

Delay-Tolerant Networking
Internet-Draft
Intended status: Informational
Expires: January 1, 2019

E. Birrane
E. DiPietro
D. Linko
Johns Hopkins Applied Physics Laboratory
June 30, 2018

AMA Application Data Model
draft-birrane-dtn-adm-03

Abstract

This document defines a physical data model that captures the information necessary to asynchronously manage applications. This model provides a set of common type definitions, data structures, and a template for publishing standardized representations of model elements.

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 January 1, 2019.

Copyright Notice

Copyright (c) 2018 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
1.1. Scope	3
2. Requirements Language	4
3. Terminology	4
4. Data Modeling Concept of Operations	5
5. Asynchronous Management Model (AMM)	6
5.1. The AMM Resource Identifier (ARI)	6
5.1.1. Namespaces	7
5.1.2. Object Names	9
5.1.3. Parameters	9
5.1.4. Special Case: Literal Values	10
5.1.5. String Canonical Forms	11
5.1.6. Examples	12
5.2. AMM Type Definitions	15
5.2.1. Primitive Types	15
5.2.2. Derived Types	15
5.2.3. Collections	17
5.3. Object Definitions	18
5.3.1. Common Object Metadata	18
5.3.2. Externally Defined Data (EDD)	19
5.3.3. Constant (CONST)	19
5.3.4. Control (CTRL)	20
5.3.5. Macro (MAC)	21
5.3.6. Operator (OP)	21
5.3.7. Reports	23
5.3.8. State-Based Rule (SBR)	24
5.3.9. Tables	25
5.3.10. Time-Based Rule (TBR)	27
5.3.11. Variable (VAR)	28
5.3.12. Common Object Processing	29
5.4. Data Type Mnemonics and Enumerations	30
5.4.1. AMM Objects	30
5.4.2. Primitive Data Types	31
5.4.3. Compound Data Types	32
5.4.4. Numeric Promotions	33
5.4.5. Numeric Conversions	33
6. JSON ADM Template	33
6.1. ADM Inclusion	34
6.2. ADMT Object Collections	34
6.3. ADM Metadata	35
6.4. Type Encodings	36
6.4.1. Primitive Type Encoding	36
6.4.2. Derived Type Encoding	36

6.4.3.	Collection Encoding	37
6.5.	ARI Encoding	38
6.6.	ADM Structures	40
6.6.1.	General Notes	40
6.6.2.	Constant (CONST) Encoding	41
6.6.3.	Control (CTRL) Encoding	41
6.6.4.	Externally Defined Data (EDD) Encoding	42
6.6.5.	Macro Encoding	43
6.6.6.	Operator (OP) Encoding	43
6.6.7.	Table Template (TBLT) Encoding	44
6.6.8.	Report Template Encoding	44
6.6.9.	Variables Encoding	46
6.6.10.	Exemptions	47
7.	ADM Author Considerations	47
8.	IANA Considerations	49
9.	Security Considerations	49
10.	References	49
10.1.	Normative References	49
10.2.	Informative References	49
	Authors' Addresses	50

1. Introduction

The Asynchronous Management Architecture (AMA) [I-D.birrane-dtn-ama] defines a concept for the open-loop control of applications (and protocols) in situations where timely, highly-available connections cannot exist amongst managing and managed nodes in a network. While the AMA provides a logical data model, it does not include the detailed information necessary to produce interoperable data models.

1.1. Scope

This document defines a physical data model suitable for managing applications in accordance with the AMA. This physical model is termed the Asynchronous Management Model (AMM) and consists of the data types and data structures needed to manage applications in asynchronous networks.

This document also provides a template, called the Application Data Model Template (ADMT), for the standardized representation of application-specific instances of this model. Using the types and structures defined by the AMM, individual applications can capture their unique, static management information in documents compliant with the ADMT. These application-specific documents are called Application Data Models (ADMs).

The AMM presented in this document does not assume any specific type of application or underlying network encoding. In order to

communicate model elements between AMA Agents and Managers in a network, the model must be encoded for transmission. Any such encoding scheme is outside of the scope of this document. Generally, the encoding of the model is a separate concern from the specification of data within the model.

Because different networks may use different encodings for data, mandating an encoding format would require incompatible networks to encapsulate data in ways that could introduce inefficiency and obfuscation. It is envisioned that different networks would be able to encode ADMs in their native encodings such that the translation of ADM data from one encoding to another can be completed using mechanical action taken at network borders.

Since the specification does not mandate an encoding format, the AMM and ADMT must provide enough information to make encoding (and translating from one encoding to another) an unambiguous process. Therefore, where necessary, this document provides identification, enumeration and other schemes that ensure ADMs contain enough information to prevent ambiguities caused by different encoding schemes.

