

Delay-Tolerant Networking
Internet-Draft
Intended status: Standards Track
Expires: October 17, 2020

E. Birrane
Johns Hopkins Applied Physics Laboratory
April 15, 2020

Asynchronous Management Protocol
draft-birrane-dtn-amp-08

Abstract

This document describes a binary encoding of the Asynchronous Management Model (AMM) and a protocol for the exchange of these encoded items over a network. This Asynchronous Management Protocol (AMP) does not require transport-layer sessions, operates over unidirectional links, and seeks to reduce the energy and compute power necessary for performing network management on resource constrained devices.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <https://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on October 17, 2020.

Copyright Notice

Copyright (c) 2020 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<https://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of

the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction	3
2. Requirements Language	3
3. Scope	3
3.1. Protocol Scope	3
3.2. Specification Scope	4
4. Terminology	4
5. Constraints and Assumptions	4
6. Technical Notes	5
7. AMP-Specific Concepts	6
7.1. Nicknames (NN)	6
7.1.1. Motivation for Compression	6
7.1.2. ADM Enumeration	7
7.1.3. ADM Template Collection Enumeration	8
7.1.4. Nickname Definition	9
7.1.5. ADM Enumeration Considerations	9
8. Encodings	10
8.1. CBOR Considerations	10
8.2. AMM Type Encodings	10
8.2.1. Primitive Types	10
8.2.2. Derived Types	11
8.2.3. Collections	14
8.3. AMM Resource Identifier (ARI)	19
8.3.1. Encoding ARIs of Type LITERAL	19
8.3.2. Encoding Non-Literal ARIs	20
8.4. ADM Object Encodings	23
8.4.1. Externally Defined Data (EDD)	23
8.4.2. Constants (CONST)	24
8.4.3. Controls (CTRL)	24
8.4.4. Macros (MAC)	25
8.4.5. Operators (OPER)	26
8.4.6. Report Templates (RPTT)	26
8.4.7. Report (RPT)	27
8.4.8. State-Based Rules (SBR)	28
8.4.9. Table Templates (TBLT)	30
8.4.10. Tables (TBL)	30
8.4.11. Time-Based Rules (TBR)	31
8.4.12. Variables (VAR)	33
9. Functional Specification	33
9.1. AMP Message Summary	33
9.2. Message Group Format	34
9.3. Message Format	35
9.4. Register Agent	37
9.5. Report Set	37

9.6. Perform Control	38
9.7. Table Set	38
10. IANA Considerations	39
11. Security Considerations	39
12. Implementation Notes	39
13. References	40
13.1. Informative References	40
13.2. Normative References	40
Appendix A. Acknowledgements	40
Author's Address	40

1. Introduction

Network management in challenged and resource constrained networks must be accomplished differently than the network management methods in high-rate, high-availability networks. The Asynchronous Management Architecture (AMA) [I-D.birrane-dtn-ama] provides an overview and justification of an alternative to "synchronous" management services such as those provided by NETCONF. In particular, the AMA defines the need for a flexible, robust, and efficient autonomy engine to handle decisions when operators cannot be active in the network. The logical description of that autonomous model and its major components is given in the AMA Data Model (ADM) [I-D.birrane-dtn-adm].

The ADM presents an efficient and expressive autonomy model for the asynchronous management of a network node, but does not specify any particular encoding. This document, the Asynchronous Management Protocol (AMP), provides a binary encoding of AMM objects and specifies a protocol for the exchange of these encoded objects.

2. Requirements Language

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].

3. Scope

3.1. Protocol Scope

The AMP provides data monitoring, administration, and configuration for applications operating above the data link layer of the OSI networking model. While the AMP may be configured to support the management of network layer protocols, it also uses these protocol stacks to encapsulate and communicate its own messages.

It is assumed that the protocols used to carry AMP messages provide addressing, confidentiality, integrity, security, fragmentation/reassembly, and other network functions. Therefore, these items are outside of the scope of this document.

3.2. Specification Scope

This document describes the format of messages used to exchange data models between managing and managed devices in a network. The rationale for this type of exchange is outside of the scope of this document and is covered in [I-D.birrane-dtn-ama]. The description and explanation of the data models exchanged is also outside of the scope of this document and is covered in [I-D.birrane-dtn-adm].

This document does not address specific configurations of AMP-enabled devices, nor does it discuss the interface between AMP and other management protocols.

4. Terminology

Note: The terms "Actor", "Agent", "Application Data Model", "Externally Defined Data", "Variable", "Control", "Literal", "Macro", "Manager", "Report Template", "Report", "Table", "Constant", "Operator", "Time-Based Rule" and "State-Based Rule" are used without modification from the definitions provided in [I-D.birrane-dtn-ama].

5. Constraints and Assumptions

The desirable properties of an asynchronous management protocol, as specified in the AMA, are summarized here to represent design constraints on the AMP specification.

- o Intelligent Push of Information - Nodes in a challenged network cannot guarantee concurrent, bi-directional communications. Some links between nodes may be strictly unidirectional. AMP Agents "push" data to Managers rather than Managers "pulling" data from Agents.
- o Small Message Sizes - Smaller messages require smaller periods of viable transmission for communication, incur less retransmission cost, and consume fewer resources when persistently stored en route in the network. AMP minimizes message size wherever practical, to include binary data representations and predefined data definitions and templates.
- o Absolute and Custom Data Identification - All data in the system must be uniquely addressable, to include operator-specified information. AMP provides a compact encoding for identifiers.

- o Autonomous, Stateless Operation - There is no reliable concept of session establishment or round-trip data exchange in asynchronous networks. AMP is designed to be stateless. Where helpful, AMP provides mechanisms for transactional ordering of commands within a single AMP protocol data unit, but otherwise degrades gracefully when nodes in the network diverge in their configuration.

6. Technical Notes

- o Unless otherwise specified, multi-byte values in this specification are expected to be transmitted in network byte order (Big Endian).
- o Character encodings for all text-based data types will use UTF-8 encodings.
- o All AMP encodings are self-terminating. This means that, given an indefinite-length octet stream, each encoding can be unambiguously decoded from the stream without requiring additional information such as a length field separate from the data type definition.
- o This specification uses the term OCTETS to refer to a sequence of one or more related BYTE values. There is no implied structure associated with OCTETS, meaning they do not encode a length value or utilize a terminator character. While OCTETS may contain CBOR-encoded values, the OCTETS sequence itself is not encoded as a CBOR structure.
- o If an OCTETS sequence is included as an element of a CBOR array then the sequence MUST be considered as a single array element when determining the size of the array.
- o Bit-fields in this document are specified with bit position 0 holding the least-significant bit (LSB). When illustrated in this document, the LSB appears on the right.
- o In order to describe the encoding of data models specified in [I-D.birrane-dtn-adm], this specification must refer to both the data object being encoded and to the encoding of that data object. When discussing the encoded version of a data object, this specification uses the notation "E(data_object)" where E() refers to a conceptual encoding function. This notation is only provided as a means of clarifying the text and imposes no changes to the actual wire coding. For example, this specification will refer to the "macro" data object as "Macro" and to the encoding of a Macro as "E(Macro)".

- o Illustrations of fields in this specification consist of the name of the field, the type of the field between []'s, and if the field is optional, the text "(opt)".
Field order is deterministic and, therefore, fields MUST be transmitted in the order in which they are specified. In cases where an optional field is not present, then the next field will be considered for transmission.
An example is shown in Figure 1 below. In this illustration two fields (Field 1 and Field 2) are shown, with Field 1 of Type 1 and Field 2 of Type 2. Field 2 is also listed as being optional. Byte fields are shown in order of receipt, from left-to-right. Therefore, when transmitted on the wire, Field 1 will be received first, followed by Field 2 (if present).

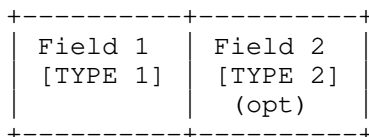


Figure 1: Byte Field Formatting Example

When types are documented in this way, the type always refers to the encoding of that type. The E() notation is not used as it is to be inferred from the context of the illustration.

7. AMP-Specific Concepts

The AMP specification provides an encoding of objects comprising the AMM. As such, AMP defines very few structures of its own. This section identifies those data structures that are unique to the AMP and required for it to perform appropriate and efficient encodings of AMM objects.

7.1. Nicknames (NN)

In the AMP, a "Nickname" (NN) is used to reduce the overall size of the encoding of ARIs that are defined in the context of an ADM. A NN is calculated as a function of an ADM Moderated Namespace and the type of object being identified.

7.1.1. Motivation for Compression

As identifiers, ARIs are used heavily in AMM object definitions, particularly in those that define collections of objects. This makes encoding ARIs an important consideration when trying to optimize the size of AMP message.

Additionally, the majority of ARIs are defined in the context of an ADM. Certain AMM objects types (EDDs, OPs, CTRLs, TBLTs) can only be defined in the context of an ADM. Other object types (VARs, CONSTs, RPTTs) may have common, useful objects defined in an ADM as well. The structure of an ADM, to include its use of a Moderated Namespace and collections by object type, provide a regular structure that can be exploited for creating a compact representation.

In particular, as specified in [I-D.birrane-dtn-adm], ARIs can be grouped by (1) their namespace and (2) the type of AMM object being identified. For example, consider the following ARIs of type EDD defined in ADM1 with a Moderated Namespace of "/DTN/ADM1/".

```
ari:/DTN/ADM1/Edd.item_1 ari:/DTN/ADM1/Edd.item_2 ... ari:/DTN/ADM1/
Edd.item_1974
```

In this case, the namespace (/DTN/ADM1/) and the object type (Edd) are good candidates for enumeration because their string encoding is very verbose and their information follows a regular structure shared across multiple ARIs. Separately, the string representation of object names (item_1, item_2, etc...) may be very verbose and they are also candidates for enumeration as they occupy a particular ADM object type in a particular order as published in the ADM.

7.1.2. ADM Enumeration

Any ARI defined in an ADM exists in the context of a Moderated Namespace. These namespaces provide a unique string name for the ADM. However, ADMs can also be assigned a unique enumeration by the same moderating entities that ensure namespace uniqueness.

An ADM enumeration is an unsigned integer in the range of 0 to $(2^{64})/20$. This range provides effective support for thousands of trillions of ADMs.

The formal set of ADMs, similar to SNMP MIBs and NETCONF YANG models, will be moderated and published. Additionally, a set of informal ADMs may be developed on a network-by-network or on an organization-by-organization bases.

Since informal ADMs exist within a predefined context (a network, an organization, or some other entity) they do not have individual ADM enumerations and are assigned the special enumeration "0". ARIs that are not defined in formal ADMs rely on other context information to help with their encoding (see Section 8.3).

7.1.3. ADM Template Collection Enumeration

The ADM template presented in [I-D.birrane-dtn-adm] defines a series of object collections for the specification of various AMM objects. Enumerating these collections in a standard way allows for their compressed representation in the context of nicknames for objects stored in these collections.

The enumeration of ADM Template collections is provided in Table 1 below.

AMM Object Type	Enumeration
CONST	0
CTRL	1
EDD	2
MAC	3
OPER	4
RPTT	5
SBR	6
TBLT	7
TBR	8
VAR	9
metadata	10
reserved	11-19

Table 1: ADM Type Enumerations

NOTE: Collection enumerations are different from AMM object types. For example, the enumeration for the VAR collection (9) in an ADM is different from the VAR object type (12).

7.1.4. Nickname Definition

As an enumeration, a Nickname is captured as a 64-bit unsigned integer (UVAST) calculated as a function of the ADM enumeration and the ADM type enumeration, as follows.

$$NN = ((ADM\ Enumeration) * 20) + (ADM\ Object\ Type\ Enumeration)$$

Considering the example set of ARIs from Section 7.1.1, assuming that ADM1 has ADM enumeration 9 and given that objects in the example were of type EDD, the NN for each of the 1974 items would be: $(9 * 20) + 2 = 182$. In this particular example, the ARI `"/DTN/ADM1/Edd.item_1974"` can be encoded in 5 bytes: two bytes to CBOR encode the nickname (182) and 3 bytes to CBOR encode the item's offset in the Edd collection (1974).

7.1.5. ADM Enumeration Considerations

The assignment of formal ADM enumerations SHOULD take into consideration the nature of the applications and protocols to which the ADM applies. Those ADMs that are likely to be used in challenged networks SHOULD be allocated low enumeration numbers (e.g. those that will fit into 1-2 bytes) while ADMs that are likely to only be used in well resourced networks SHOULD be allocated higher enumeration numbers. It SHOULD NOT be the case that ADM enumerations are allocated on a first-come, first-served basis. It is recommended that ADM enumerations should be labeled based on the number of bytes of the Nickname as a function of the size of the ADM enumeration. These labels are shown in Table 2.

ADM Enum	NN Size	Label	Comment
0x1 - 0xCCC	1-2 Bytes	Challenged Networks	Constraints imposed by physical layer and power.
0xCCD - 0xCCCCCCC	3-4 Bytes	Congested Networks	Constraints imposed by network traffic.
>=0xCCCCCCD	5-8 Bytes	Resourced Networks	Generally unconstrained networks.

Table 2: ADM Enumerations Labels

8. Encodings

This section describes the binary encoding of logical data constructs using the Concise Binary Object Representation (CBOR) defined in [RFC7049].

8.1. CBOR Considerations

The following considerations act as guidance for CBOR encoders and decoders implementing the AMP.

- o All AMP encodings are of definite length and, therefore, indefinite encodings MUST NOT be used.
- o AMP encodings MUST NOT use CBOR tags. Identification mechanisms in the AMP capture structure and other information such that tags are not necessary.
- o Canonical CBOR MUST be used for all encoding. All AMP CBOR decoders MUST run in strict mode.
- o Because AMA objects are self-delineating they can be serialized into, or deserialized from, OCTETS. CBOR containers such as BYTESTR and TXTSTR that encode length fields are only useful for data that is not self-delineating, such as name fields. Encoding self-delineating objects into CBOR containers reduced efficiency as length fields would then be added to data that does not require a length field for processing.
- o Encodings MUST result in smallest data representations. There are several cases where the AMM defines types with less granularity than CBOR. For example, AMM defines the UINT type to represent unsigned integers up to 32 bits in length. CBOR supports separate definitions of unsigned integers of 8, 16, or 32 bits in length. In cases where an AMM type MAY be encoded in multiple ways in CBOR, the smallest data representation MUST be used. For example, UINT values of 0-255 MUST be encoded as a uint8_t, and so on.

8.2. AMM Type Encodings

8.2.1. Primitive Types

The AMP encodes AMM primitive types as outlined in Table 3.

AMM Type	CBOR Major Type	Comments
BYTE	unsigned int or byte string	BYTES are individually encoded as unsigned integers unless they are defined as part of a byte string, in which case they are encoded as a single byte in the byte string.
INT	unsigned integer or negative integer	INTs are encoded as positive or negative integers from (u)int8_t up to (u)int32_t.
UINT	unsigned integer	UINTs are unsigned integers from uint8_t up to uint32_t.
VAST	unsigned integer or negative integer	VASTs are encoded as positive or negative integers up to (u)int64_t.
UVAST	unsigned integer	VASTs are unsigned integers up to uint64_t.
REAL32	floating point	Up to an IEEE-754 Single Precision Float.
REAL64	floating point	Up to an IEEE-754 Double Precision Float.
STRING	text string	Uses CBOR encoding unmodified.
BOOL	Simple Value	0 is considered FALSE. Any other value is considered TRUE.

Table 3: Standard Numeric Types

8.2.2. Derived Types

This section provides the CBOR encodings for AMM derived types.

8.2.2.1. Byte String Encoding

The AMM derived type Byte String (BYTESTR) is encoded as a CBOR byte string.

8.2.2.2. Time Values (TV) and Timestamps (TS)

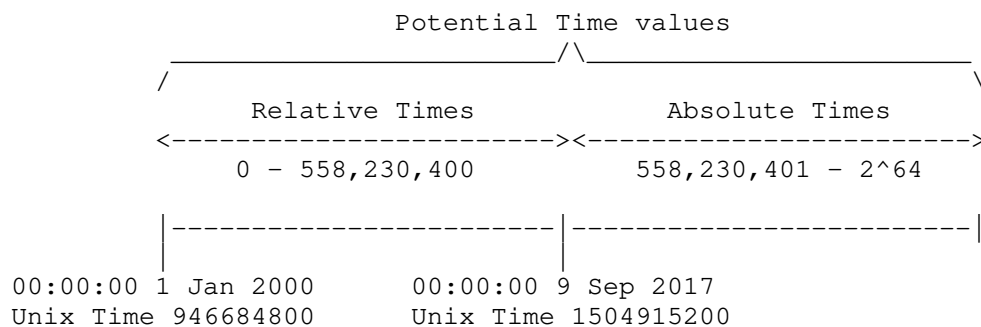
An TV is encoded as a UVAST. Similarly, a TS is also encoded as a UVAST since a TS is simply an absolute TV.

Rather than define two separate encodings for TVs (one for absolute TVs and one for relative TVs) a single, unambiguous encoding can be generated by defining a Relative Time Epoch (RTE) and interpreting the type of TV in relation to that epoch. Time values less than the RTE MUST be interpreted as relative times. Time values greater than or equal to the RTE MUST be interpreted as absolute time values.

A relative TV is encoded as the number of seconds after an initiating event. An absolute TV (and TS) is encoded as the number of seconds that have elapsed since 1 Jan 2000 00:00:00 (Unix Time 946684800).

The RTE is defined as the timestamp value for September 9th, 2017 (Unix time 1504915200). Since TS values are the number of seconds since 1 Jan 2000 00:00:00, the RTE as a TS value is $1504915200 - 946684800 = 558230400$.

The potential values of TV, and how they should be interpreted as relative or absolute is illustrated below.



For example, a time value of "10" is a relative time representing 10 seconds after an initiating event. A time value of "600,000,000" refers to Saturday, 5 Jan, 2019 10:40:00.

NOTE: Absolute and relative times are interchangeable. An absolute time can be converted into a relative time by subtracting the current time from the absolute time. A relative time can be converted into

an absolute time by adding to the relative time the timestamp of its relative event. A pseudo-code example of converting a relative time to an absolute time is as follows, assuming that current-time is expressed in Unix Epoch time.

```
IF (time_value <= 558230400) THEN
  absolute_time = (event_time - 946684800) + time_value
ELSE
  absolute_time = time_value
```

8.2.2.3. Type-Name-Value (TNV)

TNV values are encoded as a CBOR array that comprises four distinct pieces of information: a set of flags, a type, an optional name, and an optional value. In the E(TNV) the flag and type information are compressed into a single value. The CBOR array MUST have length 1, 2, or 3 depending on the number of optional fields appearing in the encoding. The E(TNV) format is illustrated in Figure 2.

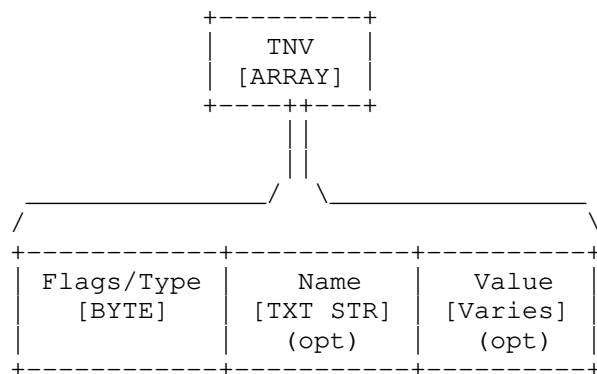


Figure 2: E(TNV) Format

The E(TNV) fields are defined as follows.

Flags/Type

The first byte of the E(TNV) describes the type associated with the TNV and which optional components are present. The layout of this byte is illustrated in Figure 3.

E(TNV) Flag/Type Byte Format

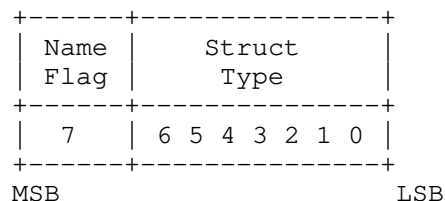


Figure 3

Name Flag

This flag indicates that the TNV contains a name field. When set to 1 the Name field **MUST** be present in the E(TNV). When set to 0 the Name field **MUST NOT** be present in the E(TNV).

Struct Type

This field lists the type associated with this TNV and **MUST** contain one of the types defined in [I-D.birrane-dtn-adm] with the exception that the type of a TNV **MUST NOT** be a TNV.

Name

This optional field captures the human-readable name for the TNV encoded as a CBOR text string. If there are 3 elements in the TNV array **OR** there are 2 elements in the array and the Name Flag is set, then this field **MUST** be present. Otherwise, this field **MUST NOT** be present.

Value

This optional field captures the encoded value associated with this TNV. The value is encoded in accordance with AMP rules for encoding of items of the type of this TNV. If there are 3 elements in the TNV array **OR** there are 2 elements in the array and the Name Flag is not set, then this field **MUST** be present. Otherwise, this field **MUST NOT** be present.

8.2.3. Collections

8.2.3.1. Type-Name-Value Collection (TNVC)

A TNV Collection (TNVC) is an ordered set of TNVs with special semantics for more efficiently encoding sets of TNVs with identical attributes.

A TNV, defined in Section 8.2.2.3, consists of three distinct components: a type, a name, and a value. When all of the TNVs in the TNVC have the same format (such as they all include type information) then the encoding of the TNVC can use this information to save encoding space and make processing more efficient. In cases when all TNVs have the same format, the types (if present), names (if present), and values (if present) are separated into their own arrays for individual processing with type information (if present) always appearing first.

Extracting type information to the "front" of the collection optimizes the performance of type validators. A validator can inspect the first array to ensure that element values match type expectations. If type information were distributed throughout the collection, as in the case with the TNVC, a type validator would need to scan through the entire set of data to validate each type in the collection.

A TNVC is encoded as a sequence of at least 1 octet, where the single required octet includes the flag BYTE representing the optional portions of the collection that are present. If the flag BYTE indicates an empty collection there will be no following octets. The format of a TNVC is illustrated in Figure 4.

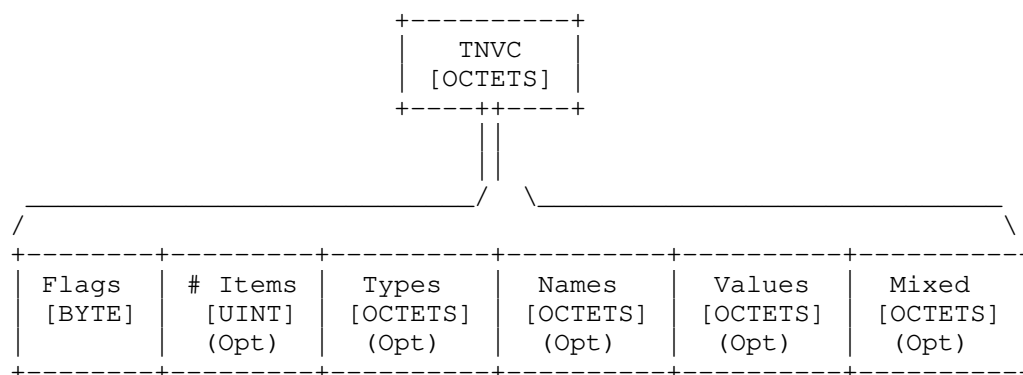


Figure 4: E(TNVC) Format

The E(TNVC) fields are defined as follows.

Flags

The first byte of the E(TNVC) describes which optional portions of a TNV will be present for each TNV in the collection.

If all non-reserved flags have the value 0 then the TNVC represents an empty collection, in which case no other information is provided for the E(TNVC).

The layout of this byte is illustrated in Figure 5.

E(TNV) Flag Byte Format

Reserved Flags	Mix Flag	Type Flag	Name Flag	Val Flag
7-4	3	2	1	0
MSB			LSB	

Figure 5

Mixed Flag

This flag indicates that the set of TNVs in the collection do not all share the same properties and, therefore, the collection is a mix of different types of TNV. When set to 1 the E(TNVC) MUST contain the Mixed Values field and all other flags in this byte MUST be set to 0. When set to 0 the E(TNVC) MUST NOT contain the Mixed Values field.

Type Flag

This flag indicates whether each TNV in the collection has type information associated with it. When set to 1 the E(TNVC) MUST contain type information for each TNV. When set to 0, type information MUST NOT be present.

Name Flag

This flag indicates whether each TNV in the collection has name information associated with it. When set to 1 the E(TNVC) MUST contain name information for each TNV. When set to 0, name information MUST NOT be present.

Value Flag

This flag indicates whether each TNV in the collection has value information associated with it. When set to 1 the E(TNVC) MUST contain value information for each TNV. When set to 0, value information MUST NOT be present.

Items

The number of items field lists the number of items that are contained in the TNVC. Each of the types, names, and values sequences (if present) MUST have exactly this number of entries in them. This field MUST be present in the E(TNVC) when any one of the non-reserved bits of the Flag Byte are set to 1.

Types

The types field is encoded as an OCTETS sequence where the Nth byte in the sequence represents the type for the Nth TNV in the collection. This field MUST be present in the E(TNVC) when the Type Flag is set to 1 and MUST NOT be present otherwise. If present, this field MUST contain exactly the same number of types as number of items in the TNVC.

Names

The names field is encoded as an OCTETS sequence containing the names of the TNVs in the collection. Each name is encoded as a CBOR string, with the Nth CBOR string representing the name of the Nth TNV in the collection. This field MUST be present in the E(TNVC) when the Names Flag is set to 1 and MUST NOT be present otherwise. If present, this field MUST contain exactly the same number of CBOR strings as number of items in the TNVC.

Values

The values field is encoded as an OCTETS sequence containing the values of TNVs in the collection.
If the Type Flag is set to 1 then each entry will be encoded in accordance with the corresponding index in the type field. For example, the 1st value will be encoded using the encoding rules for the first byte in the type OCTETS sequence.
If the Type Flag is set to 0 then the values will be encoded as native CBOR types. CBOR types do not have a one-to-one mapping with AMP types and it is the responsibility of the transmitting AMP actor and the receiving AMP actor to determine how to map these to AMP types. This field MUST be present in the E(TNVC) when the Value Flag is set to 1 and MUST NOT be present otherwise. If present, this field MUST contain exactly the same number of values as number of items in the TNVC.

Mixed

The mixed field is encoded as an OCTETS sequence containing a series of E(TNV) objects. This field MUST be present when the Mixed Flag is set to 1 and MUST NOT be present otherwise. If present, this field MUST contain exactly the same number of E(TNV) objects as numnber of items in the TNVC.

8.2.3.2. ARI Collections (AC)

An ARI collection is an ordered collection of ARI values. It is encoded as a CBOR array with each element being an encoded ARI, as illustrated in Figure 6.

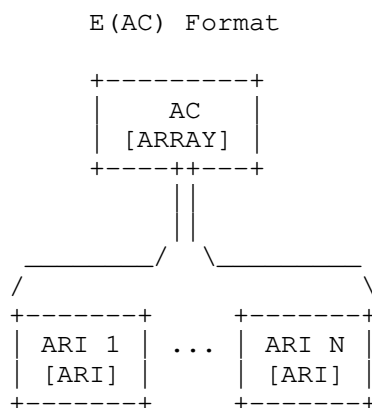


Figure 6

8.2.3.3. Expressions (EXPR)

The Expression object encapsulates a typed postfix expression in which each operator **MUST** be of type OPER and each operand **MUST** be the typed result of an operator or one of EDD, VAR, LIT, or CONST.

The Expression object is encoded as an OCTETS sequence whose format is illustrated in Figure 7.

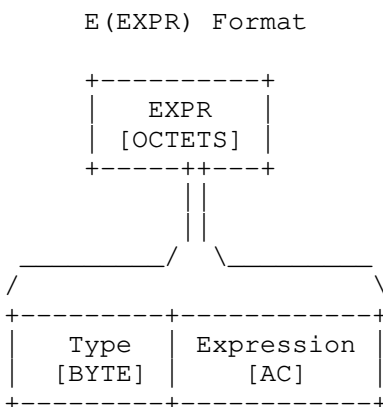


Figure 7

Type

The enumeration representing the type of the result of the evaluated expression. This type **MUST** be defined in [I-D.birrane-dtn-adm] as a "Primitive Type".

Expression

An expression is represented in the AMP as an ARI collection, where each ARI in the ordered collection represents either an operand or operator in postfix form.

8.3. AMM Resource Identifier (ARI)

The ARI, as defined in [I-D.birrane-dtn-adm], identifies an AMM object. There are two kinds of objects that can be identified in this scheme: literal objects (of type LIT) and all other objects.

8.3.1. Encoding ARIs of Type LITERAL

A literal identifier is one that is literally defined by its value, such as numbers (0, 3.14) and strings ("example"). ARIs of type LITERAL do not have issuers or nicknames or parameters. They are simply typed basic values.

The E(ARI) of a LIT object is encoded as an OCTETS sequence and consists of a mandatory flag BYTE and the value of the LIT.

The E(ARI) structure for LIT types is illustrated in Figure 8.

E(ARI) Literal Format

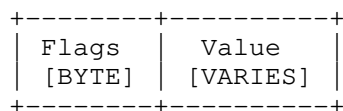


Figure 8

These fields are defined as follows.

Flags

The Flags byte identifies the object as being of type LIT and also captures the primitive type of the following value. The layout of this byte is illustrated in Figure 9.

E(ARI) Literal Flag Byte Format

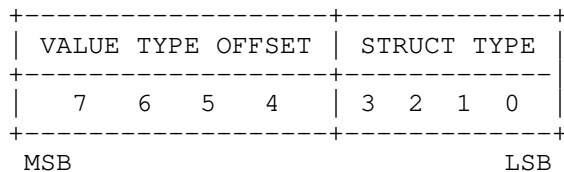


Figure 9

Value Type Offset

The high nibble of the flag byte contains the offset into the Primitive Types enumeration defined in [I-D.birrane-dtn-adm]. An offset of 0 represents the first defined Primitive Type. An offset of 1 represents the second defined Primitive Type, and so on. An offset into the data types field is used to ensure that the type value fits into a nibble.

Structure Type

The lower nibble of the flag byte identifies the type of AMM Object being identified by the ARI. In this instance, this value MUST be LIT, as defined in [I-D.birrane-dtn-adm].

Value

This field captures the CBOR encoding of the value. Values are encoded according to their Value Type as specified in the flag byte in accordance with the encoding rules provided in Section 8.2.1.

8.3.2. Encoding Non-Literal ARIs

All other ARIs are defined in the context of AMM objects and may contain parameters and other meta-data. The AMP, as a machine-to-machine binary encoding of this information removes human-readable information such as Name and Description from the E(ARI). Additionally, this encoding adds other information to improve the efficiency of the encoding, such as the concept of Nicknames, defined in Section 7.1.

The E(ARI) is encoded as an OCTETS sequence and consists of a mandatory flag byte, an encoded object name, and optional annotations to assist with filtering, access control, and parameterization. The E(ARI) structure is illustrated in Figure 10.

