embedded-cbor<admin-record>
  admin-record = $admin-record .within admin-record-structure
  admin-record-structure = [
    record-type-code: uint,
    record-content: any
  ]
  ; Only one defined record type
  $admin-record /= [1, status-record-content]
  status-record-content = [
    bundle-status-information,
    status-report-reason-code: uint,
    source-node-eid: eid,
    subject-creation-timestamp: creation-timestamp,
    ? (
      subject-payload-offset: uint,
      subject-payload-length: uint
    )
  ]
  bundle-status-information = [
    reporting-node-received-bundle: status-info-content,
```

```
    reporting-node-forwarded-bundle: status-info-content,
    reporting-node-delivered-bundle: status-info-content,
    reporting-node-deleted-bundle: status-info-content
  ]
  status-info-content = [
    status-indicator: bool,
    ? timestamp: dtn-time
  ]
  ; Previous Node extension block
  $extension-block /=
    extension-block-use<6, embedded-cbor<ext-data-previous-node>>
  ext-data-previous-node = eid
  ; Bundle Age extension block
  $extension-block /=
    extension-block-use<7, embedded-cbor<ext-data-bundle-age>>
  ext-data-bundle-age = uint
  ; Hop Count extension block
  $extension-block /=
    extension-block-use<10, embedded-cbor<ext-data-hop-count>>
  ext-data-hop-count = [
    hop-limit: uint,
    hop-count: uint
  ]
```

Acknowledgments

This work is freely adapted from RFC 5050, which was an effort of the Delay Tolerant Networking Research Group. The following DTNRG participants contributed significant technical material and/or inputs to that document: Dr. Vinton Cerf of Google; Scott Burleigh, Adrian Hooke, and Leigh Torgerson of the Jet Propulsion Laboratory; Michael Demmer of the University of

California at Berkeley; Robert Durst, Keith Scott, and Susan Symington of The MITRE Corporation; Kevin Fall of Carnegie Mellon University; Stephen Farrell of Trinity College Dublin; Howard Weiss and Peter Lovell of SPARTA, Inc.; and Manikantan Ramadas of Ohio University.

Authors' Addresses

Scott Burleigh
Jet Propulsion Laboratory, California Institute of Technology
4800 Oak Grove Dr.
Pasadena, CA 91109-8099
United States of America
Phone: +1 818 393 3353
Email: Scott.C.Burleigh@jpl.nasa.gov

Kevin Fall
Roland Computing Services
3871 Piedmont Ave. Suite 8
Oakland, CA 94611
United States of America
Email: kfall+rcs@kfall.com

Edward J. Birrane
Johns Hopkins University Applied Physics Laboratory
11100 Johns Hopkins Rd
Laurel, MD 20723
United States of America
Phone: +1 443 778 7423
Email: Edward.Birrane@jhuapl.edu