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. Terminology

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

Additional terms defined in this document are as follows.

- o Application - A software implementation running on an Agent and being managed by a Manager. This includes software that implements protocol processing on an Agent.
- o Application Data Model (ADM) - The set of statically-defined data items necessary to manage an application asynchronously.
- o Application Data Model Template (ADMT) - A standard format for expressing predefined data items for an application.

- o AMM Resource Identifier (ARI) - A unique identifier for any AMM object, syntactically conformant to the Uniform Resource Identifier (URI) syntax documented in [RFC3986] and using the scheme name "ari".
- o ADM Namespace - A moderated, hierarchical taxonomy of namespaces that describe a set of ADM scopes. Specifically, an individual ADM namespace is a specific sequence of ADM namespaces, from most general to most specific, that uniquely and unambiguously identify the namespace of a particular ADM.
- o Operational Data Model (ODM) - The operational configuration of an Agent. This includes the union of all ADM information supported by the Agent as well as all operational, dynamic configuration applied to the Agent by Managers in the network.

4. Data Modeling Concept of Operations

In order to asynchronously manage an application in accordance with the [I-D.birrane-dtn-ama], an application-specific data model must be created containing any predefined management information for that application. This model is termed the Application Data Model (ADM) and forms the core set of information for that application in whichever network it is deployed. The ADM syntactically conforms to the ADMT and uses the data structures and types that comprise the AMM.

The information standardized in the ADM represents static configurations and definitions that apply to any deployment of the application, regardless of the network in which it is operating. Within any given network, Managers supplement the information provided by ADMs with dynamic definitions and values. The operational configuration of the network is the union of all supported ADMs and all Manager-defined dynamic configurations. This is termed the Operational Data Model (ODM).

The relationships amongst the AMM, ADMT, and ADM are illustrated in Figure 1.

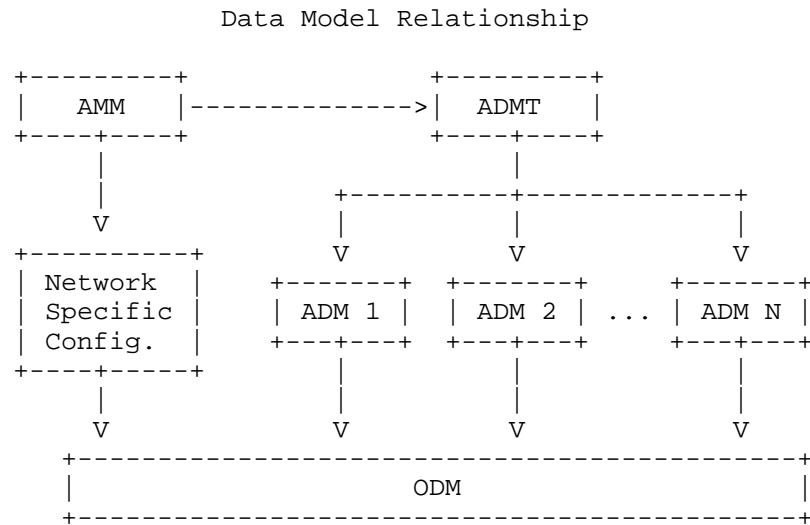


Figure 1

In this figure, AMM data types and structures form the common elements of the management model used by both ADMs and network specific configurations. Together, the set of static information provided by the union of all supported ADMs with the set of operator-specified dynamic AMM objects, forms the operational data model used to manage the network.

5. Asynchronous Management Model (AMM)

This section describes the Asynchronous Management Model, which is the set of objects used to implement the logical data model provided by the AMA. This section also provides additional information necessary to work with this model, such as data type specifications, identifier constructs, and naming conventions.

5.1. The AMM Resource Identifier (ARI)

Every object in the AMM must be uniquely identifiable, regardless of whether the item is defined formally in an ADM document or informally by operators in the context of a specific network deployment. The AMM Resource Identifier (ARI) uniquely identifies AMM objects.

There are three components to the ARI: namespaces, object names, and parameters. This section defines each of these components, discusses special cases, and presents a string canonicalization of these identifiers, with examples.

5.1.1.1. Namespaces

AMM objects exist within unique namespaces to prevent conflicting names within network deployments, particularly in cases where network operators are allowed to define their own object names. In this capacity, namespaces exists to eliminate the chance of a conflicting object name. They MUST NOT be used as a security mechanism. An Agent or Manager MUST NOT infer security information or access control based solely on namespace information.