E(ARI) General Format

Flags [BYTE]	NN [UVAST] (opt)	Name [BYTESTR]	Parms [TNVC] (opt)	Issuer [BYTESTR] (opt)	Tag [BYTESTR] opt)
-----------------	------------------------	-------------------	--------------------------	------------------------------	--------------------------

Figure 10

These fields are defined as follows.

Flags

Flags describe the type of structure and which optional fields are present in the encoding. The layout of the flag byte is illustrated in Figure 11.

E(ARI) General Flag Byte Format

NN	PARM	ISS	TAG	STRUCT	TYPE
7	6	5	4	3	2 1 0

MSBLSB

Figure 11

Nickname (NN)

This flag indicates that ADM compression is used for this E(ARI). When set to 1 the Nickname field **MUST** be present in the E(ARI). When set to 0 the Nickname field **MUST NOT** be present in the E(ARI). When an ARI is user-defined, there are no semantics for Nicknames and, therefore, this field **MUST** be 0 when the Issuer flag is set to 1. Implementations **SHOULD** use Nicknames whenever possible to reduce the size of the E(ARI).

Parameters Present (PARM)

This flag indicates that this ARI can be parameterized and that parameter information is included in the E(ARI). When set to 1 the Parms field **MUST** be present in the E(ARI). When set to 0 the Parms field **MUST NOT** be present in the E(ARI).

Issuer Present (ISS)

This flag indicates that this ARI is defined in the context of a specific issuing entity. When set to 1 the Issuer field MUST be present in the E(ARI). When set to 0 the Issuer field MUST NOT be present in the E(ARI).

Tag Present (TAG)

This flag indicates that the ARI is defined in the context of a specific issuing entity and that issuing entity adds additional information in the form of a tag. When set to 1 the Tag field MUST be present in the E(ARI). When set to 0 the Tag field MUST NOT be present in the E(ARI). This flag MUST be set to 0 if the Issuer Present flag is set to 0.

Structure Type (STRUCT TYPE)

The lower nibble of the E(ARI) flag byte identifies the kind of structure being identified. This field MUST contain one of the AMM object types defined in [I-D.birrane-dtn-adm].

Nickname (NN)

This optional field contains the Nickname as calculated according to Section 7.1.

Object Name

This mandatory field contains an encoding of the ADM object. For elements defined in an ADM Template (e.g., where the Issuer Flag is set to 0) this is the 0-based index into the ADM collection holding this element. For all user-defined ADM objects, (e.g., where the Issuer Flag is set to 1) this value is as defined by the Issuing organization.

Parameters

The parameters field is represented as a Type Name Value Collection (TNVC) as defined in Section 8.2.3.1. The overall number of items in the collection represents the number of parameters. The types of the TNVC represent the types of each parameter, with the first listed type associated with the first parameter, and so on. The values, if present, represent the values of the parameters, with the first listed value being the value of the first parameter, and so on.

Issuer

This is a binary identifier representing a predetermined issuer name. The AMP protocol does not parse or validate this identifier, using it only as a distinguishing bit pattern to ensure uniqueness. This value, for example, may

come from a global registry of organizations, an issuing node address, or some other network-unique marking. The issuer field MUST NOT be present for any ARI defined in an ADM.

Tag

A value used to disambiguate multiple ARIs with the same Issuer. The definition of the tag is left to the discretion of the Issuer. The Tag field MUST be present if the Tag Flag is set in the flag byte and MUST NOT be present otherwise.

8.4. ADM Object Encodings

The autonomy model codified in [I-D.birrane-dtn-adm] comprises multiple individual objects. This section describes the CBOR encoding of these objects.

Note: The encoding of an object refers to the way in which the complete object can be encoded such that the object as it exists on a Manager may be re-created on an Agent, and vice-versa. In cases where both a Manager and an Agent already have the definition of an object, then only the encoded ARI of the object needs to be communicated. This is the case for all objects defined in a published ADM and any user-defined object that has been synchronized between an Agent and Manager.

8.4.1. Externally Defined Data (EDD)

Externally defined data (EDD) are solely defined in the ADMs for various applications and protocols. EDDs represent values that are calculated external to an AMA Agent, such as values measured by firmware.

The representation of these data is simply their identifying ARIs. The representation of an EDD is illustrated in Figure 12.

E(EDD) Format

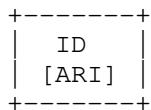


Figure 12

ID

This is the ARI identifying the EDD. Since EDDs are always defined solely in the context of an ADM, this ARI MUST NOT

have an ISSUER field and MUST NOT have a TAG field. This ARI may be defined with parameters.

8.4.2. Constants (CONST)

Unlike Literals, a Constant is an immutable, typed, named value. Examples of constants include PI to some number of digits or the UNIX Epoch.

Since ADM definitions are preconfigured on Agents and Managers in an AMA, the type information for a given Constant is known by all actors in the system and the encoding of the Constant needs to only be the name of the constant as the Manager and Agent can derive the type and value from the unique Constant name.

The format of a Constant is illustrated in Figure 13.

E(CONST) Format

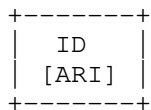


Figure 13

ID

This is the ARI identifying the Constant. Since Constant definitions are always provided in an ADM, this ARI MUST NOT have an ISSUER field and MUST NOT have a TAG field. The ARI MUST NOT have parameters.

8.4.3. Controls (CTRL)

A Control represents a pre-defined and optionally parameterized opcode that can be run on an Agent. Controls in the AMP are always defined in the context of an AMA; there is no concept of an operator-defined Control. Since Controls are pre-configured in Agents and Managers as part of ADM support, their representation is the ARI that identifies them, similar to EDDs.

The format of a Control is illustrated in Figure 14.

E(CTRL) Format

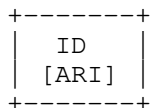


Figure 14

ID

This is the ARI identifying the Control. This ARI MUST NOT have an ISSUER field and MUST NOT have a TAG field. This ARI may have parameters.

8.4.4. Macros (MAC)

Macros in the AMP are ordered collections of ARIs (an AC) that contain Controls or other Macros. When run by an Agent, each ARI in the AC MUST be run in order.

Any AMP implementation MUST allow at least 4 levels of Macro nesting. Implementations MUST prevent recursive nesting of Macros.

The ARI associated with a Macro MAY contain parameters. Each parameter present in a Macro ARI MUST contain type, name, and value information. Any Control or Macro encapsulated within a parameterized Macro MAY also contain parameters. If an encapsulated object parameter contains only name information, then the parameter value MUST be taken from the named parameter provided by the encapsulating Macro. Otherwise, the value provided to the object MUST be used instead.

The format of a Macro is illustrated in Figure 15.

E(MAC) Format

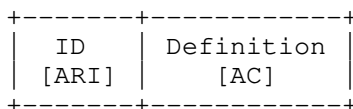


Figure 15

ID

This is the ARI identifying the Macro. When a Macro is defined in an ADM this ARI MUST NOT have an ISSUER field and MUST NOT have a TAG field. When the Macro is defined outside

of an ADM, the ARI MUST have an ISSUER field and MAY have a TAG field.

Definition

This is the ordered collection of ARIs that identify the Controls and other Macros that should be run as part of running this Macro.

8.4.5. Operators (OPER)

Operators are always defined in the context of an ADM. There is no concept of a user-defined operator, as operators represent mathematical functions implemented by the firmware on an Agent. Since Operators are preconfigured in Agents and Managers as part of ADM support, their representation is simply the ARI that identifies them.

The ADM definition of an Operator MUST specify how many parameters are expected and the expected type of each parameter. For example, the unary NOT Operator ("!") would accept one parameter. The binary PLUS Operator ("+") would accept two parameters. A custom function to calculate the average of the last 10 samples of a data item should accept 10 parameters.

Operators are always evaluated in the context of an Expression. The encoding of an Operator is illustrated in Figure 16.

E(OP) Format

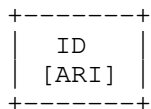


Figure 16

ID

This is the ARI identifying the Operator. Since Operators are always defined solely in the context of an ADM, this ARI MUST NOT have an ISSUER field and MUST NOT have a TAG field.

8.4.6. Report Templates (RPTT)

A Report Template is an ordered collection of identifiers that describe the order and format of data in any Report built in compliance with the template. A template is a schema for a class of reports. It contains no actual values and may be defined in a formal ADM or configured by users in the context of a network deployment.

The encoding of a RPTT is illustrated in Figure 17.

E(RPTT) Format

ID	Contents
[ARI]	[AC]

Figure 17

ID

This is the ARI identifying the report template.

Contents

This is the ordered collection of ARIs that define the template.

8.4.7. Report (RPT)

A Report is a set of data values populated using a given Report Template. While Reports do not contain name information, they MAY contain type information in cases where recipients wish to perform type validation prior to interpreting the Report contents in the context of a Report Template. Reports are "anonymous" in the sense that any individual Report does not contain a unique identifier. Reports can be differentiated by examining the combination of (1) the Report Template being populated, (2) the time at which the Report was populated, and (3) the Agent producing the report.

A Report object is comprised of the identifier of the template used to populate the report, an optional timestamp, and the contents of the report. A Report is encoded as a CBOR array with either 2 or 3 elements. If the array has 2 elements then the optional Timestamp MUST NOT be in the Report encoding. If the array has 3 elements then the optional timestamp MUST be included in the Report encoding. The Report encoding is illustrated in Figure 18.

E(RPT) Format

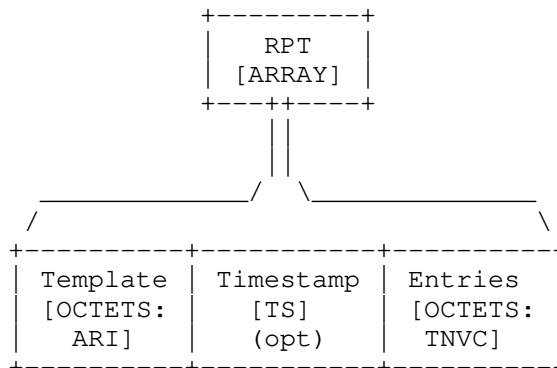


Figure 18

Template

This is the ARI identifying the template used to interpret the data in this report.

This ARI may be parameterized and, if so, the parameters **MUST** include a name field and have been passed-by-name to the template contents when constructing the report.

Timestamp

The timestamp marks the time at which the report was created. This timestamp may be omitted in cases where the time of the report generation can be inferred from other information. For example, if a report is included in a message group such that the timestamp of the message group is equivalent to the timestamp of the report, the report timestamp may be omitted and the timestamp of the included message group used instead.

Entries

This is the collection of data values that comprise the report contents in accordance with the associated Report Template.

8.4.8. State-Based Rules (SBR)

A State-Based Rule (SBR) specifies that a particular action should be taken by an Agent based on some evaluation of the internal state of the Agent. A SBR specifies that starting at a particular START time an ACTION should be run by the Agent if some CONDITION evaluates to true, until the ACTION has been run COUNT times. When the SBR is no longer valid it may be discarded by the agent.

Examples of SBRs include:

Starting 2 hours from receipt, whenever $V1 > 10$, produce a Report for Report Template R1 no more than 20 times.

Starting at some future absolute time, whenever $V2 \neq V4$, run Macro M1 no more than 36 times.

An SBR object is encoded as an OCTETS sequence as illustrated in Figure 19.

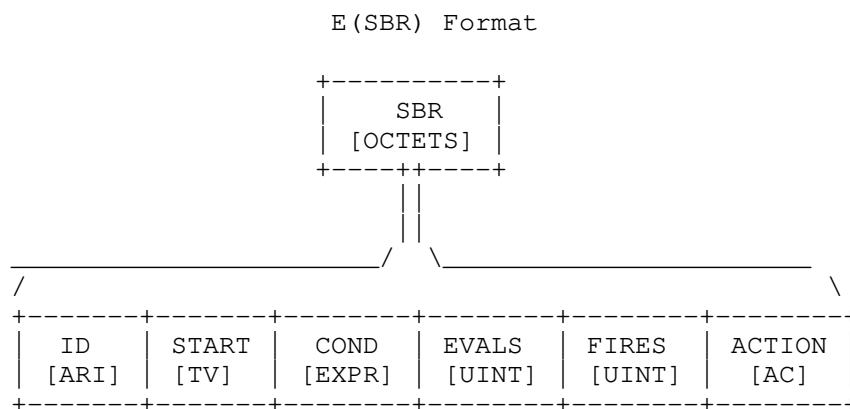


Figure 19

ID

This is the ARI identifying the SBR. If this ARI contains parameters they MUST include a name in support of pass-by-name to each element of the Action AC.

START

The time at which the SBR condition should start to be evaluated. This will mark the first evaluation of the condition associated with the SBR.

CONDITION

The Expression which, if true, results in the SBR running the associated action. An EXPR is considered true if it evaluates to a non-zero value.

EVALS

The number of times the SBR condition can be evaluated. The special value of 0 indicates there is no limit on how many times the condition can be evaluated.

FIRES

The number of times the SBR action can be run. The special value of 0 indicates there is no limit on how many times the action can be run.

ACTION

The collection of Controls and/or Macros to run as part of the action. This is encoded as an AC in accordance with Section 8.2.3.2 with the stipulation that every ARI in this collection MUST be of type CTRL or MAC.

8.4.9. Table Templates (TBLT)

A Table Template (TBLT) describes the types, and optionally names, of the columns that define a Table.

Because TBLTs are only defined in the context of an ADM, their definition cannot change operationally. Therefore, a TBLT is encoded simply as the ARI for the template. The format of the TBLT Object Array is illustrated in Figure 20.

E(TBLT) Format

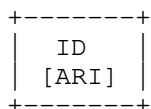


Figure 20

The elements of the TBLT array are defined as follows.

ID

This is the ARI of the table template encoded in accordance with Section 8.3.

8.4.10. Tables (TBL)

A Table object describes the series of values associated with a Table Template.

A Table object is encoded as a CBOR array, with the first element of the array identifying the Table Template and each subsequent element identifying a row in the table. The format of the TBL Object Array is illustrated in Figure 21.

E(TBL) Format

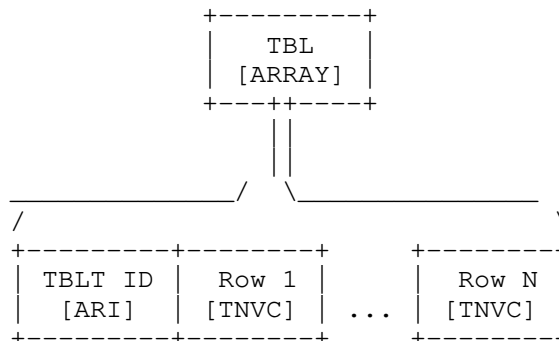


Figure 21

The TBL fields are defined as follows.

Template ID (TBLT ID)

This is the ARI of the table template describing the format of the table and is encoded in accordance with Section 8.3.

Row

Each row of the table is represented as a series of values with optional type information to aid in type checking table contents to column types. Each row is encoded as a TNVC and MAY include type information. AMP implementations should consider the impact of including type information for every row on the overall size of the encoded table.

Each TNVC representing a row MUST contain the same number of elements as there are columns in the referenced Table Template.

8.4.11. Time-Based Rules (TBR)

A Time-Based Rule (TBR) specifies that a particular action should be taken by an Agent based on some time interval. A TBR specifies that starting at a particular START time, and for every PERIOD seconds thereafter, an ACTION should be run by the Agent until the ACTION has been run for COUNT times. When the TBR is no longer valid it MAY BE discarded by the Agent.

Examples of TBRs include:

Starting 2 hours from receipt, produce a Report for Report Template R1 every 10 hours ending after 20 times.

Starting at the given absolute time, run Macro M1 every 24 hours ending after 365 times.

The TBR object is encoded as an OCTETS sequence as illustrated in Figure 22.

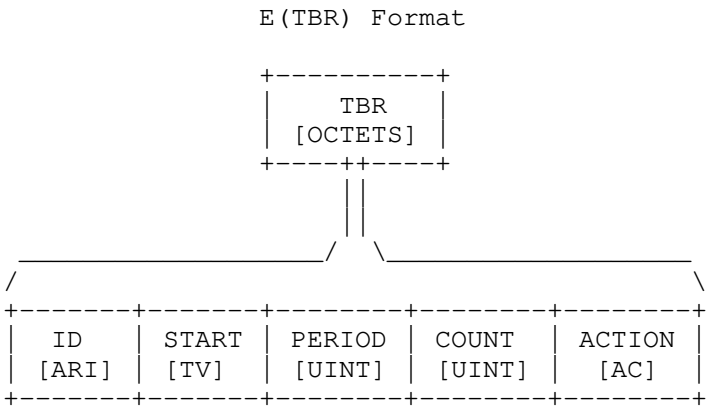


Figure 22

- ID**
- This is the ARI identifying the TBR and is encoded in accordance with Section 8.3. If this ARI contains parameters they MUST include a name in support of pass-by-name to each element of the Action AC.
- START**
- The time at which the TBR condition should start to be evaluated.
- PERIOD**
- The number of seconds to wait between running the action associated with the TBR.
- COUNT**
- The number of times the TBR action can be run. The special value of 0 indicates there is no limit on how many times the action can be run.
- ACTION**
- The collection of Controls and/or Macros to run as part of the action. This is encoded as an ARI Collection in accordance with Section 8.2.3.2 with the stipulation that every ARI in this collection MUST represent either a Control or a Macro.

8.4.12. Variables (VAR)

Variable objects are transmitted in the AMP without the human-readable description.

Variable objects are encoded as an OCTETS sequence whose format is illustrated in Figure 23.

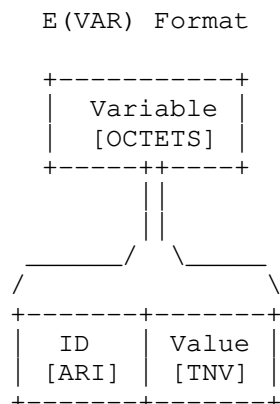


Figure 23

ID

This is the ARI identifying the VAR and is encoded in accordance with Section 8.3. This ARI MUST NOT include parameters.

Value

This field captures the value (and optionally the type and name) of the variable, encoded as a TNV.

9. Functional Specification

This section describes the format of the messages that comprise the AMP protocol.

9.1. AMP Message Summary

The AMP message specification is limited to three basic communications:

Message	Enumeration	Description
Register Agent	0	Add Agents to the list of managed devices known to a Manager.
Report Set	1	Receiving a Report of one or more Report Entries from an Agent.
Perform Control	2	Sending a Macro of one or more Controls to an Agent.
Table Set	3	Receiving one or more tables from an Agent.

Table 4: ADM Message Type Enumerations

The entire management of a network can be performed using these three messages and the configurations from associated ADMs.

9.2. Message Group Format

Individual messages within the AMP are combined into a single group for communication with another AMP Actor. Messages within a group MUST be received and applied as an atomic unit. The format of a message group is illustrated in Figure 24. These message groups are assumed communicated amongst Agents and Managers as the payloads of encapsulating protocols which should provide additional security and data integrity features as needed.

A message group is encoded as a CBOR array with at least 2 elements, the first being the time the group was created followed by 1 or more messages that comprise the group. The format of the message group is illustrated in Figure 24.

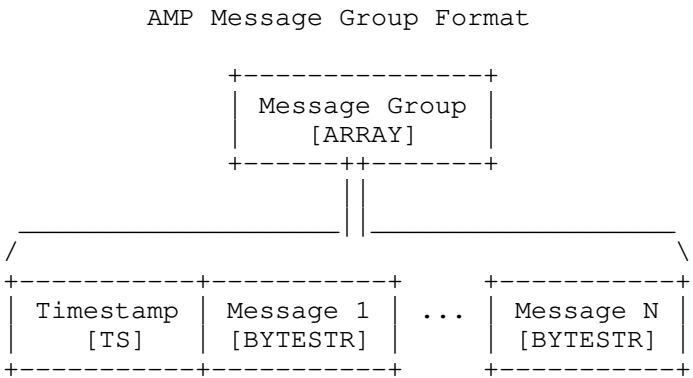


Figure 24

Timestamp
The creation time for this messaging group. Individual messages may have their own creation timestamps based on their type, but the group timestamp also serves as the default creation timestamp for every message in the group. This is encoded in accordance with Table 3.

Message N
The Nth message in the group.

9.3. Message Format

Each message identified in the AMP specification adheres to a common message format, illustrated in Figure 25, consisting of a message header, a message body, and an optional trailer.

Each message in the AMP is encode as an OCTETS sequence formatted in accordance with Figure 25.

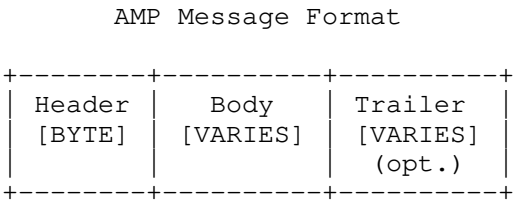


Figure 25

Header
The message header BYTE is shown in Figure 26. The header identifies a message context and opcode as well as flags that

control whether a Report should be generated on message success (Ack) and whether a Report should be generated on message failure (Nack).

AMP Common Message Header

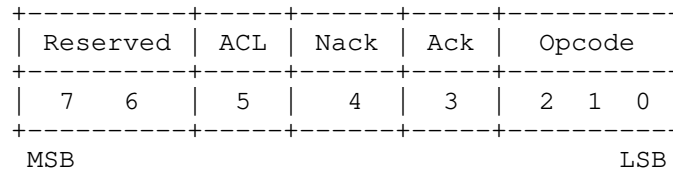


Figure 26

Opcode

The opcode field identifies which AMP message is being represented.

ACK Flag

The ACK flag describes whether successful application of the message must generate an acknowledgment back to the message sender. If this flag is set (1) then the receiving actor **MUST** generate a Report communicating this status. Otherwise, the actor **MAY** generate such a Report based on other criteria.

NACK Flag

The NACK flag describes whether a failure applying the message must generate an error notice back to the message sender. If this flag is set (1) then the receiving Actor **MUST** generate a Report communicating this status. Otherwise, the Actor **MAY** generate such a Report based on other criteria.

ACL Used Flag

The ACL used flag indicates whether the message has a trailer associated with it that specifies the list of AMP actors that may participate in the Actions or definitions associated with the message. This area is still under development.

Body

The message body contains the information associated with the given message.

Trailer

An OPTIONAL access control list (ACL) may be appended as a trailer to a message. When present, the ACL for a message identifies the agents and managers that can be affected by the definitions and actions contained within the message. The explicit impact of an ACL is described in the context of each message below. When an ACL trailer is not present, the message results may be visible to any AMP Actor in the network, pursuant to other security protocol implementations.

9.4. Register Agent

The Register Agent message is used to inform an AMP Manager of the presence of another Agent in the network.

The body of this message is the name of the new agent, encoded as illustrated in Figure 27.

Register Agent Message Body

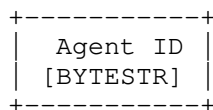


Figure 27

Agent ID

The Agent ID MUST represent the unique address of the Agent in whatever protocol is used to communicate with the Agent.

9.5. Report Set

The Report Set message contains a set of 1 or more Reports produced by an AMP Agent and sent to an AMP Manager.

The body of this message contains information on the recipient of the reports followed by one or more Reports. The body is encoded as illustrated in Figure 28.

Report Set Message Body

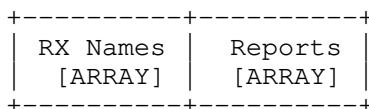


Figure 28

RX Names

This field captures the set of Managers that have been sent this report set. This is encoded as a CBOR array that MUST have at least one entry. Each entry in this array is encoded as a CBOR text string.

Reports

This field captures the set of reports being sent. This is encoded as a CBOR array that MUST have at least one entry. Each entry in this array is encoded as a RPT in accordance with Section 8.4.7.

9.6. Perform Control

The perform control message causes the receiving AMP Actor to run one or more preconfigured Controls provided in the message.

The body of this message is the start time for the controls followed by the controls themselves, as illustrated in Figure 29.

Perform Control Message Body

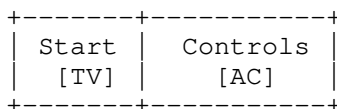


Figure 29

Start

The time at which the Controls/Macros should be run.

Controls

The collection of ARIs that represent the Controls and/or Macros to be run by the AMP Actor. Every ARI in this collection MUST be either a Control or a Macro.

9.7. Table Set

The Table Set message contains a set of 1 or more TBLs produced by an AMP Agent and sent to an AMP Manager.

The body of this message contains information on the recipient of the tables followed by one or more TBLs. The body is encoded as illustrated in Figure 30.

Table Set Message Body

<div> <div> <div></div> <div></div> </div> <div> <div></div> <div></div> </div> </div>	<div> <div> <div></div> <div></div> </div> <div> <div></div> <div></div> </div> </div>
<div> <div> <div></div> <div></div> </div> <div> <div></div> <div></div> </div> </div>	<div> <div> <div></div> <div></div> </div> <div> <div></div> <div></div> </div> </div>

Figure 30

RX Names

This field captures the set of Managers that have been sent this table set. This is encoded as a CBOR array that MUST have at least one entry. Each entry in this array is encoded as a CBOR text string.

Tables

This field captures the set of tables being sent. This is encoded as a CBOR array that MUST have at least one entry. Each entry in this array is encoded as a TBL in accordance with Section 8.4.10.

10. IANA Considerations

A Nickname registry needs to be established.

11. Security Considerations

Security within the AMP exists in two layers: transport layer security and access control.

Transport-layer security addresses the questions of authentication, integrity, and confidentiality associated with the transport of messages between and amongst Managers and Agents. This security is applied before any particular Actor in the system receives data and, therefore, is outside of the scope of this document.

Finer grain application security is done via ACLs provided in the AMP message headers.

12. Implementation Notes

A reference implementation of this version of the AMP specification is available in the 3.6.2 release of the ION open source code base available from sourceforge at <https://sourceforge.net/projects/ion-dtn/>.

13. References

13.1. Informative References

[I-D.birrane-dtn-ama]
Birrane, E., "Asynchronous Management Architecture",
draft-birrane-dtn-ama-07 (work in progress), June 2018.

13.2. Normative References

[I-D.birrane-dtn-adm]
Birrane, E., DiPietro, E., and D. Linko, "AMA Application
Data Model", draft-birrane-dtn-adm-02 (work in progress),
June 2018.

[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>>.

[RFC7049] Bormann, C. and P. Hoffman, "Concise Binary Object
Representation (CBOR)", RFC 7049, DOI 10.17487/RFC7049,
October 2013, <<https://www.rfc-editor.org/info/rfc7049>>.

Appendix A. Acknowledgements

The following participants contributed technical material, use cases, and useful thoughts on the overall approach to this protocol specification: Jeremy Pierce-Mayer of INSYEN AG contributed the concept of the typed data collection and early type checking in the protocol. David Linko and Evana DiPietro of the Johns Hopkins University Applied Physics Laboratory contributed appreciated review and type checking of various elements of this specification.

Author's Address

Edward J. Birrane
Johns Hopkins Applied Physics Laboratory

Email: Edward.Birrane@jhuapl.edu

Delay-Tolerant Networking
Internet-Draft
Intended status: Experimental
Expires: April 18, 2016

E. Birrane
JHU/APL
J. Mayer
INSYEN AG
D. Iannicca
NASA GRC
October 16, 2015

Streamlined Bundle Security Protocol Specification
draft-birrane-dtn-sbsp-01

Abstract

This document defines a streamlined bundle security protocol, which provides data authentication, integrity, and confidentiality services for the Bundle Protocol. Capabilities are provided to protect blocks in a bundle along a single path through a network.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <http://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on April 18, 2016.

Copyright Notice

Copyright (c) 2015 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of

the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction	3
1.1. Related Documents	3
1.2. Terminology	4
2. Key Properties	6
2.1. Block-Level Granularity	6
2.2. Multiple Security Sources	7
2.3. Single Security Destinations	7
2.4. Mixed Security Policy	8
2.5. User-Selected Ciphersuites	8
2.6. Deterministic Processing	8
3. Security Block Definitions	9
3.1. Block Identification	10
3.2. Abstract Security Block	11
3.3. Block Ordering	14
3.4. Bundle Authentication Block	15
3.5. Block Integrity Block	16
3.6. Block Confidentiality Block	17
3.7. Cryptographic Message Syntax Block	19
3.8. Block Interactions	20
3.9. Parameters and Result Fields	22
3.10. BSP Block Example	24
4. Security Processing	27
4.1. Canonical Forms	27
4.1.1. Bundle Canonicalization	27
4.1.2. Block Canonicalization	28
4.1.3. Considerations	31
4.2. Endpoint ID Confidentiality	32
4.3. Bundles Received from Other Nodes	32
4.3.1. Receiving BAB Blocks	32
4.3.2. Receiving BCB Blocks	33
4.3.3. Receiving BIB Blocks	33
4.4. Receiving CMSB Blocks	34
4.5. Bundle Fragmentation and Reassembly	34
4.6. Reactive Fragmentation	35
5. Key Management	35
6. Policy Considerations	35
7. Security Considerations	36
8. Conformance	37
9. IANA Considerations	37
9.1. Bundle Block Types	37
9.2. Cipher Suite Flags	37
9.3. Parameters and Results	38
10. References	39

10.1. Normative References	39
10.2. Informative References	39
Appendix A. Acknowledgements	40
Authors' Addresses	40

1. Introduction

This document defines security features for the Bundle Protocol [RFC5050] intended for use in delay-tolerant networks, in order to provide Delay-Tolerant Networking (DTN) security services.

The Bundle Protocol is used in DTNs that overlay multiple networks, some of which may be challenged by limitations such as intermittent and possibly unpredictable loss of connectivity, long or variable delay, asymmetric data rates, and high error rates. The purpose of the Bundle Protocol is to support interoperability across such stressed networks.

The stressed environment of the underlying networks over which the Bundle Protocol operates makes it important for the DTN to be protected from unauthorized use, and this stressed environment poses unique challenges for the mechanisms needed to secure the Bundle Protocol. Furthermore, DTNs may be deployed in environments where a portion of the network might become compromised, posing the usual security challenges related to confidentiality, integrity, and availability.

This document describes the Streamlined Bundle Security Protocol (SBSP), which provides security services for blocks within a bundle from the bundle source to the bundle destination. Specifically, the SBSP provides authentication, integrity, and confidentiality for bundles along a path through a DTN.

SBSP applies, by definition, only to those nodes that implement it, known as "security-aware" nodes. There MAY be other nodes in the DTN that do not implement SBSP. All nodes can interoperate with the exception that SBSP security operations can only happen at SBSP security-aware nodes.

1.1. Related Documents

This document is best read and understood within the context of the following other DTN documents:

"Delay-Tolerant Networking Architecture" [RFC4838] defines the architecture for delay-tolerant networks, but does not discuss security at any length.

The DTN Bundle Protocol [RFC5050] defines the format and processing of the blocks used to implement the Bundle Protocol, excluding the security-specific blocks defined here.

The Bundle Security Protocol [RFC6257] introduces the concepts of security blocks for authentication, confidentiality, and integrity. The SBSP is based off of this document.

1.2. Terminology

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 [RFC2119].

We introduce the following terminology for purposes of clarity.

- o Source - the bundle node from which a bundle originates.
- o Destination - the bundle node to which a bundle is ultimately destined.
- o Forwarder - the bundle node that forwarded the bundle on its most recent hop.
- o Intermediate Receiver, Waypoint, or "Next Hop" - the neighboring bundle node to which a forwarder forwards a bundle.
- o Path - the ordered sequence of nodes through which a bundle passes on its way from source to destination. The path is not necessarily known by the bundle, or any bundle-aware nodes.

Figure 1 below is adapted from [RFC5050] and shows four bundle nodes (denoted BN1, BN2, BN3, and BN4) that reside above some transport layer(s). Three distinct transport and network protocols (denoted T1/N1, T2/N2, and T3/N3) are also shown.

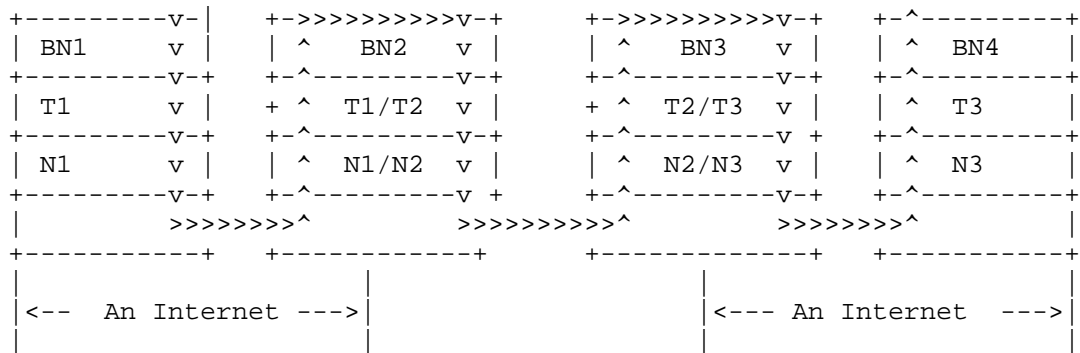


Figure 1: Bundle Nodes Sit at the Application Layer of the Internet Model

BN1 originates a bundle that it forwards to BN2. BN2 forwards the bundle to BN3, and BN3 forwards the bundle to BN4. BN1 is the source of the bundle and BN4 is the destination of the bundle. BN1 is the first forwarder, and BN2 is the first intermediate receiver; BN2 then becomes the forwarder, and BN3 the intermediate receiver; BN3 then becomes the last forwarder, and BN4 the last intermediate receiver, as well as the destination.

If node BN2 originates a bundle (for example, a bundle status report or a custodial signal), which is then forwarded on to BN3, and then to BN4, then BN2 is the source of the bundle (as well as being the first forwarder of the bundle) and BN4 is the destination of the bundle (as well as being the final intermediate receiver).

We introduce the following security-specific DTN terminology.

- o Security-Service - the security features supported by this specification: authentication, integrity, and confidentiality.
- o Security-Source - a bundle node that adds a security block to a bundle.
- o Security-Destination - a bundle node that evaluates a security block from a bundle. When a security-service is applied hop-by-hop, the security-destination is the next intermediate receiver. Otherwise, the security-destination is the same as the bundle destination.
- o Security-Target - the portion of a bundle (e.g., the primary block, payload block, extension block, or entire bundle) that receives a security-service as part of a security-operation.

- o Security Block - a single instance of a SBSP extension block in a bundle.
- o Security-Operation - the application of a security-service to a specific security-target, notated as OP(security-service, security-target). For example, OP(authentication, bundle) or OP(confidentiality, payload). Every security-operation in a bundle MUST be unique, meaning that a security-service can only be applied to a security-target once in a bundle. A security-operation MAY be implemented by one or more security blocks.

2. Key Properties

The application of security services in a DTN is a complex endeavor that must consider physical properties of the network, policies at each node, and various application security requirements. Rather than enumerate all potential security implementations in all potential DTN topologies, this specification defines a set of key properties of a security system. The security primitives outlined in this document MUST enable the realization of these properties in a DTN deploying the Bundle Protocol.

2.1. Block-Level Granularity

Blocks within a bundle represent different types of information. The primary block contains identification and routing information. The payload block carries application data. Extension blocks carry a variety of data that may augment or annotate the payload, or otherwise provide information necessary for the proper processing of a bundle along a path. Therefore, applying a single level and type of security across an entire bundle fails to recognize that blocks in a bundle may represent different types of information with different security needs.

Security services within this specification MUST provide block level granularity where applicable such that different blocks within a bundle may have different security services applied to them.

For example, within a bundle, a payload might be encrypted to protect its contents, whereas an extension block containing summary information related to the payload might be integrity signed but otherwise unencrypted to provide certain nodes access to payload-related data without providing access to the payload.

2.2. Multiple Security Sources

The Bundle Protocol allows extension blocks to be added to a bundle at any time during its existence in the DTN. When a waypoint node adds a new extension block to a bundle, that extension block may have security services applied to it by that waypoint. Similarly, a waypoint node may add a security service to an existing extension block, consistent with its security policy. For example, a node representing a boundary between a trusted part of the network and an untrusted part of the network may wish to apply payload encryption for bundles leaving the trusted portion of the network.

In each case, a node other than the bundle originator may be adding a security service to the bundle and, as such, the source for the security service will be different than the source of the bundle itself. Security services **MUST** track their originating node so as to properly apply policy and key selection associated with processing the security service at the bundle destination.

Referring to Figure 1, if the bundle that originates at BN1 is given security blocks by BN1, then BN1 is the security-source for those blocks as well as being the source of the bundle. If the bundle that originates at BN1 is then given a security block by BN2, then BN2 is the security-source for that block even though BN1 remains the bundle source.

A bundle **MAY** have multiple security blocks and these blocks **MAY** have different security-sources. Each security block in a bundle will be associated with a specific security-operation. All security blocks comprising a security-operation **MUST** have the same security-source and security-destination.

As required in [RFC5050], forwarding nodes **MUST** transmit blocks in a bundle in the same order in which they were received. This requirement applies to all DTN nodes, not just ones that implement security processing. Blocks in a bundle **MAY** be added or deleted according to the applicable specification, but those blocks that are both received and transmitted **MUST** be transmitted in the same order that they were received.

2.3. Single Security Destinations

The destination of all security blocks in a bundle **MUST** be the bundle destination, with the exception of authentication security blocks, whose destination is the next hop along the bundle path. In a DTN, there is typically no guarantee that a bundle will visit a particular intermediate receiver during its journey, or that a particular series of intermediate receivers will be visited in a particular order.

Security-destinations different from bundle destinations would place a tight (and possibly intractable) coupling between security and routing services in an overlay network.

2.4. Mixed Security Policy

Different nodes in a DTN may have different security-related capabilities. Some nodes may not be security-aware and will not understand any security-related extension blocks. Other nodes may have security policies that require evaluation of security services at places other than the bundle destination (such as verifying integrity signatures at certain waypoint nodes). Other nodes may ignore any security processing if they are not the destination of the bundle. The security services described in this specification must allow each of these scenarios.

Extension blocks representing security services MUST have their block processing flags set such that the block (and bundle, where applicable) will be treated appropriately by non-security-aware nodes.

Extension blocks providing integrity and authentication services within a bundle MUST support options to allow waypoint nodes to evaluate these signatures if such nodes have the proper configuration to do so.

2.5. User-Selected Ciphersuites

The security services defined in this specification rely on a variety of ciphersuites providing integrity signatures, ciphertext, and other information necessary to populate security blocks. Users may wish to select differing ciphersuites to implement different security services. For example, some users may wish to use a SHA-1 based hash for integrity whereas other users may require a SHA-2 hash instead. The security services defined in this specification MUST provide a mechanism for identifying what ciphersuite has been used to populate a security block.

2.6. Deterministic Processing

In all cases, the processing order of security services within a bundle must avoid ambiguity when evaluating security at the bundle destination. This specification MUST provide determinism in the application and evaluation of security services, even when doing so results in a loss of flexibility.

3. Security Block Definitions

There are four types of security blocks that MAY be included in a bundle. These are the Bundle Authentication Block (BAB), the Block Integrity Block (BIB), the Block Confidentiality Block (BCB), and the Cryptographic Messaging Syntax Block (CMSB).

The BAB is used to ensure the authenticity and integrity of the bundle along a single hop from forwarder to intermediate receiver. As such, BABs operate between topologically adjacent nodes. Security-aware nodes MAY choose to require BABs from a given neighbor in the network in order to receive and process a received bundle.

The BIB is used to ensure the authenticity and integrity of its security-target from the BIB security-source, which creates the BIB, to the bundle destination, which verifies the BIB authenticator. The authentication information in the BIB MAY (when possible) be verified by any node in between the BIB security-source and the bundle destination.

The BCB indicates that the security-target has been encrypted, in whole or in part, at the BCB security-source in order to protect its content while in transit to the bundle destination.

The CMSB contains a Cryptographic Message Syntax (CMS) payload used to describe a security service applied to another extension block. NOTE: Applications may choose to simply place CMS text as the payload to the bundle. In such cases, security is considered to be implemented at the application layer and CMSBs are not required in that case.

Certain cipher suites may allow or require multiple instances of a block to appear in the bundle. For example, an authentication cipher suite may require two security blocks, one before the payload block and one after. Despite the presence of two security blocks, they both comprise the same security-operation - `OP(authentication,bundle)` in this example.

A security-operation MUST NOT be applied more than once in a bundle. For example, the two security-operations: `OP(integrity, payload)` and `OP(integrity, payload)` are considered redundant and MUST NOT appear together in a bundle. However, the two security operations `OP(integrity, payload)` and `OP(integrity, extension_block_1)` MAY both be present in the bundle. Also, the two security operations `OP(integrity, extension_block_1)` and `OP(integrity, extension_block_2)` are unique and may both appear in the same bundle.

Many of the fields in these block definitions use the Self-Delimiting Numeric Value (SDNV) type whose format and encoding is as defined in [RFC5050].

3.1. Block Identification

This specification requires that every target block of a security operation be uniquely identifiable. In cases where there can only be a single instance of a block in the bundle (as is the case with the primary block and the payload block) then the unique identifier is simply the block type. These blocks are described as "singleton blocks". It is possible that a bundle may contain multiple instances of a block type. In such a case, each instance of the block type must be uniquely identifiable and the block type itself is not sufficient for this identification. These blocks are described as "non-singleton blocks".

The definition of the extension block header from [RFC5050] does not provide additional identifying information for a block beyond the block type. The addition of an occurrence number to the block is necessary to identify the block instance in the bundle. This section describes the use of an Artificial EID (AEID) reference in a block header to add unique identification for non-singleton blocks.

Figure 7 of [RFC5050] illustrates that an EID reference in a block header is the 2-tuple of the reference scheme and the reference scheme specific part (SSP), each of which are encoded as SDNVs. The AEID MUST encode the occurrence number in the reference scheme SDNV and MUST set the reference SSP to 0. A reference SSP value of 0 is an invalid offset for an SSP in the bundle dictionary and, therefore, the use of 0 in this field identifies the reference as an AEID.

The occurrence number MAY be any positive value that is not already present as an occurrence number for the same block type in the bundle. These numbers are independent of relative block position within the bundle, and whether blocks of the same type have been added or removed from the bundle. Once an AEID has been added to a block instance, it MUST NOT be changed until all security operations that target the block instance have been removed from the bundle.

If a node wishes to apply a security operation to a target block it MUST determine whether the target block is a singleton block or a non-singleton block. If the target block is non-singleton, then the node MUST find the AEID for the target. If an AEID is not present in the target block header then the node MAY choose to either cancel the security operation or add an AEID to the block, in accordance with security policy.

If a node chooses to add an AEID to a target block header it MUST perform the following activities.

- o The "Block contains an EID reference field" flag MUST be set for the target block, if it is not already set.
- o The EID reference count for the block MUST be updated to reflect the addition of the AEID.
- o The scheme offset of the AEID MUST be a value greater than 0. The scheme offset MUST NOT be the same as any other AEID of any other block in the bundle sharing the same block type.
- o The SSP offset of the AEID MUST be the value 0. There MUST NOT be any other EID in the block header that has a value of 0 for the SSP offset.

If there is no AEID present in a block, and if a node is unable to add an AEID by following the above process, then the block MUST NOT have an SBSP security operation applied to it.

It is RECOMMENDED that every block in a bundle other than the primary and payload blocks be treated as a non-singleton block. However, the identification of singleton blocks SHOULD be in accordance with the security policy of a node.

3.2. Abstract Security Block

Each security block uses the Canonical Bundle Block Format as defined in [RFC5050]. That is, each security block is comprised of the following elements:

- o Block Type Code
- o Block Processing Control Flags
- o Block EID Reference List (OPTIONAL)
- o Block Data Length
- o Block Type Specific Data Fields

Since the four security block types have most fields in common, we can shorten the description of the block type specific data fields if we first define an abstract security block (ASB) and then specify each of the real blocks in terms of the fields that are present/absent in an ASB. Note that no bundle ever contains an actual ASB, which is simply a specification artifact.

The structure of an Abstract Security Block is given in Figure 2. Although the diagram hints at a fixed-format layout, this is purely for the purpose of exposition. Except for the "type" field, all fields are variable in length.

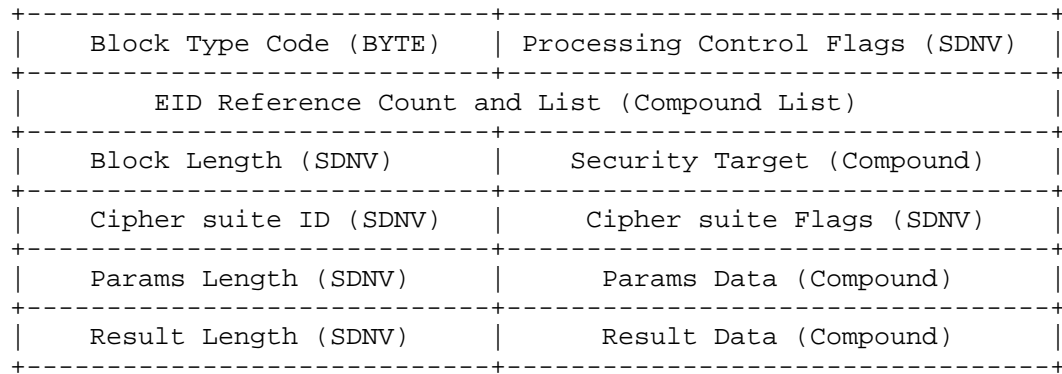


Figure 2: Abstract Security Block Structure

An ASB consists of the following fields, some of which are optional.

- o Block-Type Code (Byte) - as described in [RFC5050]. The block-type codes for security blocks are:
 - * BundleAuthenticationBlock - BAB: 0x02
 - * BlockIntegrityBlock - BIB: 0x03
 - * BlockConfidentialityBlock - BCB: 0x04
- o Block Processing Control Flags (SDNV) - as described in [RFC5050]. There are no general constraints on the use of the block processing control flags, and some specific requirements are discussed later.
- o (OPTIONAL) EID Reference Count and List - as described in [RFC5050]. Presence of the EID reference field is indicated by the setting of the "Block contains an EID reference field" (EID_REF) bit of the block processing control flags. If no EID fields are present, then the composite field itself MUST be omitted entirely and the EID_REF bit MUST be unset. A count field of zero is not permitted. The possible EIDs are:
 - (OPTIONAL) Security-source - specifies the security-source for the block. If this is omitted, then the source of the bundle is assumed to be the security-source unless otherwise indicated

by policy or associated cipher suite definition. When present, the security-source MUST be the first EID in the list.

(OPTIONAL) AEID - specifies an identifier that can be used to uniquely identify an instance of a non-singleton block. This field MUST be present for non-singleton blocks. This field MUST NOT be present for singleton blocks, such as the primary block and the payload block. The construction of the AEID is discussed in Section 3.1.

- o Block Length (SDNV) - as described in [RFC5050].
- o Block type specific data fields as follows:
 - * Security-Target (Compound) - Uniquely identifies the target of the associated security-operation.

As discussed in Section 3.1 a singleton block is identified by its block type and a non-singleton block is identified by the combination of its block type and an occurrence number. The security-target is a compound field that contains the block type (as a byte) and occurrence number (as an SDNV).

The occurrence number of a singleton block MUST be set to 0. The occurrence number of a non-singleton block MUST be set to the scheme offset of the AEID associated with the block being targeted by the security operation.

- * (OPTIONAL) Cipher suite ID (SDNV)
- * (OPTIONAL) Cipher suite flags (SDNV)
- * (OPTIONAL) Cipher Suite Parameters - compound field of the next two items.
 - + Cipher suite parameters length (SDNV) - specifies the length of the next field, which is the cipher suite-parameters data field.
 - + Cipher suite parameters data - parameters to be used with the cipher suite in use, e.g., a key identifier or initialization vector (IV). See Section 3.9 for a list of potential parameters and their encoding rules. The particular set of parameters that is included in this field is defined as part of a cipher suite specification.
- * (OPTIONAL) Security Result - compound field of the next two items.

- + Security result length (SDNV) - contains the length of the next field, which is the security-result data field.
- + Security result data - contains the results of the appropriate cipher suite specific calculation (e.g., a signature, Message Authentication Code (MAC), or cipher-text block key).

The structure of the cipher suite flags field is shown in Figure 3. In each case, the presence of an optional field is indicated by setting the value of the corresponding flag to one. A value of zero indicates the corresponding optional field is missing. Presently, there are three flags defined for the field; for convenience, these are shown as they would be extracted from a single-byte SDNV. Future additions may cause the field to grow to the left so, as with the flags fields defined in [RFC5050], the description below numbers the bit positions from the right rather than the standard RFC definition, which numbers bits from the left.

bits 6-3 are reserved for future use.

src - bit 2 indicates whether the EID-reference field of the ASB contains the optional reference to the security-source.

parm - bit 1 indicates whether or not the cipher suite parameters length and cipher suite parameters data fields are present.

res - bit 0 indicates whether or not the ASB contains the security-result length and security-result data fields.

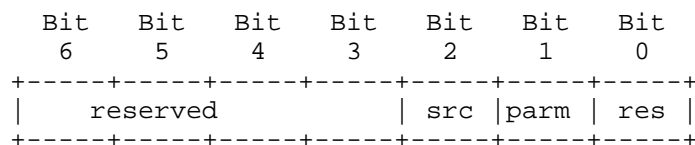


Figure 3: Cipher Suite Flags

3.3. Block Ordering

A security-operation may be implemented in a bundle using either one or two security blocks. For example, the operation `OP(authentication, bundle)` MAY be accomplished by a single BAB block in the bundle, or it MAY be accomplished by two BAB blocks in the bundle. To avoid confusion, we use the following terminology to identify the block or blocks comprising a security-operation.

The terms "First" and "Last" are used ONLY when describing multiple security blocks comprising a single security-operation. A "First" block refers to the security block that is closest to the primary block in the canonical form of the bundle. A "Last" block refers to the security block that is furthest from the primary block in the canonical form of the bundle.

If a single security block implements the security-operation, then it is referred to as a "Lone" block. For example, when a bundle authentication cipher suite requires a single BAB block we refer to it as a Lone BAB. When a bundle authentication cipher suite requires two BAB blocks we refer to them as the First BAB and the Last BAB.

This specification and individual cipher suites impose restrictions on what optional fields must and must not appear in First blocks, Last blocks, and Lone blocks.

3.4. Bundle Authentication Block

This section describes typical field values for the BAB, which is solely used to implement OP(authentication, bundle).

The block-type code field value MUST be 0x02.

The block processing control flags value can be set to whatever values are required by local policy. Cipher suite designers should carefully consider the effect of setting flags that either discard the block or delete the bundle in the event that this block cannot be processed.

The security-target MUST be the entire bundle, which MUST be represented by a <block type><occurrence number> of <0x00><0x00>.

The cipher suite ID MUST be documented as a hop-by-hop authentication cipher suite. When a Lone BAB is used, the cipher suite MUST be documented as requiring one instance of the BAB. When a First BAB and Last BAB are used, the cipher suite MUST be documented as requiring two instances of the BAB.

The cipher suite parameters field MAY be present, if so specified in the cipher suite specification.

An EID-reference to the security-source MAY be present in either a First BAB or a Lone BAB. An EID-reference to the security-source MUST NOT be present in a Last BAB.

The security-result captures the result of applying the cipher suite calculation (e.g., the MAC or signature) to the relevant

parts of the bundle, as specified in the cipher suite definition.
This field **MUST** be present in either a Lone BAB or a Last BAB.
This field **MUST NOT** be present in a First BAB.

Notes:

- o When multiple BAB blocks are used, the mandatory fields of the Last BAB must match those of the First BAB.
- o The First BAB or Lone BAB, when present, **SHOULD** immediately follow the primary block.
- o A Last BAB, when present, **SHOULD** be the last block in the bundle.
- o Since OP(authentication, bundle) is allowed only once in a bundle, it is **RECOMMENDED** that users wishing to support multiple authentication signatures define a multi-target cipher suite, capturing multiple security results in cipher suite parameters.

3.5. Block Integrity Block

A BIB is an ASB with the following additional restrictions:

The block-type code value **MUST** be 0x03.

The block processing control flags value can be set to whatever values are required by local policy. Cipher suite designers should carefully consider the effect of setting flags that either discard the block or delete the bundle in the event that this block cannot be processed.

The security-target **MUST** uniquely identify a block within the bundle. The reserved block type 0x01 specifies the singleton payload block. The reserved type 0x00 specifies the singleton primary block. The security-target for a BIB **MUST NOT** reference a security block defined in this specification (BAB, BIB, or BCB).

The cipher suite ID **MUST** be documented as an end-to-end authentication-cipher suite or as an end-to-end error-detection-cipher suite.

The cipher suite parameters field **MAY** be present in either a Lone BIB or a First BIB. This field **MUST NOT** be present in a Last BIB.

An EID-reference to the security-source **MAY** be present in either a Lone BIB or a First BIB. This field **MUST NOT** be present in a Last BIB.

The security-result captures the result of applying the cipher suite calculation (e.g., the MAC or signature) to the relevant parts of the security-target, as specified in the cipher suite definition. This field **MUST** be present in either a Lone BIB or a Last BIB. This field **MUST NOT** be present in a First BIB.

The cipher suite **MAY** process less than the entire security-target. If the cipher suite processes less than the complete, original security-target, the cipher suite parameters **MUST** specify which bytes of the security-target are protected.

Notes:

- o Since OP(integrity, target) is allowed only once in a bundle per target, it is **RECOMMENDED** that users wishing to support multiple integrity signatures for the same target define a multi-signature cipher suite, capturing multiple security results in cipher suite parameters.
- o For some cipher suites, (e.g., those using asymmetric keying to produce signatures or those using symmetric keying with a group key), the security information **MAY** be checked at any hop on the way to the destination that has access to the required keying information, in accordance with Section 3.8.
- o The use of a generally available key is **RECOMMENDED** if custodial transfer is employed and all nodes **SHOULD** verify the bundle before accepting custody.

3.6. Block Confidentiality Block

A BCB is an ASB with the following additional restrictions:

The block-type code value **MUST** be 0x04.

The block processing control flags value can be set to whatever values are required by local policy, except that a Lone BCB or First BCB **MUST** have the "replicate in every fragment" flag set. This indicates to a receiving node that the payload portion in each fragment represents cipher-text.

t. This flag **SHOULD NOT** be set otherwise. Cipher suite designers should carefully consider the effect of setting flags that either discard the block or delete the bundle in the event that this block cannot be processed.

The security-target **MUST** uniquely identify a block within the bundle. The security-target for a BCB **MAY** reference the payload

block, a non-security extension block, or a BIB block. The reserved type 0x01 specifies the singleton payload block.

The cipher suite ID MUST be documented as a confidentiality cipher suite.

Key-information, if available, MUST appear only in a Lone BCB or a First BCB.

Any additional bytes generated as a result of encryption and/or authentication processing of the security-target SHOULD be placed in an "integrity check value" field (see Section 3.9) in the security-result of the Lone BCB or Last BCB.

The cipher suite parameters field MAY be present in either a Lone BCB or a First BCB. This field MUST NOT be present in a Last BCB.

An EID-reference to the security-source MAY be present in either a Lone BCB or a First BCB. This field MUST NOT be present in a Last BCB. The security-source can also be specified as part of key-information described in Section 3.9.

The security-result MAY be present in either a Lone BCB or a Last BCB. This field MUST NOT be present in a First BCB. This compound field normally contains fields such as an encrypted bundle encryption key and/or authentication tag.

The BCB is the only security block that modifies the contents of its security-target. When a BCB is applied, the security-target body data are encrypted "in-place". Following encryption, the security-target body data contains cipher-text, not plain-text. Other security-target block fields (such as type, processing control flags, and length) remain unmodified.

Fragmentation, reassembly, and custody transfer are adversely affected by a change in size of the payload due to ambiguity about what byte range of the block is actually in any particular fragment. Therefore, when the security-target of a BCB is the bundle payload, the BCB MUST NOT alter the size of the payload block body data. Cipher suites SHOULD place any block expansion, such as authentication tags (integrity check values) and any padding generated by a block-mode cipher, into an integrity check value item in the security-result field (see Section 3.9) of the BCB. This "in-place" encryption allows fragmentation, reassembly, and custody transfer to operate without knowledge of whether or not encryption has occurred.

Notes:

- o The cipher suite MAY process less than the entire original security-target body data. If the cipher suite processes less than the complete, original security-target body data, the BCB for that security-target MUST specify, as part of the cipher suite parameters, which bytes of the body data are protected.
- o The BCB's "discard" flag may be set independently from its security-target's "discard" flag. Whether or not the BCB's "discard" flag is set is an implementation/policy decision for the encrypting node. (The "discard" flag is more properly called the "Discard if block cannot be processed" flag.)
- o A BCB MAY include information as part of additional authenticated data to address parts of the target block, such as EID references, that are not converted to cipher-text.

3.7. Cryptographic Message Syntax Block

A CMSB is an ASB with the following additional restrictions:

The block-type code value MUST be 0x05.

The content of the block must contain valid CMS data, as defined in RFC 5652, and encoded in X.690 BER or DER encoding.

The block processing control flags value can be set to whatever values are required by local policy. This flag SHOULD NOT be set otherwise. Cipher suite designers should carefully consider the effect of setting flags that either discard the block or delete the bundle in the event that this block cannot be processed.

The security-target MUST uniquely identify a block within the bundle. The reserved block type 0x01 specifies the singleton payload block.

The security operation(s) will be performed on the security-target block's data and the resulting CMS content will be stored within the CMSB block's security-result field. The security-target block's data will then be removed.

A CMSB block MAY include multiple CMS security operations within a single block to allow for multiple nested operations to be performed on a bundle block. Multiple CMSB blocks MAY be included in a bundle as long as the security-target for each is unique.

Key-information, if available, MUST appear within the CMS content contained in the security-result field.

A CMSB block is created with its corresponding security-target field pointing to a unique bundle block. The CMS security operations are performed upon the security-target's data field and the resulting encoded CMS content is stored within the CMS security-result field of the CMSB's payload. The security-target block's data MAY be left intact, replaced with alternate data, or completely erased based on the specification of the utilized CMS ciphersuite definition and applicable policy.

Multiple CMS operations may be nested within a single CMSB block to allow more than one security operation to be performed upon a security-target.

CMS Operations can be considered to have SBSP parallels: CMSB Enveloped-Data content type SHALL be considered as equivalent to a SBSP BCB block, and a CMSB Signed-Data type SHALL be considered as equivalent to a SBSP BIB block.

3.8. Block Interactions

The four security-block types defined in this specification are designed to be as independent as possible. However, there are some cases where security blocks may share a security-target creating processing dependencies.

If confidentiality is being applied to a target that already has integrity applied to it, then an undesirable condition occurs where a security-aware intermediate node would be unable to check the integrity result of a block because the block contents have been encrypted after the integrity signature was generated. To address this concern, the following processing rules MUST be followed.

- o If confidentiality is to be applied to a target, it MUST also be applied to every integrity operation already defined for that target. This means that if a BCB is added to encrypt a block, another BCB MUST also be added to encrypt a BIB also targeting that block.
- o An integrity operation MUST NOT be applied to a security-target if a BCB in the bundle shares the same security-target. This prevents ambiguity in the order of evaluation when receiving a BIB and a BCB for a given security-target.
- o An integrity value MUST NOT be evaluated if the BIB providing the integrity value is the security target of an existing BCB block in the bundle. In such a case, the BIB data contains cipher-text as it has been encrypted.

- o An integrity value MUST NOT be evaluated if the security-target of the BIB is also the security-target of a BCB in the bundle. In such a case, the security-target data contains cipher-text as it has been encrypted.
- o As mentioned in Section 3.6, a BIB MUST NOT have a BCB as its security target. BCBs may embed integrity results as part of cipher suite parameters.
- o As mentioned in Section 4.4, CMS operations are considered to have operational parallels. When a CMSB is used, these parallels MUST be considered for block interactions (e.g., a Signed-Data structure MUST NOT be evaluated if the security-target of the operation is also the security-target of a BCB)
- o If a single bundle is going to contain a CMSB as well as other security blocks, the CMS operations MUST be performed and the CMSB MUST be created before any other security operation is applied.
- o On reception of a bundle containing a CMSB and other security blocks, the CMSB must be decoded last.

Additionally, since the CMSB block may contain either integrity or confidentiality information in its encapsulated CMS, there is no way to evaluate conflicts when a BIB/BCB and a CMSB have the same security target. To address this concern, the following processing rules MUST be followed.

- o If an extension block is the target of a BIB or a BCB, then the extension block MUST NOT also be the target of a CMSB, and vice-versa.
- o If a bundle is the target of a BAB block, then the bundle MUST NOT also be the target of a CMSB, and vice-versa.
- o Generally, a CMSB MUST be processed before any BIB or BCB blocks are processed.

These restrictions on block interactions impose a necessary ordering when applying security operations within a bundle. Specifically, for a given security-target, BIBs MUST be added before BCBs, and BABs MUST be added after all other security blocks. This ordering MUST be preserved in cases where the current BPA is adding all of the security blocks for the bundle or whether the BPA is a waypoint adding new security blocks to a bundle that already contains security blocks.

3.9. Parameters and Result Fields

Various cipher suites include several items in the cipher suite parameters and/or security-result fields. Which items MAY appear is defined by the particular cipher suite description. A cipher suite MAY support several instances of the same type within a single block.