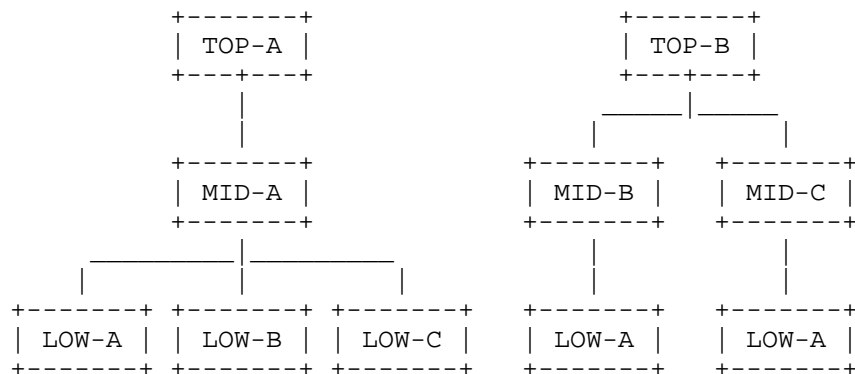
The AMM defines three ways to identify namespaces for AMM object names: Moderated Namespaces, Anonymous Namespaces, and Issuer Namespaces.

5.1.1.1.1. Moderated Namespaces

The most effective way to ensure the uniqueness of an AMM Object is to name it in the context of a moderated namespace. These namespaces are assigned by an overseeing organization as part of a maintained namespace registry.

Moderated namespaces are hierarchical, which allows the grouping of objects that share common attributes - for example, objects associated with related protocols may have protocol-specific namespaces that are grouped under a single encompassing namespace. Namespaces that are closer to a root node in the moderated hierarchy have broader scope than namespaces closer to leaf nodes in that hierarchy. There is no requirement that the namespace hierarchy be represented as a single tree structure; multiple root nodes are acceptable and likely to exist.

In a hierarchical model of namespaces, a particular namespace can be identified as the path to that namespace through the hierarchy. The expression of that path within an ADM is accomplished by listing each namespace along the path, separated by the tokenizing character "/". For example, consider the namespaces in the following figure.



Given this hierarchy, the following are all valid namespace representations.

TOP-A/

TOP-A/MID-A

TOP-A/MID-A/LOW-A

TOP-B/MID-B/LOW-A

TOP-B/MID-C/LOW-A

Moderated namespaces require resources to review and publish and are best suited for static AMM object definitions, such as those found in ADMs.

5.1.1.2. Anonymous Namespaces

It is possible for network operators to define AMM objects that are not associated with a namespace. In this case, a nil namespace can be defined. This special case is considered the use of an "anonymous" namespace.

Policy decisions as to whether anonymous namespaces are allowed in the system should be determined before network deployment. The use of an anonymous namespace greatly increases the chances of naming collisions.

5.1.1.3. Informal Namespaces

Network-specific configurations, as illustrated in Figure 1, are dynamic, ephemeral, not captured in published ADMs, and do not use moderated namespaces. Instead, AMM objects that comprise network-

specific configuration can be uniquely differentiated as a function of their "Issuer" and an issuer-specific "Tag".

An Issuer is any string that identifies the organization that is defining an AMM object. This value may come from a global registry of organizations, an issuing node address, a signed known value, or some other network-unique marking. Issuers MUST NOT conflict with known moderated namespaces, and Agents and Managers should not process Issuers that conflict with existing moderated namespaces.

A Tag is any string used to disambiguate AMM Objects for an Issuer. The contents of the tag are left to the discretion of the Issuer. Examples of potential tag values include an issuer-known version number or a (signed) hashing of the data item associated with the reference identifier.

5.1.2. Object Names

Object names are strings whose value is determined by the creator of the object. For those objects defined in accordance with the ADMT Template, the structure of the object name is given in Section 5.3.1.

5.1.3. Parameters

Parameterization is used in the AMM to enable expressive autonomous function and reduce the amount of traffic communicated between Managers and Agents. In the AMM, most objects can be parameterized and the meaning of parameterization for each object is described in detail in Section 5.3.

If there are two instances of an AMM object that have the same namespace and same object name but have different parameters, then those instances are both unique and the ARIs for those instances MUST also be unique. Therefore, parameters are considered part of an AMM object's identifier.

There are two types of parameters defined in the AMM: Formal and Actual parameters. The terms formal parameter and actual parameter follow common computer programming vernacular for discussing function declarations and function calls, respectively.

5.1.3.1. Formal Parameters

Formal parameters define the type, name, and order of the information that customizes an AMM Object. They represent the unchanging "definition" of the parameterized object.