Each item is represented as a type-length-value. Type is a single byte indicating the item. Length is the count of data bytes to follow, and is an SDNV-encoded integer. Value is the data content of the item.

Item types, name, and descriptions are defined as follows.

Cipher suite parameters and result fields.

Type	Name	Description
0	Reserved	
1	Initialization Vector (IV)	A random value, typically eight to sixteen bytes.
2	Reserved	
3	Key Information	Material encoded or protected by the key management system and used to transport an ephemeral key protected by a long-term key.
4	Content Range	Pair of SDNV values (offset,length) specifying the range of payload bytes to which an operation applies. The offset MUST be the offset within the original bundle, even if the current bundle is a fragment.
5	Integrity Signatures	Result of BAB or BIB digest or other signing operation.
6	Unassigned	
7	Salt	An IV-like value used by certain confidentiality suites.
8	BCB Integrity Check Value (ICV) / Authentication Tag	Output from certain confidentiality cipher suite operations to be used at the destination to verify that the protected data has not been modified. This value MAY contain padding if required by the cipher suite.
9-255	Reserved	

Table 1

3.10. BSP Block Example

An example of SBSP blocks applied to a bundle is illustrated in Figure 4. In this figure the first column represents blocks within a bundle and the second column represents a unique identifier for each block, suitable for use as the security-target of a SBSP security-block. Since the mechanism and format of a security-target is not specified in this document, the terminology B1...Bn is used to identify blocks in the bundle for the purposes of illustration.

Block in Bundle	ID
Primary Block	B1
First BAB OP(authentication, Bundle)	B2
Lone BIB OP(integrity, target=B1)	B3
Lone BCB OP(confidentiality, target=B5)	B4
Extension Block	B5
Lone BIB OP(integrity, target=B7)	B6
Extension Block	B7
Lone BCB OP(confidentiality, target=B9)	B8
Lone BIB (encrypted by B8) OP(integrity, target=B11)	B9
Lone BCB OP(confidentiality, target=B11)	B10
Payload Block	B11
Last BAB OP(authentication, Bundle)	B12

Figure 4: Sample Use of BSP Blocks

In this example a bundle has four non-security-related blocks: the primary block (B1), two extension blocks (B5,B7), and a payload block (B11). The following security applications are applied to this bundle.

- o Authentication over the bundle. This is accomplished by two BAB blocks: B2 and B12.
- o An integrity signature applied to the canonicalized primary block. This is accomplished by a single BIB, B3.
- o Confidentiality for the first extension block. This is accomplished by a single BCB block, B4.
- o Integrity for the second extension block. This is accomplished by a single BIB block, B6.
- o An integrity signature on the payload. This is accomplished by a single BIB block, B9.
- o Confidentiality for the payload block and its integrity signature. This is accomplished by two Lone BCB blocks: B8 encrypting B9, and B10 encrypting B11.

Block in Bundle	ID
Primary Block	B1
First BAB OP(authentication, Bundle)	B2
Lone CMSB security-target=0x01 security-result= Signed-Data { Digest Algorithm(s), Enveloped-Data { Encrypted Data, Encrypted Encryption Key(s) }, Signature(s) and Certificate Chain(s) }	B3
Payload Block (Empty Data Field)	B4
Last BAB OP(authentication, Bundle)	B5

Figure 5: Sample Bundle With CMS Block

In this example a bundle has two non-security-related blocks: the primary block (B1) and a payload block (B4). This method would allow for the bundle to carry multiple CMS payloads by utilizing a multiple CMSB ASBs. The following security applications are applied to this bundle.

- o Authentication over the bundle. This is accomplished by two BAB blocks: B2 and B5.
- o Encrypted and signed CMS content contained within the CMSB block. The first CMS operation, encryption, is performed on the data contained within the block the security-target points to, in this case, the payload block. The resulting encrypted data is then signed and the final CMS content is stored within the CMSB block's security-result field. The payload block's data is subsequently removed now that the original data has been encoded within the CMSB block.

4. Security Processing

This section describes the security aspects of bundle processing.

4.1. Canonical Forms

In order to verify a signature of a bundle, the exact same bits, in the exact same order, MUST be input to the calculation upon verification as were input upon initial computation of the original signature value. Consequently, a node MUST NOT change the encoding of any URI [RFC3986] in the dictionary field, e.g., changing the DNS part of some HTTP URL from lower case to upper case. Because bundles MAY be modified while in transit (either correctly or due to implementation errors), canonical forms of security-targets MUST be defined.

Many fields in various blocks are stored as variable-length SDNVs. These are canonicalized into an "unpacked form" as eight-byte fixed-width fields in network byte order. The size of eight bytes is chosen because implementations MAY handle larger SDNV values as invalid, as noted in [RFC5050].

4.1.1. Bundle Canonicalization

Bundle canonicalization permits no changes at all to the bundle between the security-source and the destination, with the exception of one of the Block Processing Control Flags, as described below. It is intended for use in BAB cipher suites. This algorithm conceptually catenates all blocks in the order presented, but omits all security-result data fields in security blocks having the bundle as their security-target. For example, when a BAB cipher suite specifies this algorithm, we omit the BAB security-result from the catenation. The inclusion of security-result length fields is as determined by the specified cipher suite. A security-result length field MAY be present even when the corresponding security-result data fields are omitted.

Notes:

- o In the Block Processing Control Flags field the unpacked SDNV is ANDed with mask 0xFFFF FFFF FFFF FFDF to zero the flag at bit 5 ("Block was forwarded without being processed"). If this flag is not zeroed out, then a bundle passing through a non-security aware node will set this flag which will change the message digest and the BAB block will fail to verify.
- o In the above, we specify that security-result data is omitted. This means that no bytes of the security-result data are input.

If the security-result length is included in the catenation, we assume that the security-result length will be known to the module that implements the cipher suite before the security-result is calculated, and require that this value be in the security-result length field even though the security-result data itself will be omitted.

- o The 'res' bit of the cipher suite ID, which indicates whether or not the security-result length and security-result data field are present, is part of the canonical form.
- o The value of the block data length field, which indicates the length of the block, is also part of the canonical form. Its value indicates the length of the entire block when the block includes the security-result data field.

4.1.2. Block Canonicalization

This algorithm protects those parts of a block that SHOULD NOT be changed in transit.

There are three types of blocks that may undergo block canonicalization: the primary block, the payload block, or an extension block.

4.1.2.1. Primary Block Canonicalization

The canonical form of the primary block is shown in Figure 6. Essentially, it de-references the dictionary block, adjusts lengths where necessary, and ignores flags that may change in transit.

Version	Processing flags (incl. COS and SRR)
Canonical primary block length	
Destination endpoint ID length	
Destination endpoint ID	
Source endpoint ID length	
Source endpoint ID	
Report-to endpoint ID length	
Report-to endpoint ID	
Creation Timestamp (2 x SDNV)	
Lifetime	

Figure 6: The Canonical Form of the Primary Bundle Block

The fields shown in Figure 6 are as follows:

- o The version value is the single-byte value in the primary block.
- o The processing flags value in the primary block is an SDNV, and includes the class-of-service (COS) and status report request (SRR) fields. For purposes of canonicalization, the unpacked SDNV is ANDed with mask 0x0000 0000 0007 C1BE to set to zero all reserved bits and the "bundle is a fragment" bit.
- o The canonical primary block length value is a four-byte value containing the length (in bytes) of this structure, in network byte order.
- o The destination endpoint ID length and value are the length (as a four-byte value in network byte order) and value of the destination endpoint ID from the primary bundle block. The URI is simply copied from the relevant part(s) of the dictionary block and is not itself canonicalized. Although the dictionary entries contain "null-terminators", the null-terminators are not included in the length or the canonicalization.
- o The source endpoint ID length and value are handled similarly to the destination.

- o The report-to endpoint ID length and value are handled similarly to the destination.
- o The unpacked SDNVs for the creation timestamp and lifetime are copied from the primary block.
- o Fragment offset and total application data unit length are ignored, as is the case for the "bundle is a fragment" bit mentioned above. If the payload data to be canonicalized is less than the complete, original bundle payload, the offset and length are specified in the cipher suite parameters.

4.1.2.2. Payload Block Canonicalization

When canonicalizing the payload block, the block processing control flags value used for canonicalization is the unpacked SDNV value with reserved and mutable bits masked to zero. The unpacked value is ANDed with mask 0x0000 0000 0000 0077 to zero reserved bits and the "last block" bit. The "last block" bit is ignored because BABs and other security blocks MAY be added for some parts of the journey but not others, so the setting of this bit might change from hop to hop.

Payload blocks are canonicalized as-is, with the exception that, in some instances, only a portion of the payload data is to be protected. In such a case, only those bytes are included in the canonical form, and additional cipher suite parameters are required to specify which part of the payload is protected, as discussed further below.

4.1.2.3. Extension Block Canonicalization

When canonicalizing an extension block, the block processing control flags value used for canonicalization is the unpacked SDNV value with reserved and mutable bits masked to zero. The unpacked value is ANDed with mask 0x0000 0000 0000 0057 to zero reserved bits, the "last block" flag and the "Block was forwarded without being processed" bit. The "last block" flag is ignored because BABs and other security blocks MAY be added for some parts of the journey but not others, so the setting of this bit might change from hop to hop.

The "Block was forwarded without being processed" flag is ignored because the bundle may pass through nodes that do not understand that extension block and this flag would be set.

Endpoint ID references in blocks are canonicalized using the de-referenced text form in place of the reference pair. The reference count is not included, nor is the length of the endpoint ID text.

The EID reference is, therefore, canonicalized as <scheme>:<SSP>, which includes the ":" character.

Since neither the length of the canonicalized EID text nor a null-terminator is used in EID canonicalization, a separator token MUST be used to determine when one EID ends and another begins. When multiple EIDs are canonicalized together, the character "," SHALL be placed between adjacent instances of EID text.

The block-length is canonicalized as its unpacked SDNV value. If the data to be canonicalized is less than the complete, original block data, this field contains the size of the data being canonicalized (the "effective block") rather than the actual size of the block.

4.1.3. Considerations

- o The canonical forms for the bundle and various extension blocks is not transmitted. It is simply an artifact used as input to digesting.
- o We omit the reserved flags because we cannot determine if they will change in transit. The masks specified above will have to be revised if additional flags are defined and they need to be protected.
- o Our URI encoding does not preserve the null-termination convention from the dictionary field, nor do we canonicalize the scheme and scheme-specific part (SSP) separately. Instead, the byte array < scheme name > : < scheme-specific part (SSP) > is used in the canonicalization.
- o The URI encoding will cause errors if any node rewrites the dictionary content (e.g., changing the DNS part of an HTTP URL from lower case to upper case). This could happen transparently when a bundle is synched to disk using one set of software and then read from disk and forwarded by a second set of software. Because there are no general rules for canonicalizing URIs (or IRIs), this problem may be an unavoidable source of integrity failures.
- o All SDNV fields here are canonicalized as eight-byte unpacked values in network byte order. Length fields are canonicalized as four-byte values in network byte order. Encoding does not need optimization since the values are never sent over the network.
- o These canonicalization algorithms assume that endpoint IDs themselves are immutable and they are unsuitable for use in environments where that assumption might be violated.

- o Cipher suites MAY define their own canonicalization algorithms and require the use of those algorithms over the ones provided in this specification.

4.2. Endpoint ID Confidentiality

Every bundle has a primary block that contains the source and destination endpoint IDs, and possibly other EIDs (in the dictionary field) that cannot be encrypted. If endpoint ID confidentiality is required, then bundle-in-bundle encapsulation can solve this problem in some instances.

Similarly, confidentiality requirements MAY also apply to other parts of the primary block (e.g., the current-custodian), and that is supported in the same manner.

4.3. Bundles Received from Other Nodes

Security blocks MUST be processed in a specific order when received by a security-aware node. The processing order is as follows.

- o All BAB blocks in the bundle MUST be evaluated prior to evaluating any other block in the bundle.
- o All BCB blocks in the bundle MUST be evaluated prior to evaluating any BIBs in the bundle. When BIBs and BCBs share a security-target, BCBs MUST be evaluated first and BIBs second.

4.3.1. Receiving BAB Blocks

Nodes implementing this specification SHALL consult their security policy to determine whether or not a received bundle is required by policy to include a BAB.

If the bundle is not required to have a BAB then BAB processing on the received bundle is complete, and the bundle is ready to be further processed for BIB/BCB handling or delivery or forwarding. Security policy may provide a means to override this default behavior and require processing of a BAB if it exists.

If the bundle is required to have a BAB but does not, then the bundle MUST be discarded and processed no further. If the bundle is required to have a BAB but the key information for the security-source cannot be determined or the security-result value check fails, then the bundle has failed to authenticate, and the bundle MUST be discarded and processed no further.

If the bundle is required to have a BAB, and a BAB exists, and the BAB information is verified, then the BAB processing on the received bundle is complete, and the bundle is ready to be further processed for BIB/BCB handling or delivery or forwarding.

A BAB received in a bundle MUST be stripped before the bundle is forwarded. A new BAB MAY be added as required by policy. This MAY require correcting the "last block" field of the to-be-forwarded bundle.

4.3.2. Receiving BCB Blocks

If the bundle has a BCB and the receiving node is the destination for the bundle, the node MUST decrypt the relevant parts of the security-target in accordance with the cipher suite specification.

If the relevant parts of an encrypted payload cannot be decrypted (i.e., the decryption key cannot be deduced or decryption fails), then the bundle MUST be discarded and processed no further; in this case, a bundle deletion status report (see [RFC5050]) indicating the decryption failure MAY be generated. If any other encrypted security-target cannot be decrypted then the associated security-target and all security blocks associated with that target MUST be discarded and processed no further.

When a BCB is decrypted, the recovered plain-text MUST replace the cipher-text in the security-target body data

4.3.3. Receiving BIB Blocks

A BIB MUST NOT be processed if the security-target of the BIB is also the security-target of a BCB in the bundle. Given the order of operations mandated by this specification, when both a BIB and a BCB share a security-target, it means that the security-target MUST have been encrypted after it was integrity signed and, therefore, the BIB cannot be verified until the security-target has been decrypted by processing the BCB.

If the security policy of a security-aware node specifies that a bundle SHOULD apply integrity to a specific security-target and no such BIB is present in the bundle, then the node MUST process this security-target in accordance with the security policy. This MAY involve removing the security-target from the bundle. If the removed security-target is the payload or primary block, the bundle MAY be discarded. This action may occur at any node that has the ability to verify an integrity signature, not just the bundle destination.

If the bundle has a BIB and the receiving node is the destination for the bundle, the node MUST verify the security-target in accordance with the cipher suite specification. If a BIB check fails, the security-target has failed to authenticate and the security-target SHALL be processed according to the security policy. A bundle status report indicating the failure MAY be generated. Otherwise, if the BIB verifies, the security-target is ready to be processed for delivery.

If the bundle has a BIB and the receiving node is not the bundle destination, the receiving node MAY attempt to verify the value in the security-result field. If the check fails, the node SHALL process the security-target in accordance to local security policy. It is RECOMMENDED that if a payload integrity check fails at a waypoint that it is processed in the same way as if the check fails at the destination.

4.4. Receiving CMSB Blocks

A CMSB MUST NOT be processed if its security target is also the security target of any BAB, BIB, or BCB in the bundle.

The security services provided by a CMSB will be considered successful if all services in the CMSB are validated. If any one service encapsulated in the CMSB fails to validate, then the CMSB MUST be considered as having failed to validate and MUST be dispositioned in accordance with security policy.

4.5. Bundle Fragmentation and Reassembly

If it is necessary for a node to fragment a bundle and security services have been applied to that bundle, the fragmentation rules described in [RFC5050] MUST be followed. As defined there and repeated here for completeness, only the payload may be fragmented; security blocks, like all extension blocks, can never be fragmented. In addition, the following security-specific processing is REQUIRED:

- o Due to the complexity of bundle fragmentation, including the possibility of fragmenting bundle fragments, integrity and confidentiality operations are not to be applied to a bundle fragment. Specifically, a BCB or BIB MUST NOT be added to a bundle fragment, even if the security-target of the security block is not the payload. When integrity and confidentiality must be applied to a fragment, we RECOMMEND that encapsulation be used instead.

- o The authentication security policy requirements for a bundle MUST be applied individually to all the bundles resulting from a fragmentation event.
- o A BAB cipher suite MAY specify that it only applies to non-fragmented bundles and not to bundle fragments.
- o The decision to fragment a bundle MUST be made prior to adding authentication to the bundle. The bundle MUST first be fragmented and authentication applied to each individual fragment.
- o If a bundle with a BAB is fragmented by a non-security-aware node, then the entire bundle must be re-assembled before being processed to allow for the proper verification of the BAB.

4.6. Reactive Fragmentation

When a partial bundle has been received, the receiving node SHALL consult its security policy to determine if it MAY fragment the bundle, converting the received portion into a bundle fragment for further forwarding. Whether or not reactive fragmentation is permitted SHALL depend on the security policy and the cipher suite used to calculate the BAB authentication information, if required.

Specifically, if the security policy does not require authentication, then reactive fragmentation MAY be permitted. If the security policy does require authentication, then reactive fragmentation MUST NOT be permitted if the partial bundle is not sufficient to allow authentication.

If reactive fragmentation is allowed, then all BAB blocks must be removed from created fragments.

5. Key Management

Key management in delay-tolerant networks is recognized as a difficult topic and is one that this specification does not attempt to solve.

6. Policy Considerations

When implementing the SBSP, several policy decisions must be considered. This section describes key policies that affect the generation, forwarding, and receipt of bundles that are secured using this specification.

- o If a bundle is received that contains more than one security-operation, in violation of the SBSP, then the BPA must determine

how to handle this bundle. The bundle may be discarded, the block affected by the security-operation may be discarded, or one security-operation may be favored over another.

- o BPAs in the network MUST understand what security-operations they should apply to bundles. This decision may be based on the source of the bundle, the destination of the bundle, or some other information related to the bundle.
- o If an intermediate receiver has been configured to add a security-operation to a bundle, and the received bundle already has the security-operation applied, then the receiver MUST understand what to do. The receiver may discard the bundle, discard the security-target and associated SBSP blocks, replace the security-operation, or some other action.
- o It is recommended that security operations only be applied to the payload block, the primary block, and any block-types specifically identified in the security policy. If a BPA were to apply security operations such as integrity or confidentiality to every block in the bundle, regardless of the block type, there could be downstream errors processing blocks whose contents must be inspected at every hop in the network path.

7. Security Considerations

Certain applications of DTN need to both sign and encrypt a message, and there are security issues to consider with this.

- o To provide an assurance that a security-target came from a specific source and has not been changed, then it should be signed with a BIB.
- o To ensure that a security-target cannot be inspected during transit, it should be encrypted with a BCB.
- o Adding a BIB to a security-target that has already been encrypted by a BCB is not allowed. Therefore, we recommend three methods to add an integrity signature to an encrypted security-target. First, at the time of encryption, an integrity signature may be generated and added to the BCB for the security-target as additional information in the security-result field. Second, the encrypted block may be replicated as a new block and integrity signed. Third, an encapsulation scheme may be applied to encapsulate the security-target (or the entire bundle) such that the encapsulating structure is, itself, no longer the security-target of a BCB and may therefore be the security-target of a BIB.

8. Conformance

All implementations are strongly RECOMMENDED to provide at least a BAB cipher suite. A relay node, for example, might not deal with end-to-end confidentiality and data integrity, but it SHOULD exclude unauthorized traffic and perform hop-by-hop bundle verification.

9. IANA Considerations

This protocol has fields that have been registered by IANA.

9.1. Bundle Block Types

This specification allocates three block types from the existing "Bundle Block Types" registry defined in [RFC6255].

Additional Entries for the Bundle Block-Type Codes Registry:

Value	Description	Reference
2	Bundle Authentication Block	This document
3	Block Integrity Block	This document
4	Block Confidentiality Block	This document

Table 2

9.2. Cipher Suite Flags

This protocol has a cipher suite flags field and certain flags are defined. An IANA registry has been set up as follows.

The registration policy for this registry is: Specification Required

The Value range is: Variable Length

Cipher Suite Flag Registry:

Bit Position (right to left)	Description	Reference
0	Block contains result	This document
1	Block Contains parameters	This document
2	Source EID ref present	This document
>3	Reserved	This document

Table 3

9.3. Parameters and Results

This protocol has fields for cipher suite parameters and results. The field is a type-length-value triple and a registry is required for the "type" sub-field. The values for "type" apply to both the cipher suite parameters and the cipher suite results fields. Certain values are defined. An IANA registry has been set up as follows.

The registration policy for this registry is: Specification Required

The Value range is: 8-bit unsigned integer.

Cipher Suite Parameters and Results Type Registry:

Value	Description	Reference
0	reserved	This document
1	initialization vector (IV)	This document
2	reserved	This document
3	key-information	This document
4	content-range (pair of SDNVs)	This document
5	integrity signature	This document
6	unassigned	This document
7	salt	This document
8	BCB integrity check value (ICV)	This document
9-191	reserved	This document
192-250	private use	This document
251-255	reserved	This document

Table 4

10. References

10.1. Normative References

- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.
- [RFC5050] Scott, K. and S. Burleigh, "Bundle Protocol Specification", RFC 5050, November 2007.
- [RFC5652] Housley, R., "Cryptographic Message Syntax (CMS)", STD 70, RFC 5652, DOI 10.17487/RFC5652, September 2009, <<http://www.rfc-editor.org/info/rfc5652>>.
- [RFC6255] Blanchet, M., "Delay-Tolerant Networking Bundle Protocol IANA Registries", RFC 6255, May 2011.

10.2. Informative References

- [RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, January 2005.
- [RFC4838] Cerf, V., Burleigh, S., Hooke, A., Torgerson, L., Durst, R., Scott, K., Fall, K., and H. Weiss, "Delay-Tolerant Networking Architecture", RFC 4838, April 2007.

[RFC5751] Ramsdell, B. and S. Turner, "Secure/Multipurpose Internet Mail Extensions (S/MIME) Version 3.2 Message Specification", RFC 5751, January 2010.

[RFC6257] Symington, S., Farrell, S., Weiss, H., and P. Lovell, "Bundle Security Protocol Specification", RFC 6257, May 2011.

Appendix A. Acknowledgements

The following participants contributed technical material, use cases, and useful thoughts on the overall approach to this security specification: Scott Burleigh of the Jet Propulsion Laboratory, Amy Alford and Angela Hennessy of the Laboratory for Telecommunications Sciences, and Angela Dalton and Cherita Corbett of the Johns Hopkins University Applied Physics Laboratory.

Authors' Addresses

Edward J. Birrane, III
The Johns Hopkins University Applied Physics Laboratory
11100 Johns Hopkins Rd.
Laurel, MD 20723
US

Phone: +1 443 778 7423
Email: Edward.Birrane@jhuapl.edu

Jeremy Pierce-Mayer
INSYEN AG
Muenchner Str. 20
Oberpfaffenhofen, Bavaria DE
Germany

Phone: +49 08153 28 2774
Email: jeremy.mayer@insyen.com

Dennis C. Iannicca
NASA Glenn Research Center
21000 Brookpark Rd.
Brook Park, OH 44135
US

Phone: +1-216-433-6493
Email: dennis.c.iannicca@nasa.gov

Delay-Tolerant Networking Working Group
Internet Draft
Intended status: Standards Track
Expires: December 2015

S. Burleigh
JPL, Calif. Inst. Of Technology
K. Fall
Carnegie Mellon University / SEI
E. Birrane
APL, Johns Hopkins University
June 21, 2015

Bundle Protocol
draft-dtnwg-bp-00.txt

Status of this Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

This document may contain material from IETF Documents or IETF Contributions published or made publicly available before November 10, 2008. The person(s) controlling the copyright in some of this material may not have granted the IETF Trust the right to allow modifications of such material outside the IETF Standards Process. Without obtaining an adequate license from the person(s) controlling the copyright in such materials, this document may not be modified outside the IETF Standards Process, and derivative works of it may not be created outside the IETF Standards Process, except to format it for publication as an RFC or to translate it into languages other than English.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF), its areas, and its working groups. Note that other groups may also distribute working documents as Internet-Drafts.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

The list of current Internet-Drafts can be accessed at
<http://www.ietf.org/ietf/lid-abstracts.txt>

The list of Internet-Draft Shadow Directories can be accessed at
<http://www.ietf.org/shadow.html>

This Internet-Draft will expire on July 21, 2015.

Copyright Notice

Copyright (c) 2015 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document.

Abstract

This Internet Draft presents a specification for 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.

Table of Contents

1. Introduction.....	4
2. Conventions used in this document.....	6
3. Service Description.....	6
3.1. Definitions.....	6
3.2. Implementation Architectures.....	12
3.2.1. Bundle protocol application server.....	12
3.2.2. Peer application nodes.....	13
3.2.3. Sensor network nodes.....	13
3.2.4. Dedicated bundle router.....	13
3.3. Services Offered by Bundle Protocol Agents.....	13
4. Bundle Format.....	14
4.1. Self-Delimiting Numeric Values (SDNVs).....	14
4.2. Bundle Processing Control Flags.....	16
4.3. Block Processing Control Flags.....	18
4.4. Identifiers.....	19
4.4.1. Endpoint ID.....	19
4.4.2. Node ID.....	20
4.5. Formats of Bundle Blocks.....	21
4.5.1. Primary Bundle Block.....	23
4.5.2. Canonical Bundle Block Format.....	25
4.5.3. Bundle Payload Block.....	26
4.6. Extension Blocks.....	27
4.6.1. Current Custodian.....	27
4.6.2. Flow Label.....	28
4.6.3. Previous Node ID.....	28

4.6.4. Bundle Age.....	28
4.6.5. Hop Count.....	28
5. Bundle Processing.....	29
5.1. Generation of Administrative Records.....	29
5.2. Bundle Transmission.....	30
5.3. Bundle Dispatching.....	30
5.4. Bundle Forwarding.....	31
5.4.1. Forwarding Contraindicated.....	33
5.4.2. Forwarding Failed.....	33
5.5. Bundle Expiration.....	34
5.6. Bundle Reception.....	34
5.7. Local Bundle Delivery.....	35
5.8. Bundle Fragmentation.....	36
5.9. Application Data Unit Reassembly.....	37
5.10. Custody Transfer.....	38
5.10.1. Custody Acceptance.....	38
5.10.2. Custody Release.....	39
5.11. Custody Transfer Success.....	39
5.12. Custody Transfer Failure.....	39
5.13. Bundle Deletion.....	39
5.14. Discarding a Bundle.....	40
5.15. Canceling a Transmission.....	40
6. Administrative Record Processing.....	40
6.1. Administrative Records.....	40
6.1.1. Bundle Status Reports.....	41
6.1.2. Custody Signals.....	45
6.2. Generation of Administrative Records.....	47
6.3. Reception of Custody Signals.....	48
7. Services Required of the Convergence Layer.....	48
7.1. The Convergence Layer.....	48
7.2. Summary of Convergence Layer Services.....	48
8. Security Considerations.....	49
9. IANA Considerations.....	50
10. References.....	50
10.1. Normative References.....	50
10.2. Informative References.....	51
11. Acknowledgments.....	51
12. Significant Changes From RFC 5050.....	52
13. Open Issues.....	52
13.1. Definitions section structure.....	52
13.2. Payload nomenclature.....	53
13.3. Application Agent.....	53
13.4. Bundle Endpoint definition.....	53
13.5. Alignment with ICN.....	53
13.6. Implementation Architectures.....	53
13.7. Security protocol name.....	54
13.8. Bundle format.....	54

13.9. SDNVs.....	54
13.10. Bundle Processing Control Flags.....	54
13.11. Extended class of service features.....	54
13.12. Primary block CRC type.....	54
13.13. Inventory.....	54
13.14. Block numbers.....	55
13.15. Clearing flag.....	55
13.16. Overriding BP spec.....	55
13.17. Time of forwarding.....	55
13.18. Block multiplicity.....	55
Appendix A. For More Information.....	56

1. Introduction

Since the publication of the Bundle Protocol Specification (Experimental RFC 5050[RFC5050]) in 2007, the Delay-Tolerant Networking Bundle Protocol has been implemented in multiple programming languages and deployed to a wide variety of computing platforms for a wide range of successful exercises. This implementation and deployment experience has demonstrated the general utility of the protocol for challenged network operations.

It has also, as expected, 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.

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 sits at the application layer of some number of constituent networks, forming a store-carry-forward overlay network. Key capabilities of BP include:

- . Custodial forwarding
- . Ability to cope with intermittent connectivity
- . Ability to take advantage of scheduled, predicted, and opportunistic connectivity (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]. [TUT] contains a tutorial-level overview of DTN concepts.

BP's location within the standard protocol stack is as shown in Figure 1. BP uses underlying "native" network protocols for communications within a given constituent network.

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).

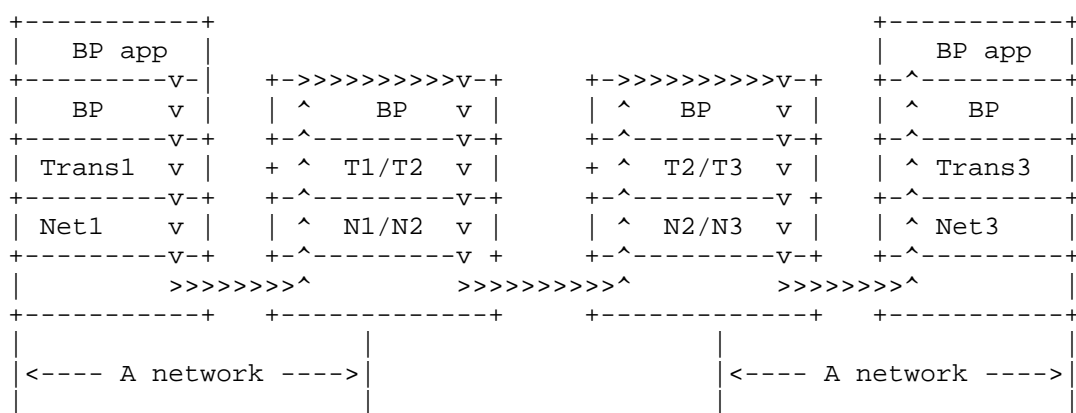


Figure 1: The Bundle Protocol Sits at the Application Layer of the Protocol Stack Model

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.

2. Conventions used in this document

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC-2119 [RFC2119].

In this document, these words will appear with that interpretation only when in ALL CAPS. Lower case uses of these words are not to be interpreted as carrying RFC-2119 significance.

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 data all metadata that might be needed in order to make the data immediately usable when delivered to applications. Each bundle comprises a sequence of two or more "blocks" of protocol data, which serve various purposes. Multiple instances of the same bundle (the same unit of DTN protocol data) might exist concurrently in different parts of a network -- possibly in different representations and/or 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 of some bundle in the network that is in that node's local memory.

Bundle payload - A bundle payload (or simply "payload") is the application data whose conveyance to the bundle's destination is the purpose for the transmission of a given bundle. The terms "bundle content", "bundle payload", and "payload" are used interchangeably in this document. The "nominal" 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 nominal payload for a bundle forwarded in response to reception of that bundle is the payload of the received bundle.

Fragment - A fragment is a bundle whose payload block contains a fragmentary payload. A fragmentary payload is either the first N bytes or the last N bytes of some other payload -- either a nominal payload or a fragmentary payload -- of length M, such that $0 < N < M$.

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. 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. Each bundle node has three conceptual components, defined below: a "bundle protocol agent", a set of zero or more "convergence layer adapters", and an "application agent".

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. The manner in which it does so 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.

Convergence layer adapters - A convergence layer adapter (CLA) 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. As such, 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 is wholly an implementation matter, exactly as described for the BPA. The definitions of CLAs and CL protocols are beyond the scope of this specification.

Application agent - The application agent (AA) of a node is the node component that utilizes the BP services to effect communication for some purpose. The application agent in turn has two elements, an administrative element and an application-specific element. The application-specific element of an AA constructs, requests transmission of, accepts delivery of, and processes application-specific application data units; the only interface between the BPA and the application-specific element of the AA is the BP service interface. The administrative element of an AA constructs and requests transmission of administrative records (including status reports and custody signals), and it accepts delivery of and

processes any custody signals that the node receives. In addition to the BP service interface, there is a (conceptual) private control interface between the BPA and the administrative element of the AA that enables each to direct the 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.

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 formats of some fundamental administrative records (and of no other application data units) are defined in this specification.

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 4.4.4 below). The special case of an endpoint that contains exactly one node is termed a "singleton" endpoint. 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 identify themselves by a single common endpoint ID and thus form a single bundle endpoint. For a bundle to be considered "delivered" to an endpoint, a minimum number of receiving nodes may be required to receive it successfully. This lower limit is called the minimum reception group, and is defined in the Transmission discussion below. *Note* too that a given bundle node might identify itself by multiple endpoint IDs and thus be a member of multiple bundle endpoints. The destination of every bundle is an endpoint, which may or may not be singleton. The source of every bundle is a singleton endpoint.

Transmission - A transmission is an attempt by a node's BPA to cause copies of a bundle to be delivered at all nodes in the minimum reception group of some endpoint (the bundle's destination) in response to a transmission request issued by the node's application agent. The minimum reception group of an endpoint may be any one of the following: (a) ALL of the nodes registered (see definition below) in an endpoint that is permitted to contain multiple nodes (in which case forwarding to the endpoint is functionally similar to "multicast" operations in the Internet, though possibly very

different in implementation); (b) ANY N of the nodes registered in an endpoint that is permitted to contain multiple nodes, where N is in the range from zero to the cardinality of the endpoint; or (c) THE SOLE NODE registered in a singleton endpoint (in which case forwarding to the endpoint is functionally similar to "unicast" operations in the Internet). The nature of the minimum reception group for a given endpoint can be determined from the endpoint's ID (again, see Section 4.4 below): for some endpoint ID "schemes", the nature of the minimum reception group is fixed - in a manner that is defined by the scheme - for all endpoints identified under the scheme; for other schemes, the nature of the minimum reception group is indicated by some lexical feature of the "scheme-specific part" of the endpoint ID, in a manner that is defined by the scheme. Any number of transmissions may be concurrently undertaken by the bundle protocol agent of a given node.

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

Registration - A registration is the state machine characterizing a given node's membership in a given endpoint. Any number of registrations may be concurrently associated with a given endpoint, and any number of registrations may be concurrently associated with a given node. Any single registration must at any time be in one of two states: Active or Passive. A registration always has an associated "delivery failure action", the action that is to be taken when a bundle that is "deliverable" (see below) subject to that registration is received at a time when the registration is in the Passive state. Delivery failure action must be one of the following:

- . defer "delivery" (see below) 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; or
- . "abandon" (see below) delivery of the bundle subject to this registration.

An additional implementation-specific delivery deferral procedure may optionally be associated with the registration. While the state of a registration is Active, reception of a bundle that is deliverable subject to this registration must cause the bundle to 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. While the state of a registration is Passive, reception of a bundle that is deliverable subject to this registration must cause delivery of the bundle to be abandoned or deferred as mandated by the registration's current delivery failure action; in the latter case, any additional delivery deferral procedure associated with the registration must also be performed.

Delivery - 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 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. 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 the value of the bundle's "Acknowledgement by application is requested" flag and any other relevant metadata (an implementation matter), has been presented to the node's application agent in a manner consistent with the state of that registration and, as applicable, the registration's delivery failure action.

Deliverability, Abandonment - 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) delivery of the bundle subject to this registration has not been abandoned. To "abandon" delivery of a bundle subject to a registration is simply to declare it no longer deliverable subject to that registration; normally only registrations' registered delivery failure actions cause deliveries to be abandoned.

Deletion, Discarding - A bundle protocol agent "discards" a bundle by simply ceasing all operations on the bundle and functionally erasing all references to it; the specific procedures by which this is accomplished are an implementation matter. Bundles are discarded silently; i.e., the discarding of a bundle does not result in generation of an administrative record. "Retention constraints" are elements of the bundle state that prevent a bundle from being discarded; a bundle cannot be discarded while it has any retention constraints. A bundle protocol agent "deletes" a bundle in response to some anomalous condition by notifying the bundle's report-to node

of the deletion (provided such notification is warranted; see Section 5.13 for details) and then arbitrarily removing all of the bundle's retention constraints, enabling the bundle to be discarded.

Custody - A node "takes custody" of a bundle when it determines that it will retain a copy of the bundle for some period, forwarding and possibly re-forwarding the bundle as appropriate and destroying that retained copy only when custody of that bundle is formally "released". Custody of a bundle may only be taken if the destination of the bundle is a singleton endpoint. A "custodial node" (or "custodian") of a bundle is a node that has taken custody of the bundle and has not yet released that custody. To "accept custody" upon receiving a bundle is to take custody of the bundle, mark the bundle in such a way as to indicate to nodes that subsequently receive the bundle that it has taken custody, and notify all current custodians of the bundle that it has taken custody. Custody may only be released when either (a) notification is received that some other node has accepted custody of the same bundle; (b) notification is received that the bundle has been delivered at the (sole) node registered in the bundle's destination endpoint; (c) the current custodian chooses to fragment the bundle, releasing custody of the original bundle and taking custody of the fragments instead, or (d) the bundle is explicitly deleted for some reason, such as lifetime expiration. To "refuse custody" of a bundle is to notify all current custodians of that bundle that an opportunity to take custody of the bundle has been declined.

The custody transfer mechanism in BP is primarily intended as a means of recovering from forwarding failures. When a bundle arrives at a node from which it cannot be forwarded, BP must recover from this error. BP can "return" the bundle back toward some node for forwarding along some different path in the network, or else it can instead send a small "signal" bundle back to such a node, in the event that this node has retained a copy of the bundle ("taken custody") and is therefore able to re-forward the bundle without receiving a copy. Custody transfer sharply reduces the network traffic required for recovery from forwarding failures, at the cost of increased buffer occupancy and state management at the custodial nodes.

Note that custodial re-forwarding can also be initiated by expiration of a timer prior to reception of a custody acceptance signal. Since the absence of a custody acceptance signal might be caused by failure to receive the bundle, rather than only a disinclination to take custody, custody transfer can additionally serve as an automated retransmission mechanism. Because custody transfer's only remedy for loss of any part of a bundle is

retransmission of the entire bundle (not just the lost portion), custody transfer is a less efficient automated retransmission mechanism than the reliable transport protocols that are typically available at the convergence layer; configuring BPAs to use reliable convergence-layer protocols between nodes is generally the best means of ensuring bundle delivery at the destination node(s). But there are some use cases (typically involving unidirectional links) in which custody transfer in BP may be a more cost-effective solution for reliable transmission between two BP agents than operating retransmission protocols at the convergence layer.

Embargo - Forwarding failures are not just operational anomalies; they may also convey information about the network, i.e., a forwarding failure may indicate a sustained lapse in forwarding capability. Since forwarding a bundle to a dead end wastes time and bandwidth, the bundle protocol agent may choose to manage such a lapse by imposing a temporary "embargo" on subsequent forwarding activity that is similar to the forwarding attempt that has been seen to fail. Mechanisms for motivating, imposing, enforcing, and lifting embargoes are beyond the scope of this document.

3.2. Implementation Architectures

The above definitions are intended to enable the bundle protocol's operations to be specified in a manner that minimizes bias toward any particular implementation architecture. To illustrate the range of interoperable implementation models that might conform to this specification, four example architectures are briefly described below.

3.2.1. Bundle protocol application server

A single bundle protocol application server, constituting a single bundle node, runs as a daemon process on each computer. The daemon's functionality includes all functions of the bundle protocol agent, all convergence layer adapters, and both the administrative and application-specific elements of the application agent. The application-specific element of the application agent functions as a server, offering bundle protocol service over a local area network: it responds to remote procedure calls from application processes (on the same computer and/or remote computers) that need to communicate via the bundle protocol. The server supports its clients by creating a new (conceptual) node for each one and registering each such node in a client-specified endpoint. The conceptual nodes managed by the server function as clients' bundle protocol service access points.

3.2.2. Peer application nodes

Any number of bundle protocol application processes, each one constituting a single bundle node, run on each computer. The functionality of the bundle protocol agent, all convergence layer adapters, and the administrative element of the application agent is provided by a library to which each node process is dynamically linked at run time. The application-specific element of each node's application agent is node-specific application code.

3.2.3. Sensor network nodes

Each node of the sensor network is the self-contained implementation of a single bundle node. All functions of the bundle protocol agent, all convergence layer adapters, and the administrative element of the application agent are implemented in simplified form in hardware, while the application-specific element of each node's application agent is implemented in a programmable microcontroller. Forwarding is rudimentary: all bundles are forwarded on a hard-coded default route.

3.2.4. Dedicated bundle router

Each computer constitutes a single bundle node that functions solely as a high-performance bundle forwarder. Many standard functions of the bundle protocol agent, the convergence layer adapters, and the administrative element of the application agent are implemented in specialized hardware, but some functions are implemented in a high-speed processor to enable reprogramming as necessary. The node's application agent has no application-specific element. Substantial non-volatile storage resources are provided, and arbitrarily complex forwarding algorithms are supported.

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.

4. Bundle Format

Each bundle shall be a concatenated sequence of at least two block structures. The first block in the sequence must be a primary bundle block, and no bundle may have more than one primary bundle block. Additional bundle protocol blocks of other types may follow the primary block to support extensions to the bundle protocol, such as the Bundle Security Protocol [BSP]. Exactly one of the blocks in the sequence must be a payload block. The last block in the sequence must have the "last block" flag (in its block processing control flags) set to 1; for every other block in the bundle after the primary block, this flag must be set to zero.

4.1. Self-Delimiting Numeric Values (SDNVs)

The design of the bundle protocol attempts to reconcile minimal consumption of transmission bandwidth with:

- . extensibility to address requirements not yet identified, and
- . scalability across a wide range of network scales and payload sizes.

A key strategic element in the design is the use of self-delimiting numeric values (SDNVs). The SDNV encoding scheme is closely adapted from the Abstract Syntax Notation One Basic Encoding Rules for sub-identifiers within an object identifier value [ASN1]. An SDNV is a numeric value encoded in N octets, the last of which has its most significant bit (MSB) set to zero; the MSB of every other octet in the SDNV must be set to 1. The value encoded in an SDNV is the unsigned binary number obtained by concatenating into a single bit string the 7 least significant bits of each octet of the SDNV. The following examples illustrate the encoding scheme for various hexadecimal values.

0xABC : 1010 1011 1100

is encoded as

{1 00 10101} {0 0111100}

= 10010101 00111100

0x1234 : 0001 0010 0011 0100

= 1 0010 0011 0100

is encoded as

```

    {1 0 100100} {0 0110100}
    = 10100100 00110100
0x4234 : 0100 0010 0011 0100
    = 100 0010 0011 0100
    is encoded as
    {1 000000 1} {1 0000100} {0 0110100}
    = 10000001 10000100 00110100
0x7F : 0111 1111
    = 111 1111
    is encoded as
    {0 1111111}
    = 01111111

```

Figure 2: SDNV Example

Note: Care must be taken to make sure that the value to be encoded is (in concept) padded with high-order zero bits to make its bitwise length a multiple of 7 before encoding. Also note that, while there is no theoretical limit on the size of an SDNV field, the overhead of the SDNV scheme is 1:7, i.e., one bit of overhead for every 7 bits of actual data to be encoded. Thus, a 7-octet value (a 56-bit quantity with no leading zeroes) would be encoded in an 8-octet SDNV; an 8-octet value (a 64-bit quantity with no leading zeroes) would be encoded in a 10-octet SDNV (one octet containing the high-order bit of the value padded with six leading zero bits, followed by nine octets containing the remaining 63 bits of the value). 148 bits of overhead would be consumed in encoding a 1024-bit RSA encryption key directly in an SDNV. In general, an N-bit quantity with no leading zeroes is encoded in an SDNV occupying $\text{ceil}(N/7)$ octets, where ceil is the integer ceiling function.

Implementations of the bundle protocol may handle as an invalid numeric value any SDNV that encodes an integer larger than $(2^{64} - 1)$.

An SDNV can be used to represent both very large and very small integer values. However, SDNV is clearly not the best way to represent every numeric value. For example, an SDNV is a poor way to represent an integer whose value typically falls in the range 128 to 255. In general, though, we believe that SDNV representation of numeric values in bundle blocks yields the smallest block sizes without sacrificing scalability.

4.2. Bundle Processing Control Flags

The bundle processing control flags field in the primary bundle block of each bundle is an SDNV; the value encoded in this SDNV is a string of bits used to invoke selected bundle processing control features. The significance of the value in each currently defined position of this bit string is described here. Note that in the figure and descriptions, the bit label numbers denote position (from least significant ('0') to most significant) within the decoded bit string, and not within the representation of the bits on the wire. This is why the descriptions in this section and the next do not follow standard RFC conventions with bit 0 on the left; if fields are added in the future, the SDNV will grow to the left, and using this representation allows the references here to remain valid.

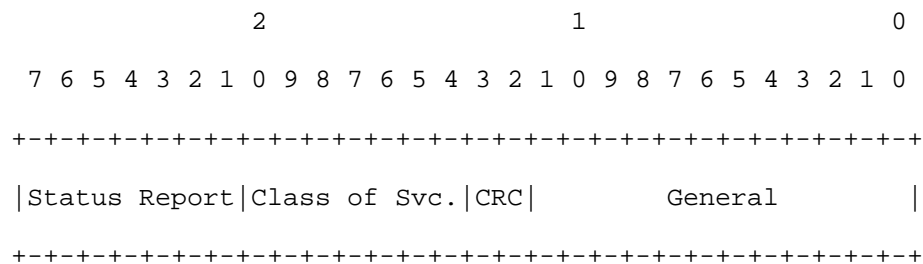


Figure 3: Bundle Processing Control Flags Bit Layout

The bits in positions 0 through 13 of the value of the bundle processing control flags SDNV are flags that characterize the bundle as follows:

- ```
0 -- Bundle is a fragment.
1 -- Payload is an administrative record.
2 -- Bundle must not be fragmented.
3 -- Custody transfer is requested.
```



- 4 -- Destination endpoint is a singleton.
- 5 -- Acknowledgement by application is requested.
- 6 -- Bundle is critical.
- 7 -- Best-efforts forwarding is requested.
- 8 -- Reliable forwarding is requested.
- 9-11 -- Reserved for future use.

The bits in positions 12 through 13 are used to indicate the type of CRC that is present at the end of the primary block. The options are:

- 0 -- No CRC.
- 1 -- CRC-8.
- 2 -- CRC-16.
- 3 -- CRC-32.

The bits in positions 14 through 20 are used to indicate the bundle's class of service. They constitute a seven-bit priority field indicating the bundle's priority, a value from 0 to 127, with higher values being of higher priority (greater urgency). Within this field, bit 20 is the most significant bit.

The bits in positions 21 through 27 are status report request flags. These flags are used to request status reports as follows:

- 21 -- Request reporting of bundle reception.
- 22 -- Request reporting of custody acceptance.
- 23 -- Request reporting of bundle forwarding.
- 24 -- Request reporting of bundle delivery.
- 25 -- Request reporting of bundle deletion.
- 26 -- Reserved for future use.
- 27 -- Reserved for future use.

If the bundle processing control flags indicate that the bundle's application data unit is an administrative record, then the custody transfer requested flag must be zero and all status report request flags must be zero. If the custody transfer requested flag is 1, then the source node requests that every receiving node accept custody of the bundle. If the bundle's source endpoint is the null endpoint (see below), then the bundle is not uniquely identifiable and all bundle protocol features that rely on bundle identity must therefore be disabled: the bundle's custody transfer requested flag must be zero, the "Bundle must not be fragmented" flag must be 1, and all status report request flags must be zero.

#### 4.3. Block Processing Control Flags

The block processing control flags field in every block other than the primary bundle block is an SDNV; the value encoded in this SDNV is a string of bits used to invoke selected block processing control features. The significance of the values in all currently defined positions of this bit string, in order from least significant position in the decoded bit string (labeled '0') to most significant (labeled '6'), is described here.



Figure 4: Block Processing Control Flags Bit Layout

- 0 - Block must be replicated in every fragment.
- 1 - Transmit status report if block can't be processed.
- 2 - Delete bundle if block can't be processed.
- 3 - Last block.
- 4 - Discard block if it can't be processed.
- 5 - Block was forwarded without being processed.
- 6 - Reserved for future use.

For each bundle whose primary block's bundle processing control flags (see above) indicate that the bundle's application data unit is an administrative record, the "Transmit status report if block can't be processed" flag in the block processing flags field of every other block in the bundle must be zero.

The 'Block must be replicated in every fragment' bit in the block processing flags must be set to zero on all blocks that follow the payload block.

#### 4.4. Identifiers

##### 4.4.1. Endpoint ID

The destinations of bundles are bundle endpoints, identified by text strings termed "endpoint IDs" (see Section 3.1). Each endpoint ID (EID) conveyed in any bundle block takes the form of 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. The set of allowable schemes is effectively unlimited. Any scheme conforming to [URIREG] may be used in a bundle protocol endpoint ID.

As used for the purposes of the bundle protocol, the length of an SSP must not exceed 1023 bytes.

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]).

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

Whenever an endpoint ID appears in a bundle block, it is encoded not in its native URI representation but rather in an encoded representation that reduces consumption of transmission bandwidth. The encoded representation of an endpoint ID is as follows:

- . An SDNV identifying the scheme of the EID (as discussed below), followed by
- . the encoded representation of the EID's scheme-specific part.

The encoded representation of the null endpoint ID is scheme identifier zero, followed by zero octets of scheme-specific part.

Every URI scheme used for forming any other EID is classified as either "numeric", meaning that all information conveyed in the scheme-specific part is to be encoded as a sequence of one or more unsigned integers in SDNV representation, or else "non-numeric" (otherwise). The scheme identifier numbers used in the encoded representations of EIDs are assigned as follows:

- . Scheme identifier zero is reserved for the null endpoint ID.
- . Scheme identifier numbers in the range 1-63 are used exclusively for numeric EID schemes.
- . All other scheme identifier numbers are used exclusively for non-numeric EID schemes.

Note that scheme of the EID is numeric if and only if the scheme identifier is non-zero and the two high-order bits of the first octet of the scheme identifier are both zero.

For each numeric EID scheme, the encoded representation of the EID's scheme-specific part shall be a sequence of from 1 to 100 SDNVs as mandated by the definition of the scheme.

For each non-numeric EID scheme, the encoded representation of the EID's scheme-specific part shall comprise:

- . a single SDNV indicating the length of the remainder of the encoded representation of the scheme-specific part of the EID, followed by
- . the remainder of the encoded representation of the scheme-specific part of the EID, formed according to the definition of the scheme. If the scheme's definition does not include a specification for encoded representation, then the EID's native scheme-specific part appears here without alteration.

It is important to note that not all BP implementations are required to implement the definitions of all EID schemes. The BP implementations used to instantiate nodes in a given network must be chosen with care in order for every node to be able to exchange bundles with every other node.

#### 4.4.2. 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 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, subject to the following restrictions:

- . Every node must be a member of at least one singleton endpoint.
- . The EID of any singleton endpoint of which a node is a member may be used to identify that node. A "node ID" is an EID that is used in this way.
- . A node's membership in a given singleton endpoint must be sustained at least until the nominal operation of the Bundle Protocol no longer depends on the identification of that node by that endpoint's ID.

#### 4.5. Formats of Bundle Blocks

This section describes the formats of the primary block and payload block. 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 [BSP]) may require that BP implementations conforming to those protocols construct and process additional blocks.

The format of these two basic BP blocks is shown in Figure 5 below.

##### Primary Bundle Block

|                           |                                        |  |                             |
|---------------------------|----------------------------------------|--|-----------------------------|
| +-----+-----+-----+-----+ |                                        |  |                             |
| Version                   | Block length                           |  | Bundle Processing flags (*) |
| +-----+-----+-----+-----+ |                                        |  |                             |
|                           | Destination EID (*)                    |  | Source Node ID (*)          |
| +-----+-----+-----+-----+ |                                        |  |                             |
|                           | Report-to EID (*)                      |  | Creation timestamp time (*) |
| +-----+-----+-----+-----+ |                                        |  |                             |
|                           | Creation Timestamp sequence number (*) |  | Lifetime (*)                |
| +-----+-----+-----+-----+ |                                        |  |                             |

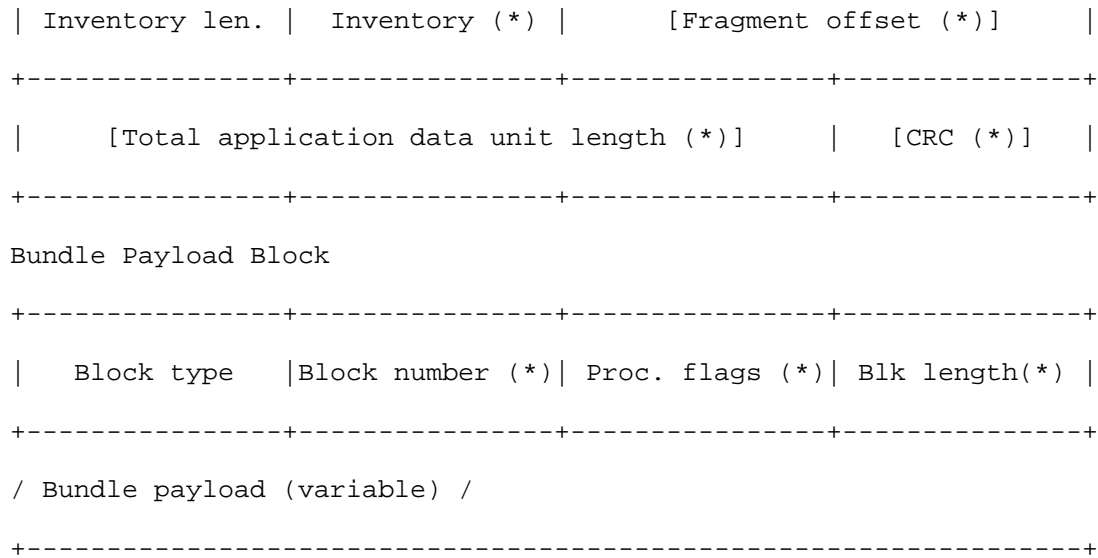


Figure 5: Basic Bundle Block Formats

(\*) Notes:

The bundle processing control flags field in the Primary Bundle Block is an SDNV and is therefore of variable length. A two-octet SDNV is shown here for convenience in representation.

The destination EID, source node ID, and report-to EID in the Primary Bundle Block are EIDs in encoded representation and are therefore of variable length. Two-octet fields are shown here for convenience in representation.

The creation timestamp time in the Primary Bundle Block is an SDNV and is therefore of variable length. A two-octet SDNV is shown here for convenience in representation.

The creation timestamp sequence number field in the Primary Bundle Block is an SDNV and is therefore of variable length. A three-octet SDNV is shown here for convenience in representation.

The lifetime field in the Primary Bundle Block is an SDNV and is therefore of variable length. A one-octet SDNV is shown here for convenience in representation.

The inventory field in the Primary Bundle Block is an array of block types (one octet each) whose length is given by the value of the

Inventory Length field and is therefore variable. A one-octet inventory array is shown here for convenience in representation.

The fragment offset field of the Primary Bundle Block is present only if the Fragment flag in the block's processing flags field is set to 1. It is an SDNV and is therefore of variable length; a two-octet SDNV is shown here for convenience in representation.

The total application data unit length field of the Primary Bundle Block is present only if the Fragment flag in the block's processing flags field is set to 1. It is an SDNV and is therefore of variable length; a three-octet SDNV is shown here for convenience in representation.

The CRC field of the Primary Bundle Block is present only if the CRC type field in the block's processing flags field is non-zero. Its actual length depends on the CRC type; a one-octet CRC is shown here for convenience in representation.

The block processing control flags ("Proc. flags") field of the Payload Block is an SDNV and is therefore of variable length. A one-octet SDNV is shown here for convenience in representation.

The block length ("Blk length") field of the Payload Block is an SDNV and is therefore of variable length. A one-octet SDNV is shown here for convenience in representation.

#### 4.5.1. Primary Bundle Block

The primary bundle block contains the basic information needed to forward bundles to their destinations. The fields of the primary bundle block are:

Version: A 4-bit field indicating the version of the bundle protocol that constructed this block. The present document describes version 0x07 of the bundle protocol.

Block Length: a 12-bit field that contains the aggregate length (in bytes) of all remaining fields of the primary block. Note that, although many fields of the primary bundle block are variable-length SDNVs, the lengths of all of these SDNVs are in practice limited; the lengths of the scheme-specific parts of non-numeric EIDs are likewise limited. These limitations make it reasonable to limit the total length of the primary block to 4095 octets.

**Bundle Processing Control Flags:** The Bundle Processing Control Flags field is an SDNV that contains the bundle processing control flags discussed in Section 4.2 above.

**Destination EID:** The Destination EID field contains the encoded representation of the endpoint ID of the bundle's destination, i.e., the endpoint containing the node(s) at which the bundle is to be delivered.

**Source node ID:** The Source node ID field contains the encoded representation of an endpoint ID that identifies the node from which the bundle was initially transmitted, except that it may contain the null endpoint ID in the event that the bundle's source chooses to remain anonymous.

**Report-to EID:** The Report-to EID field contains the encoded representation of the ID of the endpoint to which status reports pertaining to the forwarding and delivery of this bundle are to be transmitted.

**Creation Timestamp:** The creation timestamp is a pair of SDNVs 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. The first SDNV of the timestamp is the bundle's creation time, while the second is the bundle's creation timestamp sequence number. Bundle creation time is the time -- expressed in seconds since the start of the year 2000, on the Coordinated Universal Time (UTC) scale [UTC] -- at which the transmission request was received that resulted in the creation of the bundle. Sequence count is the latest value (as of the time at which that transmission request was received) of a monotonically increasing positive integer counter managed by the source node's bundle protocol agent that may be reset to zero whenever the current time advances by one second. For nodes that lack accurate clocks (that is, nodes that are not at all moments able to determine the current UTC time to within 30 seconds), bundle creation time MUST be set to zero and the counter used as the source of the bundle sequence count MUST NEVER be reset to zero. In either case, a source Bundle Protocol Agent must never create two distinct bundles with the same source node ID and bundle creation timestamp. 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).

**Lifetime:** The lifetime field is an SDNV that indicates the time at which the bundle's payload will no longer be useful, encoded as a number of seconds past the creation time. When bundle's age exceeds



its lifetime, bundle nodes need no longer retain or forward the bundle; the bundle SHOULD be deleted from the network.

**Inventory:** The Primary block may contain an accounting of all blocks that were in the bundle at the time it was transmitted from the source node. This accounting comprises an inventory list length (an SDNV) followed by an inventory list (an array of N octets, where N is the value of the inventory list length). This feature is optional: if the inventory is to be omitted, the inventory length must be set to zero. Otherwise the values of the octets in the inventory list must be the block types of all of the non-primary blocks in the bundle as originally transmitted, exactly one list element per block. Since a bundle may contain multiple instances of a given block type, multiple elements of the inventory list may have the same value. The order of block types appearing in the inventory list is undefined.

**Fragment Offset:** If the Bundle Processing Control Flags of this Primary block indicate that the bundle is a fragment, then the Fragment Offset field is an SDNV indicating the offset from the start of the original application data unit at which the bytes comprising the payload of this bundle were located. If not, then the Fragment Offset field is omitted from the block.

**Total Application Data Unit Length:** If the Bundle Processing Control Flags of this Primary block indicate that the bundle is a fragment, then the Total Application Data Unit Length field is an SDNV indicating the total length of the original application data unit of which this bundle's payload is a part. If not, then the Total Application Data Unit Length field is omitted from the block.

**CRC:** If and only if the CRC type in the Bundle Processing Control Flags of this Primary block is non-zero, a CRC is appended to the primary block. The length of the CRC is 8 bits, 16 bits, or 32 bits as indicated by the CRC type. The CRC is computed over the concatenation of all bytes of the primary block including the CRC field itself, which for this purpose is temporarily populated with the value zero.

#### 4.5.2. Canonical Bundle Block Format

Every bundle block of every type other than the primary bundle block comprises the following fields, in this order:

- . Block type code, expressed as an 8-bit unsigned binary integer. Bundle block type code 1 indicates that the block is a bundle payload block. Block type codes 2 through 10 are defined as

- noted later in this specification. Block type codes 192 through 255 are not defined in this specification and are available for private and/or experimental use. All other values of the block type code are reserved for future use.
- . Block number, an unsigned integer expressed as an SDNV. The block number uniquely identifies the block within the bundle, enabling blocks (notably bundle security protocol blocks) to explicitly reference other blocks in the same bundle. Block numbers need not be in continuous sequence, and blocks need not appear in block number sequence in the bundle. The block number of the payload block is always zero.
  - . Block processing control flags, an unsigned integer expressed as an SDNV. The individual bits of this integer are used to invoke selected block processing control features.
  - . Block data length, an unsigned integer expressed as an SDNV. The Block data length field contains the aggregate length of all remaining fields of the block, i.e., the block-type-specific data fields.
  - . Block-type-specific data fields, whose format and order are type-specific and whose aggregate length in octets is the value of the block data length field. All multi-byte block-type-specific data fields are represented in network byte order.