Formal parameters MUST include type and name information and MAY include an optional default value. If specified, a default value will be used whenever a set of actual parameters fails to provide a value for this formal parameter.

5.1.3.2. Actual Parameters

Actual parameters represent the data values passed to a parameterized AMM Object. They "fulfill" the parameter requirements defined by the formal parameters for that object.

An actual parameter MUST specify a value and MAY specify a type. If a type is provided it MUST match the type provided by the formal parameter. An actual parameter MUST NOT include NAME information.

There are two ways in which the value of an actual parameter can be specified: parameter-by-value and parameter-by-name.

Parameter-By-Value

This method involves directly supplying the value as part of the actual parameter. It is the default method for supplying values.

Parameter-By-Name

This method involves specifying the name of some other parameter and using that other parameter's value for the value of this parameter. This method is useful when a parameterized AMM Object contains another parameterized AMM Object. The contained object's actual parameter can be given as the name of the containing object's parameter. In that way, a containing object's parameters can be "pass down" to all of the objects it contains.

5.1.3.3. Optional Parameters

In cases where a formal parameter contains a default value, the associated actual parameter may be omitted. Default values in formal parameters (and, thus, optional actual parameters) are encouraged as they reduce the size of data items communicated amongst Managers and Agents in a network.

5.1.4. Special Case: Literal Values

As defined in the AMA, Literal values are those whose value and identifier are equivalent. For example, the literal "4" serves as both an identifier and a value. When literal values are used in objects in the AMM, they are able to have a simplified identification scheme.

Because the value of a Literal object serves as its identifier, there is no need for namespaces, object names, or parameters. A literal can be completely identified by its data type and data value. Since Literals in the AMA are used to identify primitive data types, the type of a Literal identifier MUST be as described in Table 2.

5.1.5. String Canonical Forms

While there may exist multiple encodings of an ARI, to include the JSON encodings presented in Section 6 and other binary encodings in other specifications, this section defines a universal string representation of the ARI, as such a representation is helpful to express examples in this and other documents.

This representation is not prescriptive; other string encodings may exist that differ from the one used in this document.

5.1.5.1. General ARI String Representation

The String Canonical Form of the ARI is expressed as a Uniform Resource Identifier (URI), as documented in [RFC3986]. A URI is syntactically decomposed into a scheme name and a scheme-specific part. The set of known scheme names is moderated by IANA. The scheme-specific part of the URI is dependent on the scheme name.

The scheme name of the ARI is "ari". The scheme-specific part of the "ari" scheme follows the format:

ari:/<Namespace>/<ObjectName><(Parameters)>

With the string representation of each scheme given as follows.

Namespaces

Namespaces are represented as "/" separated lists, with individual namespace types represented as follows:

- * Moderated namespaces are listed in order from the most general namespace to the most specific namespace. For example: "GENERAL/MIDDLE/SPECIFIC/".
- * Anonymous namespaces are empty and are represented as "/".
- * Informal namespaces follow the general pattern of moderated namespaces - starting with the general Issuer followed by the more specific issuer tag. For example: "Issuer/Tag". In cases where the Tag is omitted, then the representation is simply "Issuer/".

Object Names

The object name is a string as specified in Section 5.3.1.

Parameters

If present, parameters are represented as a comma-separated string enclosed in parenthesis. Different types of parameters are represented as follows.

- * Formal parameters follow the pattern <type> <name> and if there is a default value, it is represented by the substring "= <value>".
- * Actual Parameters-By-Value are represented as the string encoding of their value.
- * Actual Parameters-By-Name are represented as the name of the parameter enclosed in angle brackets.

Note: If an actual parameter is missing for a formal parameter that has a default value, then the ARI string MUST have a blank space where the actual parameter would have been. This missing parameter will also have a comma, separating it from other actual parameters in the ARI string.

5.1.5.1.1. Shortform Encoding

In cases where a default namespace can be assumed (for example, in the context of an ADM with a defined namespace) the prefix `ari:/Namespace/` can be omitted.

5.1.5.2. Literal String Encoding

The string representation of a Literal ARI is much simpler and consists of simply the data type of the Literal followed by the value, as follows:

"(type) value"

5.1.6. Examples

The ARIs for the following sample AMM objects are encoded in Table 1. Note that these examples are for the identifiers of AMM objects, not their entire definition.

- o The number of bytes received by an Agent, defined in the N1/N2 namespace and called `num_bytes`.

- o The number of bytes received through an interface, called `num_bytes_if`, which takes a single string parameter named `"if_name"` with a default value of `"eth0"`.
- o An anonymous, operator-defined object named `"obj1"` which takes two unsigned integer parameters, `n1` and `n2`, with default values of 3 and 4, respectively.
- o The typed, Literal value of 4.