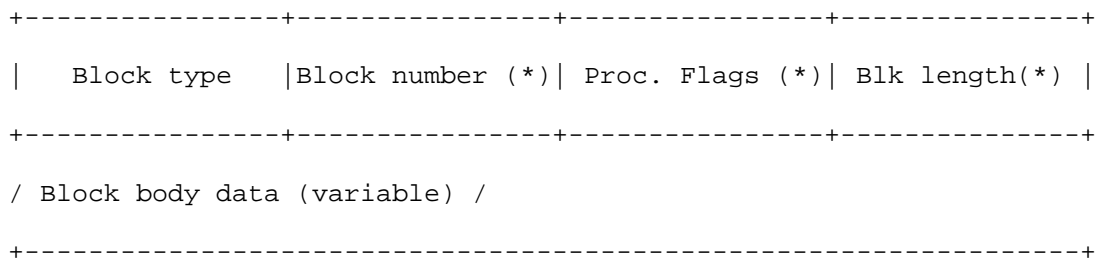


Figure 6: Block Layout

#### 4.5.3. Bundle Payload Block

The fields of the bundle payload block are:

**Block Type:** The Block Type field is a 1-byte field that indicates the type of the block. For the bundle payload block, this field contains the value 1.

**Block Number:** The Block Number field is an SDNV that contains the unique identifying number of the block. The block number of the bundle payload block is always zero.

**Block Processing Control Flags:** The Block Processing Control Flags field is an SDNV that contains the block processing control flags discussed in Section 4.3 above.

**Block Data Length:** The Block Data Length field is an SDNV that contains the aggregate length of all remaining fields of the Payload block - which is to say, the length of the bundle's payload.

**Block-type-specific Data:** The Block-type-specific Data field of the Payload Block contains the "payload", i.e., the application data carried by this bundle.

That is, bundle payload blocks conform to the canonical format described in the previous section.

#### 4.6. Extension Blocks

"Extension blocks" are all blocks other than the primary and payload blocks. Because not all extension blocks are defined in the Bundle Protocol specification (the present document), not all nodes conforming to this specification will necessarily instantiate Bundle Protocol implementations that include procedures for processing (that is, recognizing, parsing, acting on, and/or producing) all extension blocks. It is therefore possible for a node to receive a bundle that includes extension blocks that the node cannot process.

Whenever a bundle is forwarded that contains one or more extension blocks that could not be processed, the "Block was forwarded without being processed" flag must be set to 1 within the block processing flags of each such block. For each block flagged in this way, the flag may optionally be cleared (i.e., set to zero) by another node that subsequently receives the bundle and is able to process that block; the specifications defining the various extension blocks are expected to define the circumstances under which this flag may be cleared, if any.

The extension blocks of the Bundle Security Protocol (block types 2, 3, and 4) are defined separately in the Bundle Security Protocol specification (work in progress).

The following extension blocks are defined in the current document.

##### 4.6.1. Current Custodian

The Current Custodian block, block type 5, identifies a node that is known to have accepted custody of the bundle. The block-type-specific data of this block is the encoded representation of the

node ID of a custodian. The bundle MAY contain one or more occurrences of this type of block.

#### 4.6.2. Flow Label

The Flow Label block, block type 6, indicates the flow label that is intended to govern transmission of the bundle by convergence-layer adapters. The syntax and semantics of BP flow labels are beyond the scope of this document.

#### 4.6.3. Previous Node ID

The Previous Node ID block, block type 7, identifies the node that forwarded this bundle to the local node; its block-type-specific data is the encoded representation of the node ID of that node. If the local node is the source of the bundle, then the bundle MUST NOT contain any Previous Node ID block. Otherwise the bundle MUST contain one (1) occurrence of this type of block. If present, the Previous Node ID block MUST be the FIRST block following the primary block, as the processing of other extension blocks may depend on its value.

#### 4.6.4. Bundle Age

The Bundle Age block, block type 9, contains the number of seconds 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 SDNV containing the age of the bundle (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) in seconds. 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.

#### 4.6.5. Hop Count

The Hop Count block, block type 10, contains two SDNVs, hop limit and hop count, in that order. It 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: a bundle may be deleted when its hop count exceeds its hop limit. Procedures for determining the appropriate hop limit for a block are beyond the scope of this specification. A bundle MAY contain at most one (1) occurrence of this type of block.

## 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. That is, supplementary DTN protocol specifications (including, but not restricted to, the Bundle Security Protocol [BSP]) may require that additional measures be taken at specified junctures in these procedures. Such additional measures shall not override or supersede the mandated bundle protocol procedures, except that they may in some cases make these procedures moot by requiring, for example, that implementations conforming to the supplementary protocol terminate the processing of a given incoming or outgoing bundle due to a fault condition recognized by that protocol.

### 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 or a custody signal), 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.

Under some circumstances, the requesting of status reports could result in an unacceptable increase in the bundle traffic in the network. For this reason, the generation of status reports is mandatory only in one case, the deletion of a bundle for which custody transfer is requested. In all other cases, the decision on whether or not to generate a requested status report is left to the discretion of the bundle protocol agent. Mechanisms that could assist in making such decisions, 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 "custody acceptance status report" is a bundle status report with the "reporting node accepted custody of 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.
- . A "Succeeded" custody signal is a custody signal with the "custody transfer succeeded" flag set to 1.
- . A "Failed" custody signal is a custody signal with the "custody transfer succeeded" flag set to zero.
- . A "current custodian" of a bundle is a node identified in a Current Custodian extension block of that bundle.

## 5.2. Bundle Transmission

The steps in processing a bundle transmission request are:

Step 1: If custody transfer is requested for this bundle transmission then the destination must be a singleton endpoint. If, moreover, custody acceptance by the source node is required but the conditions under which custody of the bundle may be accepted are not satisfied, then the request cannot be honored and all remaining steps of this procedure must be skipped.

Step 2: 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 EID of a singleton endpoint whose only member is the node of which the BPA is a component or else the null endpoint ID, indicating that the source of the bundle is anonymous.

Step 3: Processing proceeds from Step 1 of Section 5.4.

## 5.3. Bundle Dispatching

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.

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 for any of the reasons listed in Figure 12. In particular:

- . The bundle protocol agent must determine which node(s) to forward the bundle to. 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 other state but in any case is beyond the scope of this document. If the BPA elects to forward the bundle to some other node(s) for further forwarding:
  - o If the "Bundle is critical" flag (in the bundle processing flags) is set to 1, then ALL nodes that have some plausible prospect of forwarding the bundle to its destination node(s) SHOULD be selected for this purpose.
  - o If the agent 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 select the convergence layer adapter(s) whose services will enable the node to send the bundle to those nodes. If both the "Best-efforts forwarding requested" and the "Reliable forwarding is requested" bundle processing flags are set to 1, then all selected CLAs MUST be for bundle streaming CL protocols such as the proposed Bundle Streaming Service Protocol. Otherwise, if only the "Reliable forwarding is requested" bundle processing flag is set to 1, then all selected CLAs MUST be for reliable protocols such as TCP/IP. Otherwise, if only the "Best-efforts forwarding requested" bundle processing flag is set to 1, then all selected CLAs MUST be for best-efforts protocols such as UDP/IP. Otherwise, any available CLAs may be selected. The manner in which specific appropriate convergence layer adapters are selected is beyond the scope of this document. If the agent finds it impossible to select appropriate convergence layer adapters 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 Figure 12, then the Forwarding Contraindicated procedure defined in Section 5.4.1 must be followed; the remaining steps of Section 5 are skipped at this time.

Step 4: If the bundle's custody transfer requested flag (in the bundle processing flags field) is set to 1, then the custody transfer procedure defined in Section 5.10.2 must be followed.

Step 5: 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 is to be sent by each convergence layer adapter is an implementation matter. Note that:

- . The order in which convergence layer adapters send bundles SHOULD normally conform to the priority indicated in each bundle's bundle processing control flags field: all bundles of priority 255 sent from any single source should be sent before all bundles of priority 254 sent from the same source and so on.
- . But if the bundle contains a flow label extension block then that flow label value may identify overriding procedures for determining the order in which convergence layer adapters must send bundles, e.g., considering bundle source when determining the order in which bundles are sent. The definition of such procedures is beyond the scope of this specification.
- . If the bundle has a bundle age block, then at the last possible moment before the CLA initiates conveyance of the bundle node 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 6: 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:

- . If the "request reporting of bundle forwarding" flag in the bundle's status report request field is set to 1, then a bundle forwarding status report should be generated, destined for the bundle's report-to endpoint ID. If the bundle has the retention constraint "custody accepted" and all of the nodes to which the bundle was forwarded are known to be unable to send bundles back to this node, then the reason code on this bundle forwarding status report must be "forwarded over unidirectional



- link"; otherwise, the reason code must be "no additional information".
- . The bundle's "Forward pending" retention constraint must be removed.

#### 5.4.1. Forwarding Contraindicated

The steps in responding to contraindication of forwarding for some reason are:

Step 1: The bundle protocol agent must determine whether or not to declare failure in forwarding the bundle for this reason. 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, (a) if the bundle's custody transfer requested flag (in the bundle processing flags field) is set to 1, then the custody transfer procedure defined in Section 5.10 MUST be followed; (b) when -- at some future time - the forwarding of this bundle ceases to be contraindicated, processing proceeds from Step 5 of Section 5.4.

#### 5.4.2. Forwarding Failed

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

Step 1: If the bundle's custody transfer requested flag (in the bundle processing flags field) is set to 1, custody transfer failure must be handled. The bundle protocol agent MUST handle the custody transfer failure by generating a "Failed" custody signal for the bundle, destined for the bundle's current custodian(s); the custody signal must contain a reason code corresponding to the reason for which forwarding was determined to be contraindicated. (Note that discarding the bundle will not delete it from the network, since each current custodian still has a copy.)

If the bundle's custody transfer requested flag (in the bundle processing flags field) is set to 0, then the bundle protocol agent MAY forward the bundle back to the node that sent it, as identified by the Previous Node ID block.

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 5.13 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. 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 (as discussed in section 4.5.1 above); 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": the bundle deletion procedure defined in Section 5.13 MUST be followed.

#### 5.6. Bundle Reception

The steps in processing a bundle 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, 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: 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, then a bundle reception status report with reason code "Block unintelligible" 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 unintelligible"; the bundle deletion procedure defined in Section 5.13 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 the block must be discarded, then the bundle protocol agent must set to 1 the "Block was forwarded without being processed" flag in the block processing flags of the block.

Step 4: If the bundle's custody transfer requested flag (in the bundle processing flags field) is set to 1 and the bundle has the same source node ID, creation timestamp, and (if the bundle is a fragment) fragment offset and payload length as another bundle that (a) has not been discarded and (b) currently has the retention constraint "Custody accepted", custody transfer redundancy must be handled. Otherwise, processing proceeds from Step 5. The bundle protocol agent must handle custody transfer redundancy by generating a "Failed" custody signal for this bundle with reason code "Redundant reception", destined for this bundle's current custodian, and removing this bundle's "Dispatch pending" retention constraint.

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 this bundle (whose fragmentary payload has been replaced by the reassembled application data unit) 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:

- . If the registration is in the Active state, then the bundle must be delivered subject to this registration (see Section 3.1 above) as soon as all previously received bundles that are deliverable subject to this registration have been delivered.
- . If the registration is in the Passive state, then the registration's delivery failure action must be taken (see Section 3.1 above).

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, 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.
- . If the bundle's custody transfer requested flag (in the bundle processing flags field) is set to 1, custodial delivery must be reported. The bundle protocol agent must report custodial delivery by generating a "Succeeded" custody signal for the bundle, destined for the bundle's current custodian(s).

#### 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 are the first  $N$  and the last  $(M - N)$  bytes of the original bundle's payload, where  $0 < N < M$ . Note that fragments may themselves be fragmented, so 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 that has any Current Custodian extension block citing any node other than the local node MUST NOT be fragmented. This restriction aside, 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.

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 bundle that was fragmented. Note that the payloads of fragments resulting from different fragmentation episodes, in different

parts of the network, may be overlapping subsets of the original bundle's payload.

- . The bundle processing flags in the primary block of each fragment must differ from those of the bundle that is being fragmented, in that they must indicate that the bundle is a fragment, and both fragment offset and total application data unit length must be provided at the end of each fragment's primary bundle block. Additionally, the CRC of the bundle that is being fragmented, if any, must be replaced in each fragment by a new CRC computed for the primary block of that fragment.
- . The primary blocks of the fragments will differ from that of the fragmented bundle as noted above.
- . The payload blocks of fragments will differ from that of the fragmented bundle as noted above.
- . If the bundle being fragmented is not a fragment or is the fragment with offset zero, then all extension blocks of the bundle being fragmented MUST be replicated in the fragment whose offset is zero.
- . Each extension block whose "Block must be replicated in every fragment" flag, in the block processing flags, is set to 1 MUST be replicated in every fragment.
- . Beyond these rules, replication of extension blocks in the fragments is an implementation matter.
- . If the local node had taken custody of the fragmented bundle, then the BPA MUST release custody of the fragmented bundle before fragmentation occurs and MUST take custody of every fragment.

#### 5.9. Application Data Unit Reassembly

If the concatenation -- as informed by fragment offsets and payload lengths -- of the payloads of all previously received fragments with the same source node ID and creation timestamp as this fragment, together with the payload of this fragment, forms a byte array whose length is equal to the total application data unit length in the fragment's primary block, then:

- . This byte array -- the reassembled application data unit -- must replace the payload of this fragment.
- . For each fragmentary bundle whose payload is a subset of the reassembled application data unit, for which custody transfer is requested but the BPA has not yet taken custody, the BPA must take custody of that bundle.
- . The BPA must then release custody of all fragments whose payload is a subset of the reassembled application data unit, for which it has taken custody.

- . The "Reassembly pending" retention constraint must be removed from every other fragment whose payload is a subset of the reassembled application data unit.

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. Custody Transfer

The decision as to whether or not to accept custody of a bundle is an implementation matter that may involve both resource and policy considerations.

If the bundle protocol agent elects to accept custody of the bundle, then it must follow the custody acceptance procedure defined in Section 5.10.1.

##### 5.10.1. Custody Acceptance

Procedures for acceptance of custody of a bundle are defined as follows.

The retention constraint "Custody accepted" must be added to the bundle.

If the "request reporting of custody acceptance" flag in the bundle's status report request field is set to 1, a custody acceptance status report should be generated, destined for the report-to endpoint ID of the bundle. However, if a bundle reception status report was generated for this bundle (Step 1 of Section 5.6), then this report SHOULD be generated by simply turning on the "Reporting node accepted custody of bundle" flag in that earlier report's status flags byte.

The bundle protocol agent must generate a "Succeeded" custody signal for the bundle, destined for the bundle's current custodian(s).

The bundle protocol agent must assert the new current custodian for the bundle. It does so by inserting a new Current Custodian extension block whose value is the node ID of the local node or by changing the value of an existing Current Custodian extension block to the local node ID.

The bundle protocol agent may set a custody transfer countdown timer for this bundle; upon expiration of this timer prior to expiration

of the bundle itself and prior to custody transfer success for this bundle, the custody transfer failure procedure detailed in Section 5.12 may be followed. The manner in which the countdown interval for such a timer is determined is an implementation matter.

The bundle should be retained in persistent storage if possible.

#### 5.10.2. Custody Release

When custody of a bundle is released, the "Custody accepted" retention constraint must be removed from the bundle and any custody transfer timer that has been established for this bundle should be destroyed.

#### 5.11. Custody Transfer Success

Upon receipt of a "Succeeded" custody signal at a node that is a custodial node of the bundle identified in the custody signal, custody of the bundle must be released as described in Section 5.10.2.

#### 5.12. Custody Transfer Failure

Custody transfer is determined to have failed at a custodial node for that bundle when either (a) that node's custody transfer timer for that bundle (if any) expires or (b) a "Failed" custody signal for that bundle is received at that node.

Upon determination of custody transfer failure, the action taken by the bundle protocol agent is implementation-specific and may depend on the nature of the failure. For example, if custody transfer failure was inferred from expiration of a custody transfer timer or was asserted by a "Failed" custody signal with the "Depleted storage" reason code, the bundle protocol agent might choose to re-forward the bundle, possibly on a different route (Section 5.4). Receipt of a "Failed" custody signal with the "Redundant reception" reason code, on the other hand, might cause the bundle protocol agent to release custody of the bundle and to revise its algorithm for computing countdown intervals for custody transfer timers.

#### 5.13. Bundle Deletion

The steps in deleting a bundle are:

Step 1: If the retention constraint "Custody accepted" currently prevents this bundle from being discarded, then:

- . Custody of the node is released as described in Section 5.10.2.
- . A bundle deletion status report citing the reason for deletion must be generated, destined for the bundle's report-to endpoint ID.

Otherwise, if the "request reporting of bundle deletion" flag in the bundle's status report request field is set to 1, 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.

#### 5.14. Discarding a Bundle

As soon as a bundle has no remaining retention constraints it may be discarded.

#### 5.15. 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 5.13 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. Two types of administrative records have been defined to date: bundle status reports and custody signals. Note that additional types of administrative records may be defined by supplementary DTN protocol specification documents.

Every administrative record consists of a five-bit record type code followed by three bits of administrative record flags, followed by record content in type-specific format. Record type codes are defined as follows:

|                                 |       |  |         |  |
|---------------------------------|-------|--|---------|--|
| +-----+-----+-----+-----+-----+ |       |  |         |  |
|                                 | Value |  | Meaning |  |
| +=====+=====+=====+=====+=====+ |       |  |         |  |



|                                 |                          |  |
|---------------------------------|--------------------------|--|
| 00001                           | Bundle status report.    |  |
| +-----+-----+-----+-----+-----+ |                          |  |
| 00010                           | Custody signal.          |  |
| +-----+-----+-----+-----+-----+ |                          |  |
| (other)                         | Reserved for future use. |  |
| +-----+-----+-----+-----+-----+ |                          |  |

Figure 8: Administrative Record Type Codes

|                                 |                                       |  |
|---------------------------------|---------------------------------------|--|
| +-----+-----+-----+-----+-----+ |                                       |  |
| Value                           | Meaning                               |  |
| +=====+=====+=====+=====+=====+ |                                       |  |
| 0001                            | Record is for a fragment; fragment    |  |
|                                 | offset and length fields are present. |  |
| +-----+-----+-----+-----+-----+ |                                       |  |
| (other)                         | Reserved for future use.              |  |
| +-----+-----+-----+-----+-----+ |                                       |  |

Figure 9: Administrative Record Flags

The contents of the two types of administrative records defined in the present document are described below.

#### 6.1.1.1. 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, custody transfer, forwarding, final delivery, and deletion. They are transmitted to the Report-to endpoints of bundles.

+-----+-----+-----+-----+-----+

|                                                           |                                  |                         |  |
|-----------------------------------------------------------|----------------------------------|-------------------------|--|
| Status Flags                                              | Reason code                      | Fragment offset (*) (if |  |
| present)                                                  | Fragment length (*) (if present) |                         |  |
| Source node ID of bundle X (*)                            |                                  |                         |  |
| Copy of bundle X's Creation Timestamp time (*)            |                                  |                         |  |
| Copy of bundle X's Creation Timestamp sequence number (*) |                                  |                         |  |

Figure 10: Bundle Status Report Format

(\*) Notes:

The Fragment Offset field, if present, is an SDNV and is therefore variable length. A three-octet SDNV is shown here for convenience in representation.

The Fragment Length field, if present, is an SDNV and is therefore variable length. A three-octet SDNV is shown here for convenience in representation.

The Source Node ID and Creation Timestamp fields replicate the Source Node ID and Creation Timestamp fields in the primary block of the subject bundle. As such they are of variable length. Four-octet values are shown here for convenience in representation.

The fields in a bundle status report are:

Status Flags: A 1-byte field containing the following flags:

|       |         |  |
|-------|---------|--|
| Value | Meaning |  |
|-------|---------|--|

|               |                                            |  |
|---------------|--------------------------------------------|--|
| 00000001      | Reporting node received bundle.            |  |
| +-----+-----+ |                                            |  |
| 00000010      | Reporting node accepted custody of bundle. |  |
| +-----+-----+ |                                            |  |
| 00000100      | Reporting node forwarded the bundle.       |  |
| +-----+-----+ |                                            |  |
| 00001000      | Reporting node delivered the bundle.       |  |
| +-----+-----+ |                                            |  |
| 00010000      | Reporting node deleted the bundle.         |  |
| +-----+-----+ |                                            |  |
| 00100000      | Unused.                                    |  |
| +-----+-----+ |                                            |  |
| 01000000      | Unused.                                    |  |
| +-----+-----+ |                                            |  |
| 10000000      | Unused.                                    |  |
| +-----+-----+ |                                            |  |

Figure 11: Status Flags for Bundle Status Reports

Reason Code: A 1-byte field explaining the value of the flags in the status flags byte. 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 [BSP]) may define additional reason codes. Status report reason codes are defined as follows:

|               |         |  |
|---------------|---------|--|
| +-----+-----+ |         |  |
| Value         | Meaning |  |
| +=====+       |         |  |

|                     |                                            |  |
|---------------------|--------------------------------------------|--|
| 0x00                | No additional information.                 |  |
| +-----+-----+-----+ |                                            |  |
| 0x01                | Lifetime expired.                          |  |
| +-----+-----+-----+ |                                            |  |
| 0x02                | Forwarded over unidirectional link.        |  |
| +-----+-----+-----+ |                                            |  |
| 0x03                | Transmission canceled.                     |  |
| +-----+-----+-----+ |                                            |  |
| 0x04                | Depleted storage.                          |  |
| +-----+-----+-----+ |                                            |  |
| 0x05                | Destination endpoint ID unintelligible.    |  |
| +-----+-----+-----+ |                                            |  |
| 0x06                | No known route to destination from here.   |  |
| +-----+-----+-----+ |                                            |  |
| 0x07                | No timely contact with next node on route. |  |
| +-----+-----+-----+ |                                            |  |
| 0x08                | Block unintelligible.                      |  |
| +-----+-----+-----+ |                                            |  |
| (other)             | Reserved for future use.                   |  |
| +-----+-----+-----+ |                                            |  |

Figure 12: Status Report Reason Codes

Fragment Offset: If the bundle fragment bit is set in the status flags, then the offset (within the original application data unit) of the payload of the bundle that caused the status report to be generated is included here.

Fragment length: If the bundle fragment bit is set in the status flags, then the length of the payload of the subject bundle is included here.

Source Node ID of Subject Bundle: The source node ID of the bundle that caused the status report to be generated.

Creation Timestamp of Subject Bundle: A copy of the creation timestamp of the bundle that caused the status report to be generated.

#### 6.1.2. Custody Signals

Custody signals are administrative records that effect custody transfer operations. They are transmitted to the nodes that are the current custodians of bundles.

Custody signals have the following format.

Custody signal regarding bundle 'X':

```

+-----+-----+-----+-----+
| Status | Fragment offset (*) (if present) |
+-----+-----+-----+-----+
| Fragment length (*) (if present) |
+-----+-----+-----+-----+
| Source node ID of bundle X (*) |
+-----+-----+-----+-----+
| Copy of bundle X's Creation Timestamp time (*) |
+-----+-----+-----+-----+
| Copy of bundle X's Creation Timestamp sequence number (*) |
+-----+-----+-----+-----+

```

Figure 13: Custody Signal Format

(\*) Notes:

The Fragment Offset field, if present, is an SDNV and is therefore variable length. A three-octet SDNV is shown here for convenience in representation.

The Fragment Length field, if present, is an SDNV and is therefore variable length. A four-octet SDNV is shown here for convenience in representation.

The Source Node ID and Creation Timestamp fields replicate the Source Node ID and Creation Timestamp fields in the primary block of the subject bundle. As such they are of variable length. Four-octet values are shown here for convenience in representation.

The fields in a custody signal are:

Status: A 1-byte field containing a 1-bit "custody transfer succeeded" flag followed by a 7-bit reason code explaining the value of that flag. Custody signal reason codes are defined as follows:

| +-----+-----+-----+-----+-----+-----+-----+ |                                          |  |
|---------------------------------------------|------------------------------------------|--|
| Value                                       | Meaning                                  |  |
| +=====+=====+=====+=====+=====+=====+=====+ |                                          |  |
| 0x00                                        | No additional information.               |  |
| +-----+-----+-----+-----+-----+-----+-----+ |                                          |  |
| 0x01                                        | Reserved for future use.                 |  |
| +-----+-----+-----+-----+-----+-----+-----+ |                                          |  |
| 0x02                                        | Reserved for future use.                 |  |
| +-----+-----+-----+-----+-----+-----+-----+ |                                          |  |
| 0x03                                        | Redundant (reception by a node that is a |  |
|                                             | custodial node for this bundle).         |  |
| +-----+-----+-----+-----+-----+-----+-----+ |                                          |  |
| 0x04                                        | Depleted storage.                        |  |
| +-----+-----+-----+-----+-----+-----+-----+ |                                          |  |

|         |                                            |         |
|---------|--------------------------------------------|---------|
| 0x05    | Destination endpoint ID unintelligible.    |         |
| +-----+ | +-----+                                    | +-----+ |
| 0x06    | No known route destination from here.      |         |
| +-----+ | +-----+                                    | +-----+ |
| 0x07    | No timely contact with next node on route. |         |
| +-----+ | +-----+                                    | +-----+ |
| 0x08    | Block unintelligible.                      |         |
| +-----+ | +-----+                                    | +-----+ |
| (other) | Reserved for future use.                   |         |
| +-----+ | +-----+                                    | +-----+ |

Figure 14: Custody Signal Reason Codes

Fragment offset: If the bundle fragment bit is set in the status flags, then the offset (within the original application data unit) of the payload of the bundle that caused the custody signal to be generated is included here.

Fragment length: If the bundle fragment bit is set in the status flags, then the length of the payload of the subject bundle is included here.

Source Node ID of Subject Bundle: The source node ID of the bundle that caused the custody signal to be generated.

Creation Timestamp of Subject Bundle: A copy of the creation timestamp of the bundle to which the signal applies.

## 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 with reference to some bundle, the following procedure must be followed:

Step 1: The administrative record must be constructed. If the referenced bundle is a fragment, the administrative record must have the Fragment flag set and must contain the fragment offset and

fragment length fields. The value of the fragment offset field must be the value of the referenced bundle's fragment offset, and the value of the fragment length field must be the length of the referenced bundle's payload.

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

### 6.3. Reception of Custody Signals

For each received custody signal that has the "custody transfer succeeded" flag set to 1, the administrative element of the application agent must direct the bundle protocol agent to follow the custody transfer success procedure in Section 5.11.

For each received custody signal that has the "custody transfer succeeded" flag set to 0, the administrative element of the application agent must direct the bundle protocol agent to follow the custody transfer failure procedure in Section 5.12.

## 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;
- . 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 [BSP]) may expect convergence layer adapters that



serve BP implementations conforming to those protocols to provide additional services such as 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.

## 8. Security Considerations

The bundle protocol has taken security into concern from the outset of its design. It was always assumed that security services would be needed in the use of the bundle protocol. As a result, the bundle protocol security architecture and the available security services are specified in an accompanying document, the Bundle Security Protocol specification [BSP]; an informative overview of this architecture is provided in [SEC0].

The bundle protocol has been designed with the notion that it will be run over networks with scarce resources. For example, the networks might have limited bandwidth, limited connectivity, constrained storage in relay nodes, etc. Therefore, the bundle protocol must ensure that only those entities authorized to send bundles over such constrained environments are actually allowed to do so. All unauthorized entities should be prevented from consuming valuable resources as soon as practicable.

Likewise, because of the potentially high latencies and delays involved in the networks that make use of the bundle protocol, data sources should be concerned with the integrity of the data received at the intended destination(s) and may also be concerned with ensuring confidentiality of the data as it traverses the network. Without integrity, the bundle payload data might be corrupted while in transit without the destination able to detect it. Similarly, the data source can be concerned with ensuring that the data can only be used by those authorized, hence the need for confidentiality.

Internal to the bundle-aware overlay network, the bundle nodes should be concerned with the authenticity of other bundle nodes as well as the preservation of bundle payload data integrity as it is forwarded between bundle nodes.

As a result, bundle security is concerned with the authenticity, integrity, and confidentiality of bundles conveyed among bundle nodes. This is accomplished via the use of two independent security-specific bundle blocks, which may be used together to provide multiple bundle security services or independently of one another, depending on perceived security threats, mandated security requirements, and security policies that must be enforced.

To provide end-to-end bundle authenticity and integrity, the Block Integrity Block (BIB) is used. The BIB allows any security-enabled entity along the delivery path to ensure the integrity of the bundle's payload or any other block other than a Block Confidentiality Block.

To provide payload confidentiality, the use of the Block Confidentiality Block (BCB) is available. The bundle payload, or any other block aside from the primary block and the Bundle Security Protocol blocks, may be encrypted to provide end-to-end payload confidentiality/privacy.

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.

Bundle security must not be invalidated by forwarding nodes even though they themselves might not use the Bundle Security Protocol.

In particular, while blocks may be added to bundles transiting intermediate nodes, removal of blocks with the 'Discard block if it can't be processed' flag unset in the block processing control flags may cause security to fail.

Inclusion of the Bundle Security Protocol in any Bundle Protocol implementation is RECOMMENDED. Use of the Bundle Security Protocol in Bundle Protocol operations is OPTIONAL.

## 9. IANA Considerations

The "dtn" and "ipn" URI schemes have been provisionally registered by IANA. See <http://www.iana.org/assignments/uri-schemes.html> for the latest details.

Registries of scheme type numbers, extension block type numbers, and administrative record type numbers will be required.

## 10. References

### 10.1. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[URI] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", RFC 3986, STD 66, January 2005.

[URIREG] Thaler, D., Hansen, T., and T. Hardie, "Guidelines and Registration Procedures for URI Schemes", RFC 7595, BCP 35, June 2015.

## 10.2. Informative References

[ARCH] V. Cerf et. al., "Delay-Tolerant Network Architecture", RFC 4838, April 2007.

[ASN1] "Abstract Syntax Notation One (ASN.1), "ASN.1 Encoding Rules: Specification of Basic Encoding Rules (BER), Canonical Encoding Rules (CER) and Distinguished Encoding Rules (DER)," ITU-T Rec. X.690 (2002) | ISO/IEC 8825- 1:2002", 2003.

[BSP] Symington, S., "Bundle Security Protocol Specification", Work Progress, October 2007.

[RFC3987] Duerst, M. and M. Suignard, "Internationalized Resource Identifiers (IRIs)", RFC 3987, January 2005.

[RFC5050] Scott, M. and S. Burleigh, "Bundle Protocol Specification", RFC 5050, November 2007.

[SECO] Farrell, S., Symington, S., Weiss, H., and P. Lovell, "Delay-Tolerant Networking Security Overview", Work Progress, July 2007.

[SIGC] Fall, K., "A Delay-Tolerant Network Architecture for Challenged Internets", SIGCOMM 2003.

[TUT] Warthman, F., "Delay-Tolerant Networks (DTNs): A Tutorial", <<http://www.dtnrg.org>>.

[UTC] Arias, E. and B. Guinot, "Coordinated universal time UTC: historical background and perspectives" in "Journées systèmes de référence spatio-temporels", 2004.

## 11. Acknowledgments

This work is freely adapted from [RFC5050], 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 Intel Research, Stephen Farrell of Trinity College Dublin, Peter Lovell of SPARTA, Inc., Manikantan Ramadas of Ohio University, and Howard Weiss of SPARTA, Inc.

This document was prepared using 2-Word-v2.0.template.dot.

## 12. 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.
- . Amplify discussion of custody transfer. Move current custodian to an extension block, of which there can be multiple occurrences (possible support for the MITRE idea of multiple concurrent custodians, from several years ago); define that block in this specification.
- . Introduce the concept of "node ID" as functionally distinct from endpoint ID, while having the same syntax.
- . Introduce a new method of encoding endpoint IDs (including node IDs) in a transmitted bundle, replacing both the "dictionary" and the CBHE compression mechanism.
- . Add ECOS features to primary block.
- . Restrict the scope of bundle prioritization to all bundles from the same source.
- . Restructure primary block, making it immutable. Add optional CRC and inventory.
- . Add optional CRCs to non-primary blocks.
- . Add block ID number to canonical block format (to support streamlined BSP).
- . Add bundle age extension block, defined in this specification.
- . Add previous node ID extension block, defined in this specification.
- . Add flow label block, \*not\* defined in this specification.
- . Add hop count extension block, defined in this specification.
- . Clean up a disconnect between fragmentation and custody transfer that Ed pointed out.
- . Remove "DTN time" values from admin records.

## 13. Open Issues

### 13.1. Definitions section structure

Would it be better to restrict the Definitions section to definitions only, and move description, conceptual operation,

potential implementation, justification, and commentary to one or more additional sections? Or would fractionating this information across multiple sections would make it harder to grasp?

#### 13.2. Payload nomenclature

It has been proposed that (a) there is no need to define "nominal payload" and (b) "partial" payload would be better than "fragmentary" payload. (The term "nominal payload" is used in the definition of fragmentation.)

#### 13.3. Application Agent

Does the discussion of Application Agent functionality need to be in the BP spec? If so, should there be a diagram explaining how the various components of the BPA interact?

#### 13.4. Bundle Endpoint definition

Is the source of a bundle always a node, or do we want to define a way in which a set of nodes (an endpoint) can collectively transmit a bundle? Does the latter trace back to a use case we need to support?

Is a bundle custodian always a node, or do we want to define a way in which a set of nodes (an endpoint) can collectively take custody of a bundle? Does the latter trace back to a use case we need to support?.

#### 13.5. Alignment with ICN

Is it necessary to modify the bundle transmission procedure to enable BP to be used for information-centric networking, i.e., delivering data to a node who requests that data after it has already been transmitted? Specifically, would a DTN ICN cache point "transmit" data to a client (i.e., source a new bundle) or would it merely "forward" a previously transmitted bundle of which it has retained a copy?

#### 13.6. Implementation Architectures

Should the BP spec be divided into two documents? One to talk about conops and context and one that focuses specifically on the protocol?

## 13.7. Security protocol name

Will the name of the DTN security protocol be Bundle Security Protocol or Streamlined Bundle Security Protocol?

## 13.8. Bundle format

Should the rules for defining block structure be presented at the start of section 4 or in the discussion of the payload block and the "last block" flag? Should we require that the payload block always be the last block of the bundle, so that the "last block" flag is no longer needed? (This would make reactive fragmentation easier.)

## 13.9. SDNVs

Should the SDNV discussion in 4.1 be deleted?

## 13.10. Bundle Processing Control Flags

Should the bit numbering convention described in section 4.2 be moved to another location in the document?

## 13.11. Extended class of service features

Should these features (critical bundle, best-efforts forwarding requested, reliable forwarding requested) be omitted from the primary block? If they are omitted, should these application-selected CoS markings be supported in some other way? If the "critical" CoS feature is retained, should it have a different name?

Note: a node selection (route computation) procedure might consider the availability of CLAs that match the bundle's CoS when selecting a node to forward to, and that is entirely the business of the route computation procedure. (Not all route computation procedures will do so.)

## 13.12. Primary block CRC type

What are the best CRC options to support here? CRC-16-ARINC, CRC-16-CCITT, CRC-16-CDMA2000, CRC-16-DECT, etc.? Are there more than 4?

## 13.13. Inventory

Is a list of the block types of all blocks in the bundle as forwarded by the source node a good implementation of the requested "inventory" feature? If not, what would be better?

## 13.14. Block numbers

Should the payload always have block number zero?

## 13.15. Clearing flag

Should an node that is able to process a given extension block be permitted to clear block's "Block was forwarded without being processed" flag?

## 13.16. Overriding BP spec

Is a supplementary DTN protocol specification allowed to override or supersede the BP specification (other than making some BP procedures moot by requiring that the processing of a bundle be terminated under fault conditions recognized by that protocol)?

## 13.17. Time of forwarding

Should the BPA control the time at which a bundle is to be forwarded to another node, or should that determination be left to the selected convergence-layer protocol adapter(s)?

## 13.18. Block multiplicity

Would it be good to restrict BP extensions to one extension block per extension per bundle? That is, should we require that all information needed to implement a given BP extension for a given bundle be contained in a single extension block?

This would entail encapsulating any necessary multiplicity for a given extension (for example, multiple Metadata records) within a single block.

Among the advantages: no need for block numbers (block type would always be sufficient to identify the block), therefore no need for a block number generation mechanism; shorter and simpler inventory; simpler extension implementation (all information is in one block, no need to search through extension blocks for additional relevant information).

Among the disadvantages: very different from RFC 5050; would in some cases require that security blocks operate on data structures that are internal to extension blocks rather than always operate on entire extension blocks.

## Appendix A.

## For More Information

Please refer comments to [dtn@ietf.org](mailto:dtn@ietf.org). The Delay Tolerant Networking Research Group (DTNRG) Web site is located at <http://www.dtnrg.org>.

Copyright (c) 2015 IETF Trust and the persons identified as authors of the code. All rights reserved.

Redistribution and use in source and binary forms, with or without modification, is permitted pursuant to, and subject to the license terms contained in, the Simplified BSD License set forth in Section 4.c of the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>).

## Authors' Addresses

Scott Burleigh  
Jet Propulsion Laboratory, California Institute of Technology  
4800 Oak Grove Dr.  
Pasadena, CA 91109-8099  
US  
Phone: +1 818 393 3353  
EMail: [Scott.Burleigh@jpl.nasa.gov](mailto:Scott.Burleigh@jpl.nasa.gov)

Kevin Fall  
Carnegie Mellon University / Software Engineering Institute  
4500 Fifth Avenue  
Pittsburgh, PA 15213  
US  
Phone: +1 412 268 3304  
Email: [kfall@cmu.edu](mailto:kfall@cmu.edu)

Edward J. Birrane  
Johns Hopkins University Applied Physics Laboratory  
11100 Johns Hopkins Rd  
Laurel, MD 20723  
US  
Phone: +1 443 778 7423  
Email: [Edward.Birrane@jhuapl.edu](mailto:Edward.Birrane@jhuapl.edu)





Network Working Group  
Internet-Draft  
Intended status: Informational  
Expires: August 29, 2015

F. Templin, Ed.  
Boeing Research & Technology  
S. Burleigh  
JPL, Calif. Inst. Of Technology  
February 25, 2015

DTN Security Key Management - Requirements and Design  
draft-templin-dtnskmreq-00.txt

Abstract

Delay/Disruption Tolerant Networking (DTN) introduces a network model in which communications may be subject to long delays and/or intermittent connectivity. These challenges render traditional security key management mechanisms infeasible since round trip delays may exceed the duration of communication opportunities. This document therefore proposes requirements and outlines a design for security key management in DTNs.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <http://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on August 29, 2015.

Copyright Notice

Copyright (c) 2015 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must

include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

## Table of Contents

|                                                                                                   |    |
|---------------------------------------------------------------------------------------------------|----|
| 1. Introduction . . . . .                                                                         | 2  |
| 2. Discussion . . . . .                                                                           | 4  |
| 3. DTN Security Key Management Core Requirements . . . . .                                        | 4  |
| 3.1. REQ1: Must Provide Keys When Needed . . . . .                                                | 4  |
| 3.2. REQ2: Must Be Trustworthy . . . . .                                                          | 5  |
| 3.3. REQ3: No Single Point of Failure . . . . .                                                   | 5  |
| 3.4. REQ4: Multiple Points of Authority . . . . .                                                 | 5  |
| 3.5. REQ5: No Veto . . . . .                                                                      | 5  |
| 3.6. REQ6: Must Bind Public Key with DTN Node Identity . . . . .                                  | 5  |
| 3.7. REQ7: Must Support Secure Bootstrapping of a Node's<br>Identity and its Public Key . . . . . | 6  |
| 3.8. REQ8: Must Support Revocation . . . . .                                                      | 6  |
| 3.9. REQ9: Revocations Must Be Delay Tolerant . . . . .                                           | 6  |
| 4. DTN Security Key Management Design Criteria . . . . .                                          | 6  |
| 4.1. DC1: Must Perform Timely Key Provisioning . . . . .                                          | 6  |
| 4.2. DC2: Pub/Sub Model . . . . .                                                                 | 6  |
| 4.3. DC3: Publication Must Be Spread Over Multiple KAs . . . . .                                  | 7  |
| 4.4. DC4: Availability and Security . . . . .                                                     | 7  |
| 5. Candidate DTN Security Key Management Design . . . . .                                         | 8  |
| 6. Limitations and Challenges . . . . .                                                           | 9  |
| 7. IANA Considerations . . . . .                                                                  | 10 |
| 8. Security Considerations . . . . .                                                              | 10 |
| 9. Acknowledgments . . . . .                                                                      | 11 |
| 10. References . . . . .                                                                          | 11 |
| 10.1. Normative References . . . . .                                                              | 11 |
| 10.2. Informative References . . . . .                                                            | 11 |
| Authors' Addresses . . . . .                                                                      | 12 |

## 1. Introduction

The Delay/Disruption Tolerant Network (DTN) architecture [RFC4838] introduces a data communications concept in which "bundles" of data are exchanged in store-and-forward fashion between endpoints that may be separated by long-delay or intermittently-connected paths. The Bundle Protocol Specification [RFC5050] provides the bundle message format and operations, including convergence layer transmission, fragmentation and custody transfer. Each bundle further may include extensions, among which may be security parameters designed to ensure confidentiality, integrity and authentication [RFC6257][I-D.irtf-dtnrg-sbsp]. These securing mechanisms (termed "Bundle Security Protocol") operate within the constraints imposed by various "ciphersuites". Prominent among these are ciphersuites that

rely on public/private key pairs where the public key is used to encrypt data and verify signatures while the private key is used to decrypt data and sign messages. Like any other public/private key system, however, Delay Tolerant Networks require some form of Public Key Infrastructure (PKI) to ensure that private key holders are properly authorized to use them as attested by a trusted Certificate Authority (CA) [RFC4210].

Public key cryptography in DTNs may be in some ways simpler than in traditional Internet security approaches. In particular, some BSP ciphersuites impose no need for peers to establish a long-term secret "symmetric" session key to be applied across a stream of bundles in the way that protocols such as the Internet Key Exchange (IKE) [RFC5996] establish session keys to be applied across a stream of packets. Instead, per the provisions of these ciphersuites, each bundle carries its own secret symmetric key in which the bundle is encrypted (in which case the symmetric key is itself encrypted in the public key of the receiver) or by which the bundle is signed (in which case the symmetric key is itself signed in the private key of the sender).

While the operation of the DTN securing mechanisms themselves can be applied independently of the key management scheme, in their current incarnation they can only be used with pre-placed irrevocable keys since there are no published mechanisms for automated security key management. On the surface, the use of standard PKI mechanisms would seem to be a natural fit, but traditional methods are not appropriate for long-delay and/or disrupted paths. This issue has prompted earlier IRTF investigations into an automated key management scheme for DTN [I-D.farrell-dtnrg-km][I-D.irtf-dtnrg-sec-overview], and it was also highlighted in "A Bundle of Problems" [WOOD08], Section 4.13 and "Security Analysis of DTN Architecture and Bundle Protocol Specification for Space-Based Networks" [IVAN09].

Therefore, an automated system for the publication and revocation of public keys will be necessary for many DTN applications, and that system must be designed to function in the presence of long delays and/or intermittent connectivity. The system must provide timely delivery of new public keys and security-key meta-data even though the delay inherent in the system may result in actual conveyance to DTN nodes long after transmission. Moreover the improper operation of this system, whether caused by malfunction or by a deliberate attack, could have significant impact on the usability of the network; the system must therefore be highly resistant to operational failure. In this document, we discuss the problem, provide requirements and propose a design for a suitable solution.

## 2. Discussion

Traditional automated PKI key management protocols allow for a subject (aka "end entity") to create a self-generated public/private key pair and then register the public key with a trusted Certificate Authority (CA) [RFC4210]. However, in a network based on DTN there may be significant delays between the time at which an end entity requests another entity's certificate and the time at which the requested certificate is delivered. Also, issues such as the publication of a new key pair can result in communication failures if end entities do not discover the new public key until some time after the old public key is deprecated. Alternatives such as a "web of trust" (e.g., via Pretty Good Privacy (PGP) [RFC4880]) may have application in some DTNs, but this is a topic for further study.

An old adage that also needs to be addressed is whether there is a "one-size-fits-all" solution. DTNs may come in various shapes and sizes, and various approaches may be better suited to some DTNs than others. More specifically, in the future there may not be one "DTN" in the same way that there is one public Internet. But rather, there may be many DTNs for public or private use - each with its own operational capabilities and constraints.

There will likely be ways to accomplish public key publication in the presence of long delays and/or disruptions, since keys can be published to take effect at some point in the future. However, timely certificate revocation may be infeasible due to the long delays inherent in many DTNs. DTN subjects therefore must be vigilant in ascertaining the degree to which long-delay correspondents can be trusted. These and many more issues must be carefully considered in any design.

## 3. DTN Security Key Management Core Requirements

A number of fundamental requirements must be satisfied by any security key management design for DTN. The requirements include the following:

### 3.1. REQ1: Must Provide Keys When Needed

The practical significance of this requirement is that the DTN security key management design must not rely on timely responses to queries directed to a Public Key Infrastructure (PKI). Low-delay online access using standard Internet connections (i.e., TCP/IP) may never be available. Even if the query is submitted using some delay-tolerant protocol, the opportunity to use the key to encrypt or verify data may have ended by the time the key arrives. In short, traditional PKIs are considered incompatible with DTN.

### 3.2. REQ2: Must Be Trustworthy

The design must be based on a trust anchor common to all nodes in the DTN network. A common trust anchor is needed to ensure that all DTN nodes will receive public keys from a secured key authority and not from an anonymous source. In particular, DTN nodes cannot simply accept public keys directly from one another with no prior trust basis. Otherwise, the network and all devices that use it could be compromised. The trust anchor should store and forward only authentic public keys from DTKA Key Authorities in an authentic manner so that the unavailability of DTKA Key Authorities will not prevent or delay communications between any two DTN nodes.

### 3.3. REQ3: No Single Point of Failure

The design must not introduce a single point of failure; the system must not fail in the event that one or more critical infrastructure elements are damaged. In particular, DTN nodes cannot always depend on receiving information from any single key authority node, since that node may not always be reachable over the network, may be subject to failures such as power outages, or may be compromised by an attacker. Much like the way RAID disc arrays operate, the system must be resilient to one or more failures.

### 3.4. REQ4: Multiple Points of Authority

The design must not introduce a single point of authority that could degrade the entire network if hijacked by an attacker. In particular, DTN nodes must never be forced to trust information provided by any single key authority node without corroboration by other key authority nodes.

### 3.5. REQ5: No Veto

Correspondingly, the design must never enable any single key authority node (possibly hijacked by an attacker) to degrade the network by declining to corroborate the information provided by other key authority nodes.

### 3.6. REQ6: Must Bind Public Key with DTN Node Identity

This requirement is about the claim for binding a public key with the ID of a DTN node. The key authority must certify the association of a public key with an identified DTN node when and only when that association is asserted by some entity that the key authority trusts. The mechanism by which such assertions are communicated must itself be secured. This requirement is a generic requirement for all secure Public Key Infrastructures.

### 3.7. REQ7: Must Support Secure Bootstrapping of a Node's Identity and its Public Key

The Key Authority must authorize the use of the association between a Node's identity and its public key, along with other administrative information, in its DTN. Such association is essentially random and cannot be verified in an automated manner. Thus, the association must be verified manually before the Key Authority can approve the use of the association in its DTN.

### 3.8. REQ8: Must Support Revocation

The DTN PKI must provide a mechanism that allows Certificate Authorities to revoke a certificate even before the certificate expires.

### 3.9. REQ9: Revocations Must Be Delay Tolerant

The propagation of information about revocation of issued and valid certificates must use DTN only. DTN certificate revocation must not assume the application will employ low-delay communications to verify public key certificates as is normal in the terrestrial Internet, where the Online Certificate Status Protocol (OCSP) is available to verify the absence of a public key in the revocation list in an on-demand manner.

## 4. DTN Security Key Management Design Criteria

We believe these core requirements imply several structural guidelines on security key management design for DTN. A candidate DTN security key management design can be formulated according to the following design criteria:

### 4.1. DC1: Must Perform Timely Key Provisioning

The design must ensure that security keys are put in place before they are actually needed. For example, if a source signs a bundle of data using its private key, each DTN node in the path may require access to the public key before the bundle arrives. Otherwise, the bundle could be rejected due to security policy. This means that DTN nodes must generate public/private key pairs and assert them to the key authority long in advance of when they would actually be needed.

### 4.2. DC2: Pub/Sub Model

The design must be based on a publish/subscribe model instead of an online (pull-based, or client/server) directory service, since on-demand retrieval from a traditional server is not possible in many

DTN environments due to delays/disruptions. One alternative is for the key authority to publish public key "bulletins" to which all DTN nodes subscribe. The bulletins must reach all DTN nodes in the network over the same long-delay links that carry ordinary data bundles. Bulletins therefore must convey keys to be used at some point in the future.

#### 4.3. DC3: Publication Must Be Spread Over Multiple KAs

The key management system's responsibility for distributing key information bulletins must be spread across multiple Key Authority Nodes (KAs); a monolithic bulletin generated by a single KA would violate requirements 3, 4, and 5. The cooperating KA nodes must publish fractionated data that can be aggregated to reconstitute the original bulletin; it must never be possible for the compromise of any single KA to result in reception of an inauthentic bulletin. Specifically, the KAs must agree on a bulletin through control message exchanges, after which each KA publishes a few overlapping fragments of the bulletin instead of the full bulletin. Each DTN node then receives the fragments and reassembles them into a complete bulletin. In this way, it is OK if one or more of the KAs fails because the fragments are overlapping and DTN nodes will be able to reconstruct the full bulletin. It is also OK if one or more of the KAs has been hacked, because the integrity of the bulletin will be ensured by the consensus agreement of all KAs. However, at least a few non-compromised KAs (functioning as trust anchors) must be present and reachable for the system to survive with assured integrity.

#### 4.4. DC4: Availability and Security

Like all other critical infrastructure elements, the key management system must be maintained as highly available and hardened against compromise. The latter requirement may require strong physical security, e.g., secured data centers, hardened mobile platforms, etc. This is no different than for other core network services such as the Domain Name System (DNS), Dynamic Host Configuration Protocol (DHCP) and many others. As in all other networking operations, nodes depend on at least occasional contact with critical infrastructure. Where fully ad-hoc networks are needed, dynamic key distribution may not be feasible. In that case, permanent Pre-Placed Keys (PPK) and/or limited-scope pairwise key exchanges may be the only solution alternatives.



## 5. Candidate DTN Security Key Management Design

We anticipate a security model for DTN that is based on ephemeral secret keys included on a per-bundle basis, i.e., in a similar manner as for S/MIME. That is, the symmetric keys used to secure DTN bundle traffic should normally be single-use (ephemeral) keys carried in individual bundles rather than persistent session keys. DTN nodes use public/private key pairs to encrypt/decrypt or sign/verify the ephemeral keys. The ephemeral keys are used to decrypt/authenticate bundle data efficiently.

In the design, DTN node public keys are registered with a Key Management System (KMS) that serves as the trust anchor for all secured DTN transactions. The KMS is organized as a group of N Key Authority (KA) nodes that act in an inter-dependent fashion to distribute public keys to all DTN nodes.

Each DTN node generates its own public/private key pair and registers the public key with the KMS. The KMS in turn issues key assertions and revocations in periodic bulletins sent via multicast transmissions to all DTN nodes. The keys are designated for use at some time in the future, since delays/disruptions may preclude immediate delivery.

Each KA node in the KMS has all current public key information for the DTN, but for each bulletin publication it sends only a subset of blocks (or "fragments") of the entire bulletin. Each bulletin is erasure-coded for Forward Error Correction (FEC) in case some fragments are lost, corrupted, or deemed untrustworthy. The resulting parity blocks for error detection are also included in the publication. Receivers then reassemble the bulletin from the union of fragments and parity blocks received, i.e., even if some fragments are lost, and extract time-tagged public keys from the bulletin.

In subsequent operation, the public key that a node uses to encrypt or sign an outbound bundle will be selected based on bundle creation time. The node must ensure that when it creates a bundle it is using a key that other nodes have been informed of. This means that each DTN node must cache keys for sufficiently long times to account for delays in the path.

DTN nodes must therefore keep track of all recently-received public keys for each potential peer node in the DTN. A DTN node that receives a bundle then uses the newest key that is no younger than the bundle creation time to verify or decrypt the ephemeral key included with the bundle.

Since multiple keys are retained at each node with different creation times, there is no need to synchronize key transmission and reception; the receiving node has the appropriate key in place long before the bundle arrives.

Additionally, no information in the key distribution system is kept secret - it's all public information. The point of the KMS is to provide a critical infrastructure trust basis so that DTN nodes can tell whether a prospective correspondent is authorized to use the public key it claims.

Security is then based on the DTN node's trust relationship with the KMS. As a result, all public keys are distributed securely. The KMS service is automated, with potential human intervention for revocation. No multi-message exchanges over long-delay links are needed (i.e., as for services such as the Internet Key Exchange (IKE) protocol), since ephemeral keys are used instead of session keys. The system also provides no single point of failure or compromise.

## 6. Limitations and Challenges

The candidate KMS design requires a scalable, reliable multicast capability. The DTN Bundle Protocol (BP) reliably delivers bundles to one or more recipients based on convergence layer protocols such as TCP and LTP. Reliable delivery in the BP is "hop-by-hop", where each hop needs to receive data reliably from the previous hop to ensure that end-to-end delivery is reliable. Scalable reliable multicast delivery is also based on hop-by-hop convergence layers, but large-scale reliable multicast is an end-to-end consideration that is not dealt with well in the Internet and needs to be better understood in the DTN context.

Security of the KMS is a fundamental requirement for service integrity. Just as for core Internet services (e.g., the DNS, DHCP, etc.), the KMS must be protected against network-based and physical security attacks. The system design is resilient to one or more elements being compromised, but bringing down all nodes essentially brings down the DTN. History has proven that services of this nature in the public Internet can be protected against comprehensive destruction, but measures must be taken to ensure network and physical security.

Another measure that may be considered in this context is KMS confederation. The KAs of a "local" KMS might forward bulletins to the KAs of another KMS as well as to the local node populations they serve. Such a structure would tend to make the KMS not only more durable but also more scalable.

Nodes that (re)enter the DTN after a long time away can present a challenging bootstrapping situation. Sometimes DTN nodes can go offline for extended periods of time (days/weeks/months), which would essentially bring the same consideration as for a new DTN node entering service for the first time. Upon (re)entering the DTN, the node has to publish its public key via the KMS. This "first contact" trust establishment is crucial to the security of the entire system, i.e., there needs to be a way for the new DTN node to trust the KMS, and for the KMS to validate the identity of the DTN node. In effect, a trusted entity (a node or a human) must somehow "vouch" for the new node.

DTN KMS services in fixed networks are not a problem, since the DTN topology does not change. On the other hand, Mobile Ad-hoc Networks (MANETs) typically show up in Unmanned Aerial Vehicle (UAV) networks, tactical military networks, etc. In that case, portions of the DTN may become detached from the rest of the DTN and re-attach at a different point of the DTN at a later time. This is more of a routing issue than a KMS issue, but routing aspects (especially in MANETs where there is no critical infrastructure) need to be understood.

Scaling considerations in terms of the size of the public key database must be analyzed on a per-DTN basis. For example, it may not be necessary for all DTN nodes to receive the public keys of all other DTN nodes since only a subset of all public keys may ever be needed. This is the same scaling consideration that motivated the design of the public Internet Domain Name System (DNS), when maintenance and distribution of a single, central repository at the SRI Network Information Center (SRI-NIC) became too unwieldy to maintain as the Internet grew exponentially.

## 7. IANA Considerations

There are no IANA considerations for this document.

## 8. Security Considerations

This document is entirely about security aspects of key management as a crucial component of DTN security; hence, security considerations appear throughout the document.

DTN security considerations are discussed in [RFC6257][I-D.irtf-dtnrg-sbsp].

## 9. Acknowledgments

Security key management has been discussed broadly in DTN mailing list discussions as well as in many of the documents cited in this publication. The candidate design discussed here is based on the original ideas of Scott Burleigh from NASA/JPL. Kapaleeswaran Viswanathan provided valuable review input.

## 10. References

### 10.1. Normative References

- [RFC0791] Postel, J., "Internet Protocol", STD 5, RFC 791, September 1981.
- [RFC2460] Deering, S. and R. Hinden, "Internet Protocol, Version 6 (IPv6) Specification", RFC 2460, December 1998.
- [RFC4838] Cerf, V., Burleigh, S., Hooke, A., Torgerson, L., Durst, R., Scott, K., Fall, K., and H. Weiss, "Delay-Tolerant Networking Architecture", RFC 4838, April 2007.
- [RFC5050] Scott, K. and S. Burleigh, "Bundle Protocol Specification", RFC 5050, November 2007.

### 10.2. Informative References

- [I-D.farrell-dtnrg-km]  
Farrell, S., "DTN Key Management Requirements", draft-farrell-dtnrg-km-00 (work in progress), June 2007.
- [I-D.irtf-dtnrg-sbsp]  
Birrane, E., "Streamlined Bundle Security Protocol Specification", draft-irtf-dtnrg-sbsp-01 (work in progress), May 2014.
- [I-D.irtf-dtnrg-sec-overview]  
Farrell, S., Symington, S., Weiss, H., and P. Lovell, "Delay-Tolerant Networking Security Overview", draft-irtf-dtnrg-sec-overview-06 (work in progress), March 2009.
- [IVAN09] Ivancic, W., "Security Analysis of DTN Architecture and Bundle Protocol Specification for Space-Based Networks", October 2009.
- [RFC4210] Adams, C., Farrell, S., Kause, T., and T. Mononen, "Internet X.509 Public Key Infrastructure Certificate Management Protocol (CMP)", RFC 4210, September 2005.

- [RFC4880] Callas, J., Donnerhackle, L., Finney, H., Shaw, D., and R. Thayer, "OpenPGP Message Format", RFC 4880, November 2007.
- [RFC5996] Kaufman, C., Hoffman, P., Nir, Y., and P. Eronen, "Internet Key Exchange Protocol Version 2 (IKEv2)", RFC 5996, September 2010.
- [RFC6257] Symington, S., Farrell, S., Weiss, H., and P. Lovell, "Bundle Security Protocol Specification", RFC 6257, May 2011.
- [WOOD08] Wood, L., Eddy, W., and P. Holliday, "A Bundle of Problems", December 2008.

#### Authors' Addresses

Fred L. Templin (editor)  
Boeing Research & Technology  
P.O. Box 3707  
Seattle, WA 98124  
USA

Email: [fltemplin@acm.org](mailto:fltemplin@acm.org)

Scott Burleigh  
JPL, Calif. Inst. Of Technology  
4800 Oak Grove Dr.  
Pasadena, CA 91109-8099  
USA

Phone: +1 818 393 3353  
Email: [Scott.Burleigh@jpl.nasa.gov](mailto:Scott.Burleigh@jpl.nasa.gov)

Network Working Group  
Internet-Draft  
Intended status: Standards Track  
Expires: February 28, 2016

K. Viswanathan  
F. Templin  
Boeing Research & Technology  
August 27, 2015

Architecture for a Delay-and-Disruption Tolerant Public-Key Distribution  
Network (PKDN)  
draft-viswanathan-dtnwg-pkdn-00.txt

Abstract

Delay/Disruption Tolerant Networking (DTN) introduces a network model in which communications can be subject to long delays and/or intermittent connectivity. DTN specifies the use of public-key cryptography to secure the confidentiality and integrity of messages in transit. The use of public-key cryptography posits the need for certification of public keys and revocation of certificates. This document formally defines the DTN key management problem and then provides a high-level design solution for delay and disruption tolerant distribution and revocation of public-key certificates along with relevant design options and recommendations for design choices.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <http://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on February 28, 2016.

Copyright Notice

Copyright (c) 2015 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of

publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

## Table of Contents

|                                                                      |    |
|----------------------------------------------------------------------|----|
| 1. Introduction . . . . .                                            | 2  |
| 1.1. Related Documents . . . . .                                     | 3  |
| 1.2. Terminology . . . . .                                           | 4  |
| 2. DTN Key Management . . . . .                                      | 6  |
| 2.1. The DTN-Key-Management Problem Statement . . . . .              | 6  |
| 2.2. Communication patterns for solving the DTN problem . . . . .    | 7  |
| 3. Architecture for Public Key Distribution Network (PKDN) . . . . . | 10 |
| 3.1. Design Choices for Routing Function . . . . .                   | 11 |
| 3.2. Design Choices for Cache Synchronization . . . . .              | 12 |
| 3.3. Design Choices for Revocation Updates . . . . .                 | 13 |
| 4. Summary of Recommended Design Choices . . . . .                   | 14 |
| 5. Future work . . . . .                                             | 15 |
| 6. IANA Considerations . . . . .                                     | 15 |
| 7. Security Considerations . . . . .                                 | 15 |
| 8. References . . . . .                                              | 15 |
| 8.1. Normative References . . . . .                                  | 15 |
| 8.2. Informative References . . . . .                                | 16 |
| Authors' Addresses . . . . .                                         | 17 |

## 1. Introduction

Key management protocols, for distribution and revocation of public keys on the terrestrial Internet, have required on-demand interactive communications, which has been realized using TCP [RFC0793] connections. The interactions in a public-key management system are between: (a) the sender (or owner of the public key) and the receiver (or user of the public key); and, (b) the receiver and a trusted authority (Certificate Authority or CA). On-demand messaging is not feasible on DTN. Therefore, terrestrial key management protocols may not always function as intended on DTN.