ARI String	Description
"ari:N1/N2/num_bytes"	Unparameterized num_bytes object in the N1/N2 namespace.
"num_bytes"	Shortform encoding where the N1/N2 namespace can be assumed.
"num_bytes_if(String if_name)"	Formal parameter definition of num_bytes object that accepts a string interface name.
"num_bytes_if(String if_name=eth0)"	Formal parameter definition of num_bytes object that accepts a string interface name with a default value.
"num_bytes_if()"	Actual parameter using the default value of eth0.
"num_bytes_if(eth0)"	Actual parameter of eth0.
"ari:/obj1(Int n1 = 0, Int n2 = 3)"	Formal parameter of object obj1 in anonymous namespace taking 2 default parameters.
"ari:/obj1(,)"	Actual parameter using the default values of 0 for n1 and 3 for n2.
"ari:/obj1(, 4)"	Actual parameter using the default value of 0 for n1.
"ari:/obj1(4,)"	Actual parameter using the default value of 3 for n2.
"ari:/obj1(4,4)"	Actual parameters provided for all obj1 parameters.
"ari:/obj1(<input>,4)"	Actual parameters provided for all obj1 parameters, with the value of the first parameter taken from some other parameter named "input".
"(UINT) 4"	The Literal value 4 interpreted as a 32-bit unsigned integer.

Table 1

5.2. AMM Type Definitions

This section describes the type definitions used by the AMM.

5.2.1. Primitive Types

The AMM supports a series of primitive types as outlined in Table 2.

Type	Description
BYTE	unsigned byte value
INT	32-bit signed integer in 2's complement
UINT	32-bit unsigned integer in 2's complement
VAST	64-bit signed integer in 2's complement
UVAST	64-bit unsigned integer in 2's complement
REAL32	Single-precision, 32-bit floating point value in IEEE-754 format.
REAL64	Double-precision, 64-bit floating point value in IEEE-754 format.
STRING	NULL-terminated series of characters in UTF-8 format.
BOOL	A Boolean value of FALSE (whose integer interpretation is 0) or TRUE (whose integer interpretation is not 0).

Table 2: Primitive Types

5.2.2. Derived Types

A derived type is a primitive type that is interpreted with special semantics. The AMM supports the following derived types.

5.2.2.1. Byte String

A Byte String is a specialization of the String primitive data type used to store binary data using base64 encoding as defined in [RFC4648].

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

A Time Value (TV) is a specialization of the String primitive data type whose time interpretation is as given in this section. There are two "types" of time representations within the AMM: relative times and absolute times.

An absolute time represents an instant in time. It MUST be formatted as a date-time in accordance with [RFC3339].

A relative time is defined as the amount of time after an instant in time. A relative time MUST be formatted as a full-time in accordance with [RFC3339]. Relative times have advantages over absolute times: they do not require time to be synchronized across Agents and Managers, and they are more compact in their representation. For example, expressing the semantics "run control_one 10 seconds after receiving it" or "run control_two 20 seconds after running control_one" is more appropriate using relative times than absolute times. The initiating event of a relative time MUST be unambiguously defined in the context using the time value.

As a practical matter, encodings of relative times MAY impose a limit of no more than 17 years of relative time, which corresponds to roughly 29 bits of information and is considered well past an upper bound of efficiency for using a relative time versus an absolute time.

An absolute time may be differentiated from a relative time based on whether the time specification is a date-time or a full-time.

For example, "00:00:10Z" is a relative time representing 10 seconds after an initiating event. "2019-01-01T08:00:00Z" is an absolute time that refers to 8am, Tuesday January 1st, 2019.

A Timestamp (TS) represents a specific point in time when an event occurred. As such, it MUST be represented as an absolute time.

5.2.2.3. Type-Name-Value (TNV)

A Type-Name-Value (TNV) is a three-tuple of information that describes a typed, named value in the AMM. Since the length of a data value is a matter of encoding, there is not an explicit length field present for the data value; it is assumed that any encoding scheme either explicitly encodes length or that the length is self-delineating in the encoding.

- o Type - The strong typing for this value. Types MUST be one of those defined in Section 5.4.

- o Name - A unique identifier for this value.
- o Value - The value of the data item.

5.2.2.4. User-Specified Derived Types

Individual ADMs and network operators may derive other types that specialize the types provided by the AMM. When doing so, AMM data types MUST be used to capture the specialization and any user-specific verification or validation MUST occur in user-specific implementations on Agents and Managers.

5.2.3. Collections

AMM objects, or parameters associated with those objects, often need to represent groups of related information. Since the AMM is strongly typed, these groups of related information are represented by special data types called collections. AMM collections are ordered and may contain duplicate entries.

The AMM defines three typed collections that capture TNVs, ARIs, and mathematical expressions.

5.2.3.1. Type-Name-Value Collection (TNVC)

A Type-Name-Value Collection (TNVC) is an ordered array where each element of the array is a TNV.

TNVCs are often used to capture formal and actual parameters for AMM objects.

5.2.3.2. ARI Collection (AC)

An ARI Collection (AC) is an ordered set of ARIs.

ACs are often used when there exists a need to refer to multiple AMM objects as a single unit. For example, when defining a Report Template, the definition may have an AC that defines the ordered ARIs whose values constitute that report.

5.2.3.3. Expression (EXPR)

An Expression (EXPR) is a specialization of an AC where each ARI in the collection is either an operand or an operator. These operands and operators form a mathematical expression that is used to compute a numerical value.

Within an Expression, an operand MUST be an ARI with one of the following types: Literal, Constant, Externally Defined Data, or Variable. An operator MUST be an ARI of type Operator.

Since the Expression is an AC, there are no annotative constructs such as parenthesis to enforce certain orders of operation. To preserve an unambiguous calculation of values, the ARIs that form an Expression MUST be represented in postfix order. Postfix notation requires no additional symbols to enforce precedence, always results in a more efficient encoding, and post-fix engines can be implemented efficiently in embedded systems.

For example, the infix expression $A * (B * C)$ is represented as the postfix $A B C * *$.

Expressions are often used when assigning values to a Variable or when calculating the state of the Agent in the context of a State-Based Rule.

5.3. Object Definitions

This section identifies the AMM Objects that instantiate the AMA logical data model and the processing required to support these objects at Agents and Managers in the network.

5.3.1. Common Object Metadata

Every object in the AMM includes a set of metadata providing annotative or otherwise user-friendly descriptive information for the object. This information may be used as documentation (for example, only present in ADMs and on operator consoles) and/or encoded and transmitted over the wire as part of a management protocol.

Metadata is not required to be unique amongst objects and individual encodings MAY choose to not encode metadata in cases where the information is not needed to uniquely identify objects. The metadata supported by the AMM for objects is as follows:

(STR) Name

An object name is a string associated with the object, but does not constitute the sole identifier for the object. Names provide human-readable and/or user-friendly ways to refer to objects within a given context.

(STR) Description

An object description is a string describing the purpose or usage of the object in a human-readable format. The description serves as documentation for the object and SHOULD

be the same regardless of how the object might be parameterized. For example, the description of a CTRL object should document the purpose of the CTRL in a way that is independent of the value of any particular parameter value passed to that CTRL.

5.3.2. Externally Defined Data (EDD)

Externally defined data (EDD), as defined in the AMA, represent data values that are computed external to the network management system. The definition of these values are solely defined in the context of an ADM; since their calculation exists outside of the network management system, they are not added or removed as part of the dynamic configuration of the network management system.

An EDD consists of an ARI, type, and a description, with the following caveats:

(ARI) Identifier

This Identifier **MUST** be of type EDD and **MAY** be parameterized, particularly in cases where the specific computed value can be identified by an associative look-up, as discussed in Section 7.

(UINT) Type

The data type of the EDD value **MUST** be specified as part of the EDD definition and this type **MUST** be one of the primitive data types defined in Table 2.

(STR) Description

This represents the human-readable description of the EDD.

5.3.3. Constant (CONST)

Constants represent named basic values. Examples include common mathematical values such as PI or well-known Epochs such as the UNIX Epoch. Constants are defined solely within the context of ADMs. Constants **MUST NOT** be defined as part of dynamic network configuration.

Allowing network operators to define constants dynamically means that a Constant could be defined, removed, and then re-defined at a later time with a different value, which defeats the purpose of having Constants. Variables **MUST** be used instead of Constants for the purpose of adding new values to the dynamic network configuration.

A CONST is defined by its ARI, value, and description, with the following caveats.

(ARI) Identifier

This Identifier MUST be of type CONST and MUST NOT be parameterized. Parameterizing a Constant implies that its value is dependent upon the set of parameters sent to it, which defeats the purpose of defining a constant value.

(TNV) Typed Value

The value of a constant is the immutable value that should be used in lieu of the Constant ARI.

This value is expressed as a TNV with the following requirements.

- * Type MUST be one of the primitive data types defined in Table 2.
- * Name MUST be omitted as the CONST ARI defines the name for this value.
- * Value must be present and consistent with the data type for this CONST.

(STR) Description

This represents the human-readable description of the CONST, as a string.