The Online Certificate Status Protocol (OCSP) [RFC6960], for example, requires the receiver of a public key certificate to have on-demand interactions with a Certification Authority (CA) in order to get the current status information for the certificate. Three status responses may be received by the receiver from the CA, namely: good, revoked, and unknown. The receiver needs to accept good certificates and reject revoked certificates. The CA sends a response indicating the unknown state usually when it does not recognize the issuer of

the certificate. In this case, the receiver is expected to interact on-demand with other CAs for determining if the certificate was revoked. When the status in the response is good, since the CA does not remember the receiver's interest in the certificate, the receiver is required to periodically request the status before every use of the certificate.

OCSP is a resource intensive protocol. In order to reduce the round-trip costs for the temporal validation of the certificates, especially in constrained clients (receivers), a provision in TLS Extensions (see Section 8) [RFC6066] has been proposed so that the senders shall send what is called a "stapled Certificate Status" to the receivers. The stapled Certificate Status is a time-stamped certificate-status certificate obtained from a trusted authority by the sender. If the constrained receiver (client) accepts the stapled Certificate Status, then it need not interact with any CA to ascertain the temporal validity of the certificate -- thus reducing communication costs on the receiver side. Although such proposals are useful when dealing with constrained clients (or receivers of certificate), they only transfer the burden of certificate-status queries towards the senders and away from the receivers. Such mechanisms do not obviate the need for on-demand interactions.

The Secure/Multi-purpose Internet Mail Extensions (S/MIME) [RFC5751] allows a sender to encapsulate its certificate as a meta-data (in the message header) for processing an email message. The receiver is expected to consult with a Certificate Revocation List (CRL) or other certificate status verification mechanisms to validate the temporal validity of the certificate. Thus, S/MIME does not obviate the need for on-demand interactions with remote trusted authorities.

As mentioned earlier, on-demand interactions with any party, trusted or otherwise, is not feasible in the network model for DTN. Therefore, existing terrestrial key management protocols are not suitable for DTN. This proposal describes the high-level design choices for a mechanism, which can satisfy the requirements for DTN Key Management [I-D.templin-dtnskmreq], that does not require on-demand interactions with remote parties.

### 1.1. Related Documents

The following documents provide the necessary context for the high-level design described in this document.

RFC 4838 [RFC4838] describes the architecture for DTN and is titled, "Delay-Tolerant Networking Architecture." That document provides a high-level overview of DTN architecture and the decisions that underpin the DTN architecture.



RFC 5050 [RFC5050] describes the protocol and message formats for DTN and is titled, "Bundle Protocol Specification." That document provides details for the protocol message format for DTN, which is called as Bundle, along with the description of processes for generating, sending, forwarding, and receiving Bundles. It also specifies an encoding format called SDNV (Self-Delimiting Numeric Values) for use in DTN.

RFC 6257 [RFC6257] is titled, "Bundle Security Protocol Specification." It specifies the message formats and processing rules for providing three types of security services to bundles, namely: confidentiality, integrity, and authentication. It does not specify mechanisms for key management. Rather, it assumes that cryptographic keys are somehow in place and then specifies how the keys shall be used to provide the security services. Additionally, it attempts to standardize the cipher suite in DTN.

The Internet Draft [I-D.birrane-dtn-sbsp] for DTN Key Management is titled, "Streamlined Bundle Security Protocol Specification (SBSP)." When compared with RFC 6257, it is silent on concepts such as Security Regions, at-most-once-delivery option, and cipher suite specification. It provides more detailed specification for bundle canonicalization and rules for processing bundles received from other nodes. Like RFC 6257, the draft does not describe any key management mechanisms for DTN but assumes that suitable key management mechanism shall be in place.

The Internet Draft for specifying requirements for DTN Key Management [I-D.templin-dtnskmreq] is titled, "DTN Security Key Management - Requirements and Design." It sketches nine requirements and four design criteria for DTN Key Management system. The last two requirements are the need to support revocation in a delay tolerant manner. It also specifies the requirements for avoiding single points of failure and opportunities for the presence of multiple key management authorities.

## 1.2. Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119]. Lower case uses of these words are not to be interpreted as carrying RFC2119 significance.

This draft introduces the following terminologies.

Public Key Distribution Network (PKDN)

is an overlay network that can operate on top of DTN. It is a network of trusted authorities that have information about temporal validity (revoked or otherwise) of public keys certificates. The objective of PKDN is the distribution of valid public-key certificates and revocation of invalidated public-key certificates in a secure, delay and disruption-tolerant manner.

#### PKDN Bundle

encapsulates a public-key certificate and can be transported in a DTN Bundle. PKDN bundle may optionally encapsulate one or more message payloads (or application data) that are authenticated using the public-key in the encapsulated certificate. The source of the PKDN bundle may provide confidentiality to the message payloads using the public-key of the intended receiver of the message payloads. The message payloads may be DTN Bundles.

#### PKDN Sender

is the source of a PKDN bundle. It generates PKDN bundles by encapsulating its public-key certificate and using the corresponding private key. It may optionally encapsulate authenticated message payloads in the PKDN bundle. It sends the PKDN bundle to a PKDN Router so that the bundle can be forwarded to the PKDN Receiver in designated the bundle.

#### PKDN Router

receives a PKDN Bundle from a PKDN Sender, validates it, and generates & forwards a Validated PKDN Bundle to the designated destination. Additionally, the PKDN Router records the destination's interest in the public-key certificate encapsulated in the PKDN Bundle so that it can send periodic status updates to the destination.

#### Validated PKDN Bundle

is generated by an authorized PKDN Router after receiving a PKDN Bundle that satisfies two conditions, namely: (a) it can be authenticated successfully using the encapsulated certificate; and, (b) revocation information for the encapsulated certificate is not available. A Validated PKDN Bundle includes the PKDN Bundle and a PKDN Bundle Validation Block (PBVB) generated by a PKDN Router. PBVB essentially includes the identity of PKDN Router and information for: (a) asserting the temporal validity of the public-key certificate encapsulated in the PKDN Bundle; (b) the time when the assertion was made; and, (c) message-origin authentication for the PKDN Bundle, the assertion of temporal validity, and the PKDN Router time.

#### PKDN Receiver

is the destination designated in the PKDN bundle and the node that shall consume the Validated PKDN Bundle. Upon validating the PKDN Bundle and verifying the Validated PKDN Bundle, the PKDN Receiver may store the encapsulated public-key certificate locally. Upon accepting the received Validated PKDN Bundle, it may optionally respond with an acknowledgement to the PKDN Sender via the PKDN Router, from which it received the Validated PKDN Bundle. The acknowledgement may include its own encapsulated public-key certificate and message payloads -- this would be the optional return path for the messaging.

#### Certificate Revocation Manager (CRM)

is an operationally off-line DTN node that shall maintain the System's Certificate Revocation List (CRL) and publish any changes to the CRL as Delta-CRLs. The CRM shall be housed in a physically protected location that is easily accessible for authorized and trusted human operators, who shall inject CRL updates into the CRM. The CRM, in-turn, shall inject the Delta-CRLs to PKDN Routers in the PKDN administered by the human operators. It is important to note that the CRM only propagates revocation information but not certificates. Certificates are propagated by the owners of the certificates, namely PKDN Senders.

## 2. DTN Key Management

This section shall introduce the problem statement for DTN Key Management problem followed by an enumeration of communication-patterns that can be used for potential solutions and a proposed solution for the problem that is called a Public-Key Distribution Network.

### 2.1. The DTN-Key-Management Problem Statement

The problem of DTN Key Management can be visualized as shown in Figure 1. The Receiver receives a public key certificate from the Sender. Since the Sender is not trusted to share timely revocation information, the Receiver needs to receive timely revocation information from a Trusted Authority. A basic problem is: (a) how can the Trusted Authority know when the Receiver needs the revocation information for a Public-Key Certificate; and, (b) how can periodic and consistent revocation information be availability in timely and delay-and-disruption tolerant manner? The second question gains importance in DTN because the delay and disruption in the communication link between the Sender and Receiver may not be correlatable with that between the Receiver and the Trusted Authority. This makes the DTN Key Management problem different from terrestrial key management systems, where communication links are assumed to be uniform, interactive, on-demand, and similar.

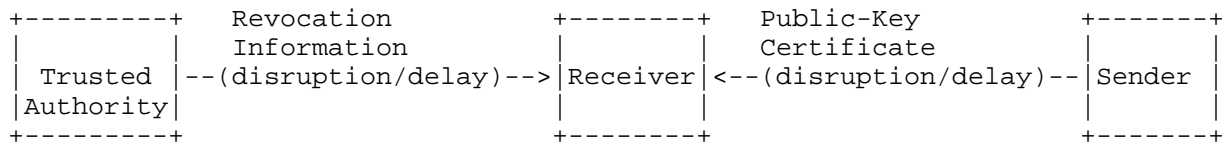


Figure 1: DTN Key Management Problem

An analysis of the above problem using CAP theorem [CAP] suggests that when network partition occurs, due to delay or disruption, the receiver needs to make a local decision in favour of either availability of its service for the received message or consistency of its operations in not accepting revoked certificate, which was used to provide integrity service to the received message. In other words, when the Receiver has received the public key certificate but has not received any revocation information as yet, it needs to vote in favour of either: (a) availability, by accepting the certificate without waiting for revocation information; or, (b) consistency, by waiting for the receipt of revocation information. If it votes in favour of availability, it risks the use of inconsistent information. If it votes in favour of consistency, it risks lack of availability of the public-key for some dependent information processing, which must be paused. Clearly, in the presence of delay and disruption, both consistency and availability cannot be achieved.

DTN Key Management solutions must be partition tolerant and provide trade-off options for their applications between availability and security consistency. Such a trade-off may be realized in an application-agnostic manner by aiming for eventual consistency instead of immediate consistency. Eventual consistency means that all DTN nodes will eventually reject revoked keys but until such an eventuality some DTN nodes are allowed to work with stale revocation information depending on their application security sensitivity. Immediate consistency is not possible in DTN but is possible in the terrestrial Internet. The time available for accepting or rejecting the certificate (and the message) will be decided by the application's security threshold.

## 2.2. Communication patterns for solving the DTN problem

As mentioned previously, the two-fold problem of DTN Key Management Problem is: (a) how can the Trusted Authority know when the Receiver needs the revocation information for a Public-Key Certificate; and, (b) how can periodic and consistent revocation information be made available in timely and delay-and-disruption tolerant manner?

Five communication patterns can provide solutions to the first question (Question a), namely:

- Pattern 1: (Request-response) The Receiver informs the Trusted Authority every time when it needs fresh revocation information for a certificate by sending a request. The Trust Authority responds with a fresh status information for that certificate.
- Pattern 2: (Publish-subscribe) The Receiver informs the Trusted Authority about its interest in a certificate only once, which is the first time when it needs the revocation information, by sending a subscription request. The Trusted Authority responds to the subscription request with a fresh status information for that certificate and remembers the subscription request. Whenever there is a change in status information, the Trusted Authority sends the updates to the Receiver without having to receive a request for the same.
- Pattern 3: (Blacklist broadcast) The Trusted Authority does not receive any certificate-specific request from any Receiver. It periodically broadcasts Certificate Revocation Lists (CRLs) to all DTN nodes including the Receiver. If the broadcast mechanism were to be replaced with a multicast mechanism, then the Receiver will be expected to register its address with the Trusted Authority exactly once as a registration process. Note that the registration process does not reference any certificate unlike the subscription process in the previous pattern.
- Pattern 4: (White-list broadcast) This communication pattern is similar to the previous communication pattern except that the Trusted Authority periodically broadcasts a list of valid certificates instead of broadcasting a list of invalidated certificates. This communication pattern is useful when the number of certified public-keys are less.
- Pattern 5: (Publish with proxy subscribe) The Sender routes its certificate through the Trusted Authority to the Receiver, who shall accept certificates only from the Trusted Authority. The Trusted Authority validates the certificate before forwarding it to the Receiver. The Trusted Authority subscribes the Receiver for interest in the Sender's certificate so that periodic updates can be sent in the future for the certificate. Thus, the Sender acts as a proxy for the Receiver and subscribes the Receiver for future updates from the Trusted Authority.

Pattern 1 describes the communication style used by terrestrial key management solutions such as OCSP. The Receiver may receive the certificate from the Sender every time a security session is established as is the case in TLS [RFC5246]. Thus, the Receiver may need to send a request to the Trusted Authority every time a security session is established. Section 1 discussed why this communication style is not suitable for DTN.

Pattern 2 has a similar complexity as Pattern 1 for the first round of communication for a certificate between the Receiver and the Trusted Authority. The communication complexity greatly eases from the second round onwards when the Trusted Authority can send updates to the Receiver without requiring a request. Although this pattern improves the communication complexity from the second round onwards, it does not improve communication complexity of the first round of communications, which is a bottleneck in the DTN settings as described for Pattern 1 in Section 1.

Patterns 3 and 4 require periodical broadcast/multicast of a list data structure (CRL or list of valid public keys). The efficiency of such patterns depend on three factors, namely: the size of the list of revoked certificates, the number of communication recipients, and the frequency of communication. If any one of these factor were to increase, bandwidth utilization will be inefficient because not all recipients of the communication may be interested in all elements of the list that they receive. Thus, most recipients will end up discarding many communications that they receive from the Trusted Authority. When two or more of the factors were to increase simultaneously, the communication system may be overloaded and normal application communications may be affected. Clearly, this solution is not scalable with the increase in number of recipients. Additionally, since Pattern 4 uses white-lists and, in public key management, white-lists grow more frequently than black-lists, the frequency of communications between the Trusted Authority and the Receivers will be higher than in Pattern 3. Also, since the Receivers depend on the Trusted Authority for timely delivery of white-listed keys, the first communication from the Sender to the Receiver must strictly happen after the Trusted Authority has sent the Sender's public key to the Receiver in a white-list communication. Otherwise, the Sender's communication will have to be rejected by the Receiver even though the Sender may be in possession of a registered (or authorized) public key. This calls for increased out-of-band delay-tolerant synchronization between the Sender and the Receiver. For reasons mentioned above, this document shall not pursue Patterns 3 and 4.

Pattern 5 requires every Sender to route their public-key certificates through the Trusted Authority to the Receiver. The

Trusted Authority can be a PKDN Router, which is allowed to filter communications with revoked public-key certificates. Additionally, the PKDN Router remembers the Receiver's interest in order to send periodic revocation updates for the forwarded public-key certificates. The rest of this document shall employ this communication pattern.

### 3. Architecture for Public Key Distribution Network (PKDN)

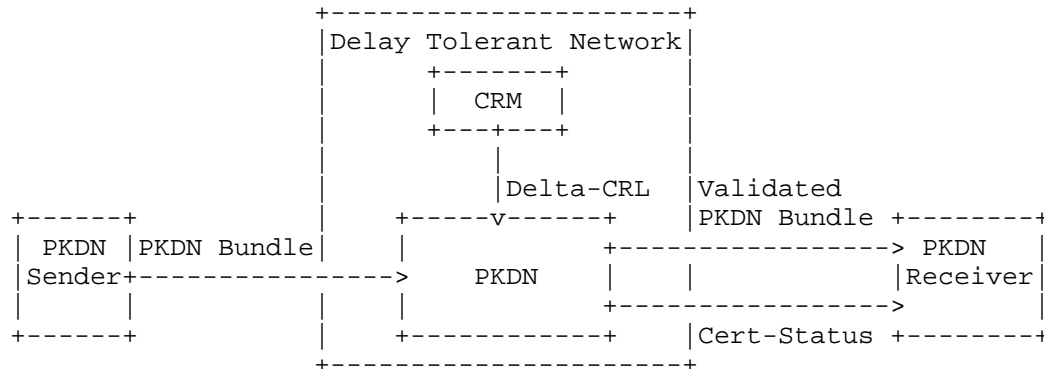


Figure 2: Architecture of Public Key Distribution Network

As mentioned in the previous section, this proposal adopts Communication Pattern 5 for designing Public Key Distribution Network (PKDN). The elements of PKDN are shown in Figure 2.

- a. An operationally off-line Certificate Revocation Manager (CRM) periodically injects timestamped updates to the System's Certificate Revocation Lists, called Delta-CRLs, into a few routers in the PKDN, which, in turn, shall propagate the updates to other routers in the PKDN. The information in the CRL needs to reach PKDN Receivers in part or full.
- b. PKDN is an overlay network on a Delay Tolerant Network (DTN) that is composed of a logical interconnection of PKDN Routers.
- c. PKDN Senders send PKDN Bundles to PKDN (PKDN Routers) so that the PKDN Bundles can be forwarded to PKDN Receivers.
- d. PKDN Receivers received Validated PKDN Bundles from PKDN and install the public key certificates in the PKDN Bundles locally. They also received PKDN Status messages from the PKDN

Within this architectural setting, loosely synchronized PKDN Routers perform three basic functions as described below.

1. (Routing) Receive and validate PKDN Bundles from Senders and forward Validated PKDN Bundles to Receivers designated in the PKDN Bundles.
2. (Cache synchronization) Update local CRL cache using authenticated and time-stamped CRL updates from authorized PKDN nodes. Such authenticated updates to certificate revocation lists shall be called delta-CRLs. Forward delta-CRLs to other PKDN Routers.
3. (Revocation updates) PKDN Routers construct and send periodic certificate status updates to PKDN Receivers using the local CRL cache.

The above three basic functions can now be used to enumerate the design choices for PKDN.

### 3.1. Design Choices for Routing Function

It was discussed that PKDN is an overlay network of PKDN routers. Also, a PKDN Bundle from a PKDN Sender to a PKDN Receiver needs to go through at least one PKDN Router. The following questions lead to the design choices for PKDN.

1. How many PKDN Routers must there be between any given PKDN Sender and PKDN Receiver? The answers can be one, two, or more. The higher the number of PKDN Routers, the higher will be the routing delay. In order to reduce delay, having only one PKDN Router in any given path for a PKDN Bundle is the best design choice.
2. How to determine the designated PKDN Router between a given PKDN Sender and PKDN Receiver? Naming of routers is fundamental to determining designated PKDN Router between two given communication endpoints. Two types of naming options have been considered for DTN [EPDTN], namely: (i) addresses with topological information; and, (ii) identifiers without topological information. The design choices for determining the designated PKDN Router are: (a) the PKDN Router name for a given PKDN Sender is manually configured for every PKDN Sender; (b) the PKDN Sender discovers the name of its nearest PKDN Router using a broadcast-based discovery protocol; (c) the PKDN Sender uses its DTN address to derive the DTN address of its PKDN Router; or, (d) the PKDN Sender uses the PKDN Receiver's DTN address to derive the DTN address of the PKDN Receiver's PKDN Router. When DTN node addresses with topological encoding are available, Options (c) and (d) provide non-interactive PKDN Router determination, which will be well suited for delay-and-disruption tolerance. To see how Options (c) and (d) may be designed, let's assume that the



address of PKDN Sender has the following encoded region and entity information: {region, entity:port} = {earth.sol.int, rover\_monitor.nasa.gov:23}. Let the address of the PKDN Receiver be: {mars.sol.int, rover1.rovernet.nasa.gov:23}. The PKDN Router for the PKDN Sender could be derived as: {earth.sol.int, pkdn.nasa.gov:2} and the PKDN Router for the PKDN Receiver could be derived as: {mars.sol.int, pkdn.nasa.gov:2}. Thus, if such a topological encoding were available, network service discovery can simply be address-based host discovery. But, when only DTN identifiers (without topological information) are available, only design choices (a) and (b) are feasible. Furthermore, since Option (a) is non-interactive while Option (b) is not, Option (a) may be better suited when only DTN identifiers are available.

### 3.2. Design Choices for Cache Synchronization

It was specified that the Certificate Revocation Manager (CRM) needs to publish Delta-CRLs into the PKDN. It was also specified that PKDN is a network of PKDN Routers (please refer to Figure 2). Thus, the CRM needs to publish its Delta-CRLs to one or more PKDN Routers. The problem is that of synchronization of a distributed cache of CRL information, which is a distributed aggregation problem. A survey of decentralized aggregation protocols has been published by Makhoulfi et. al. [Dagg]. They identify gossip based, tree based, and hybrid aggregation protocols. Although decentralized aggregation is best suited for decentralized DTN, an additional centralized aggregation choice (as hub-and-spoke propagation) is identified as a choice. Note that the PKDN Senders and Receivers are assumed, without loss of generality, to be agnostic of these design choices. The design choices for such a network propagation of Delta-CRLs are as follows.

1. (Hub-and-spoke propagation) Every authorized PKDN Router is registered with the CRM and the CRM (as the hub) periodically sends updates to all registered PKDN Routers (as the spokes) using DTN. The hub-and-spoke propagation is deterministic and simple but the load on the CRM is high and the same information (Delta-CRL) is carried by multiple DTN Bundles along similar DTN paths. In other words, the hub-and-spoke arrangement is not efficient use of the network but simple and deterministic.
2. (Depender graph propagation) This model of propagation is described by Wright et. al. [FTCR]. The basic idea is to let a few first-level PKDN Routers receive Delta-CRLs from CRM, which shall propagate the same to second-level PKDN Routers. The second-level PKDN Routers shall propagate the same to third-level PKDN Routers and so on and so forth. Thus a hierarchy of PKDN Routers shall be organized as an Rooted, Directed Acyclic Graph (ADAG), with the CRM functioning as the root of the graph. The

number DTN bundles with the same information (Delta-CRL) along the same DTN path can be reduced. Additionally, unlike the hub-and-spoke propagation, k-path-redundancy can be realized in the PKDN by requiring every PKDN Router to receive Delta-CRL updates from k sources (one of the k sources for the first-level PKDN Routers must be the CRM.) The disadvantage of this design choice is its design and implementation complexity when compared with the hub-and-spoke design choice.

3. (Gossip propagation) No security literature has been found, as yet, for propagation of CRL information in a dependable manner using gossip protocols. The security property expected out of such gossip-based CRL propagation protocols is only a theoretical feasibility that each Delta-CRL shall eventually reach all PKDN Routers in the PKDN.
4. (Hybrid propagation) Uses a combination of tree-based and gossip-based propagation. No security literature has been found, as yet, for propagation of CRL information in a dependable manner. The security property for such protocols is the same as that stated for the Gossip propagation.

The current recommendation for PKDN Cache Synchronization protocols is either: (a) to develop depender-graph propagation mechanisms; (b) to design and develop gossip-based propagation mechanisms; or, (c) to design and develop hybrid propagation mechanism. The lead-time for developing depender-graph propagation mechanisms may be least among the recommendations. The hub-and-spoke model of propagation is not recommended as it is a special case and not useful for a highly decentralized application of DTN.

### 3.3. Design Choices for Revocation Updates

The design choice for revocation updates is centered around the following questions. Which PKDN Router needs to send updates to a specific PKDN Receiver? The answers to this question provides the following choices:

1. (Updates from distributed PKDN Routers) Every PKDN Router that generated a Validated PKDN Bundle designated for the specified PKDN Receiver. in this case two sub-choices exist as follows: either (a) every PKDN Router sends the entire Delta-CRL to the PKDN Receiver; or (b) each PKDN Router sends authenticated updates only for those certificates that were forwarded to the PKDN Receiver by that PKDN Router. Option (a) generates redundant traffic in the DTN as a PKDN Receiver will receive the same information from multiple PKDN Routers. Therefore, it is recommended that Option (a) be avoided. Option (b) conserves

bandwidth while avoiding single points of failures in PKDN revocation update functionality. But, Option (b) implies increased complexity of design when dealing with PKDN-Receiver crash recovery or de-registering a PKDN Receiver's interest in a given certificate.

2. (Updates from a designated PKDN Router) Every PKDN Receiver registers with a designated PKDN Router for receiving updates or Delta-CRLs. This option requires a one-time idempotent registration from a PKDN Receiver with a PKDN Router during bootstrap. A local copy of CRL need not be saved by the PKDN Receiver. Whenever the PKDN receiver receives a Delta-CRL from the network, it only needs to determine which of the PKDN Sender Certificates in its local database have been revoked due to a Delta-CRL. Crash recovery in this option is naturally available because PKDN Receivers need not store CRLs and no state needs to be stored in the PKDN Routers. To avoid single point of failures in receiving revocation updates, a given PKDN Receiver may subscribe to more than one PKDN Router.

Having a designated PKDN Router for each PKDN Receiver results in a stateless system, which will be scalable. Typically, the design choice for designated PKDN Router is valid when the size of Delta-CRLs are small enough for resource-constrained PKDN Receivers, such as Mars Rovers, to handle. The maximum reported size of CRLs [SizeCRL] on the terrestrial Internet is about 27 Mega Bytes. The size of the Delta-CRLs will be much smaller because the CRLs are partitioned into sub-sets using suitably sized windows of time. In a given 24 hour period, the reported [SizeCRLGrowth] maximum number of certificates issued by VeriSign Inc. [verisign], during a given year, is about 200 certificates or 1% of total number of certified public keys for use on the terrestrial Internet. The Delta-CRL for 200 certificates will be a maximum of few tens of Kilo Bytes. Assuming that the statistics of certificate revocation is going to be similar for DTNs, having a designated PKDN Router for each PKDN Receiver will be a good design choice.

#### 4. Summary of Recommended Design Choices

The following are the recommended design choices for each function of PKDN.

1. (Routing) Since the state-of-art of DTN only includes endpoint identifiers instead of addresses, Option (a) is recommended for designating the PKDN Router between a given PKDN Sender and PKDN Receiver. The PKDN Sender shall route all its PKDN Bundles through its PKDN Router. A (certificate-based) chain of trust must be in place so that the PKDN Receiver can authenticate the

origin of Validated PKDN Bundles. The design for the key management structures for establishing the trust relationship between the sender's PKDN Routers and PKDN Receivers shall be described in a follow-up Internet Draft.

2. (Cache synchronization) The use of depender-graph propagation is recommended because the eventual availability of Delta-CRLs at all PKDN Routers has been proved [FTCR]. If a gossip or hybrid propagation were to be available with similar proof, they will be preferred over depender-graph propagation. This is because gossip and hybrid propagation can allow the existence of an unplanned PKDN while depender-graph propagation requires a planned PKDN.
3. (Revocation updates) Assuming small sized Delta-CRLs, which is evinced [SizeCRLGrowth] in the terrestrial Internet, a designated PKDN Router for every PKDN Receiver is recommended. The PKDN Receiver's PKDN Router shall be designated using the same mechanism as the PKDN Sender's PKDN Router was designated -- Option (a) was recommended above for the Routing function.

## 5. Future work

The feedbacks to this document shall be used to finalize the design of PKDN as a key management protocol suite for DTN in a subsequent Internet Draft. Additionally, the detailed protocol, data structure, and key hierarchy for PKDN shall be described in the subsequent Internet Draft.

## 6. IANA Considerations

This document potentially contains IANA considerations depending on the design choices adopted for future work. But, in its present form, there are no immediate IANA considerations.

## 7. Security Considerations

Security issues and considerations are discussed through out this document.

## 8. References

### 8.1. Normative References

[I-D.birrane-dtn-sbsp]  
Birrane, E., "Streamlined Bundle Security Protocol Specification", draft-birrane-dtn-sbsp-00 (work in progress), December 2014.

- [I-D.templin-dtnskmreq] Templin, F. and S. Burleigh, "DTN Security Key Management - Requirements and Design", draft-templin-dtnskmreq-00 (work in progress), February 2015.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<http://www.rfc-editor.org/info/rfc2119>>.
- [RFC4838] 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, <<http://www.rfc-editor.org/info/rfc4838>>.
- [RFC5050] Scott, K. and S. Burleigh, "Bundle Protocol Specification", RFC 5050, DOI 10.17487/RFC5050, November 2007, <<http://www.rfc-editor.org/info/rfc5050>>.
- [RFC6257] Symington, S., Farrell, S., Weiss, H., and P. Lovell, "Bundle Security Protocol Specification", RFC 6257, DOI 10.17487/RFC6257, May 2011, <<http://www.rfc-editor.org/info/rfc6257>>.

## 8.2. Informative References

- [CAP] Brewer, E., "CAP twelve years later: How the "rules" have changed", Feb 2012, <<http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=6133253>>.
- [DAgg] Makhoulfi, R., Bonnet, G., Doyen, G., and D. Gaiti, "Decentralized Aggregation Protocols in Peer-to-Peer Networks: A Survey", March 2010, <<http://dl.acm.org/citation.cfm?id=1692756>>.
- [EPDTN] Clare, L., Burleigh, S., and K. Scott, "Endpoint naming for space delay / Disruption Tolerant Networking", March 2010, <<http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=5446949>>.
- [FTCR] Rebecca, N., Lincoln, P., and J. Millen, "Efficient Fault-Tolerant Certificate Revocation", Jun 2000, <<http://www.csl.sri.com/papers/dependers/>>.
- [RFC0793] Postel, J., "Transmission Control Protocol", STD 7, RFC 793, DOI 10.17487/RFC0793, September 1981, <<http://www.rfc-editor.org/info/rfc793>>.

- [RFC5246] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5246, DOI 10.17487/RFC5246, August 2008, <<http://www.rfc-editor.org/info/rfc5246>>.
- [RFC5751] Ramsdell, B. and S. Turner, "Secure/Multipurpose Internet Mail Extensions (S/MIME) Version 3.2 Message Specification", RFC 5751, DOI 10.17487/RFC5751, January 2010, <<http://www.rfc-editor.org/info/rfc5751>>.
- [RFC6066] Eastlake 3rd, D., "Transport Layer Security (TLS) Extensions: Extension Definitions", RFC 6066, DOI 10.17487/RFC6066, January 2011, <<http://www.rfc-editor.org/info/rfc6066>>.
- [RFC6960] Santesson, S., Myers, M., Ankney, R., Malpani, A., Galperin, S., and C. Adams, "X.509 Internet Public Key Infrastructure Online Certificate Status Protocol - OCSP", RFC 6960, DOI 10.17487/RFC6960, June 2013, <<http://www.rfc-editor.org/info/rfc6960>>.
- [SizeCRL] Raytheon Websense, "Digging Into Certificate Revocation Lists", July 2013, <<http://community.websense.com/blogs/securitylabs/archive/2013/07/11/digging-into-certificate-revocation-lists.aspx>>.
- [SizeCRLGrowth] Walleck, D., Li, Y., and S. Xu, "Empirical Analysis of Certificate Revocation Lists", 2008, <[http://rd.springer.com/chapter/10.1007%2F978-3-540-70567-3\\_13](http://rd.springer.com/chapter/10.1007%2F978-3-540-70567-3_13)>.
- [verisign] Wikipedia Inc, "Wikipedia entry for Verisign Inc", August 2015, <<https://en.wikipedia.org/wiki/Verisign>>.

#### Authors' Addresses

Kapali Viswanathan  
Boeing Research & Technology  
Unit 501, 5th Floor, Tower D, RMZ Infinity  
No 3, Old Madras Rd  
Bangalore, KA 560016  
IN

Email: [kapaleeswaran.viswanathan@boeing.com](mailto:kapaleeswaran.viswanathan@boeing.com)

Fred L. Templin  
Boeing Research & Technology  
P.O. Box 3707  
Seattle, WA 98124  
USA

Email: [fltemplin@acm.org](mailto:fltemplin@acm.org)