5.3.4. Control (CTRL)

A Control represents a predefined function that can be run on an Agent. Controls are not able to be defined as part of dynamic network configuration since their execution is typically part of the firmware or other implementation of the Agent outside of the context of the network management system.

Network operators that wish to dynamically execute functions on an Agent may use Macros, State-Based Rule, and Time-Based Rule instead.

Controls are identified by their ARI and their description, with the following caveats.

(ARI) Identifier

This Identifier MUST be of type CTRL and MAY be parameterized in cases where the function executed by that Control takes parameters.

When defined in the context of an ADM, the Control ARI MUST match the definition of a Formal Parameter list. This is because the ADM defines the Controls that can be invoked, but does not define any particular invocation of a Control.

When used as part of network operations, a Control ARI MUST match the definition of an Actual Parameter list. This is because when used operationally, a parameterized Control required parameters to be run. In cases where a Control takes no parameters, the definition in the ADM document MUST be considered the definition of the Control and the presence of the same ARI in the context of an operational system MUST be seen as an invocation of that Control.

(STR) Description

This represents the human-readable description of the Control, as a string.

5.3.5. Macro (MAC)

Macros are ordered collections of Controls or other Macros. When run by an Agent, each ARI in the AC is run in order. A Macro may be defined as part of an ADM or as part of dynamic network configuration.

In cases where a Macro contains another Macro, implementations MUST implement some mechanism for preventing infinite recursions, such as defining maximum nesting levels, performing Macro inspection, and/or enforcing maximum execution times.

A Macro is defined by an ARI, a content definition, and a description, as follows.

(ARI) Identifier

This Identifier MUST be of type MAC and MAY be parameterized and, if so, the parameter may be passed-by-name to any parameterized elements within the Macro.

(AC) Definition

The Macro definition is modeled as an AC, where each ARI within the AC MUST be either a Control or a Macro.

(STR) Description

This represents the human-readable description of the Macro, as a string.

5.3.6. Operator (OP)

Operators represent user-defined mathematical functions implemented in the firmware of an Agent for the purpose of aiding the evaluation of Expressions.

The AMM separates the concepts of Operators and Controls to prevent side-effects in Expression evaluation (e.g. to avoid constructs such as `A = B + GenerateReport()`). For this reason, Operators are given their own structure type and Controls do not support a return value.

Because Operators represent custom firmware implemented on the Agent, they are not defined dynamically as part of network operations. Therefore, they may only be defined in an ADM.

An Operator is defined by its ARI, its resultant type, the number of operands, the type of operands, and a description, as follows.

(ARI) Identifier

This Identifier MUST be of type OP and MUST NOT be parameterized. Much like Constants, Operators represent immutable mathematical functions. The operands of an Operator are not considered as "parameters" to the Operator.

(UINT) Out Type

This is the return value of the Operator and MAY be different than the operand types accepted by the Operator. This type MUST be one of the primitive data types defined in Table 2.

(UINT) Num Operands

This is the number of operands evaluated by the operator. For example, the unary NOT Operator ("!") would operate on a single operand. The binary PLUS Operator ("+") would operate on two operands. A custom operator to calculate the average of the last 10 samples of data would operate on 10 operands.

(TNVC) In Types

This is the type information for each operand operated on by the Operator, modeled as a TNV Collection (TNVC). There MUST be one TNV in the TNVC for each operand, and each TNV MUST adhere to the following requirements:

- * The Type field MUST be present and MUST be one of the primitive data types defined in Table 2.
- * The Name field MAY be present to capture a semantic name for the operand.
- * The Value field MUST NOT be present.

(STR) Description

This represents the human-readable description of the Operator, as a string.

5.3.7. Reports

A Report is a set of non-tabular, potentially nested data items that may be predefined in the context of an ADM, or defined dynamically in the context of a deployed network.

Reports are represented in two ways in the AMM: Report Templates and Reports. A Report Template defines the type of information to be included in a report, and a Report contains that information.

5.3.7.1. Report Template (RPTT)

A Report Template (RPTT) is the ordered set of data descriptions that describe how values will be represented in a corresponding Report. RPTTs can be viewed as a schema that describes how to interpret a Report; they contain no values and are either defined in an ADM or configured between Managers and Agents during network operations.

Since a RPTT may contain other RPTTs, implementations **MUST** implement some mechanism to prevent the definition of circular references.

RPTTs are defined by an ARI, the report template definition, and a description, as follows.

(ARI) Identifier

This Identifier **MUST** be of type RPTT and **MAY** be parameterized and, if so, the parameter may be passed-by-name to any parameterized elements within the RPTT.

(AC) Definition

The Report Definition is modeled as an AC, where each ARI within the AC **MUST** identify either a CONST, LIT, EDD, VAR, or other RPTT.

(STR) Description

This represents the human-readable description of the Report template, as a string.

5.3.7.2. Report (RPT)

A Report (RPT) is a set of data values populated in conformance to a given data definition. Reports do not have an individual identifier - rather they are uniquely identified by their definition and the timestamp at which their data values were collected.

RPTs are defined by their associated template, the time at which the report was generated, and the individual entries in the report, as follows.

(ARI) Template Id

This is the ARI of the object that defines the format of the report data values. This ARI MUST define an AMM object of type RPTT, EDD, or VAR, or CTRL.
If this ARI is parameterized, this ARI MUST include the actual parameters used in the generation of the report.

(TV) Generation Time

This is the absolute time at which the report was generated by the Agent.

(TNVC) Report Entries

This is the collection of data values that comprise the report. If the template for this report is an EDD, VAR, or CTRL then there MUST be one entry for this report. If the template is a RPTT, then there MUST be one entry for every item defined in the template.
Entries are modeled as a TNVC, with each TNV representing a report entry with the following requirements.

- * Type MAY be omitted in cases where checking type safety is not required.
- * Name MAY be omitted in cases where a semantic name for the entry can be derived from the template.
- * Value MUST be present and consistent with the type for this entry from the associated template item.

5.3.8. State-Based Rule (SBR)

A State-Based Rule (SBR) specifies that starting at a particular time an action should be run by the Agent if some condition evaluates to true, until the action has been run a maximum number of 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 Variable V1 > 10, produce a Report Entry for Report Template R1 no more than 20 times.

Starting at some future absolute time, whenever Variable V2 != Variable V4, run Macro M1 no more than 36 times.

SBRs are defined by their ARI, start time, condition, maximum run count, action, and description, as follows.

(ARI) Identifier

This Identifier MUST be of type SBR and MUST NOT be parameterized.

(TV) 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.

(EXPR) CONDITION

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

(UFAST) COUNT

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.

(AC) ACTION

The collection of Controls and/or Macros to run as part of the action. This is captured as an AC data type with the constraint that every ARI within the AC represent a Control or Macro.

(STR) Description

This represents the human-readable description of the SBR, as a string.

5.3.9. Tables

A Table is a named, typed, collection of tabular data. Columns within a table are named and typed. Rows within a table capture individual data sets with one value in each row corresponding to one column in the table. Tables are represented in two ways in the AMM: Table Templates and Table Instances.

5.3.9.1. Table Template (TBLT)

A table template identifies the strongly-typed column template that will be followed by any instance of this table available in the network. Table templates appear statically in ADMs and may not be created dynamically within the network by Managers. Changing a table template within an asynchronously managed network would result in confusion if differing template definitions for the same table identifier were to be active in the network at one time.

TBLTs are defined by an ARI, a set of column descriptions, and table metadata, as follows.

(ARI) Identifier

This Identifier MUST be of type TBLT and MUST be of type TBLT, and MUST NOT contain parameters.

(TNVC) Columns

A TBLT is defined by its ordered set of columns descriptions captured as a TNVC with each TNV in the collection describing a table column with the following requirements.

- * Type MUST be present and MUST be one of the primitive types defined in Table 2.
- * Name MAY be omitted in cases where a semantic name for the column can be derived from the template.
- * Value MUST NOT be present as a column does not contain data values.

(STR) Description

This represents the human-readable description of the TBLT, as a string.

5.3.9.2. Table (TBL)

Tables are collections of data that MUST be constructed in accordance with an associated Table Template. Tables MUST NOT appear in the ADM for an application; they are only instantiated dynamically as part of the operation of a network.

TBLs are defined by their Table Template, the number of rows in the table, and the associated set of data values for each row.

(ARI) Template Id

This is the ARI of the Table Template holding the column definitions for this table. This ARI MUST be of type TBLT and match a known Table Template.

(UINT) Number of Rows

This is the number of rows in the table. A Table MAY have zero rows.

(TNVC) Rows Information

Each row in a TBL is represented by a TNVC, with each TNV in the collection representing the value for a specific column with the following requirements.

- * Type MAY be present, when necessary to verify that elements in the row match the types of table columns.

- * Name MUST NOT be present.
- * Value MUST be present and compatible with the type of the associated column.
The number of TNVs in the collection MUST be equal to the number of columns defined for the Table Template.

5.3.10. Time-Based Rule (TBR)

A Time-Based Rule (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.

TBRs are defined by their ARI, start time, period, maximum run count, action, and description, as follows.

(ARI) Identifier

This Identifier MUST be of type TBR and MUST NOT be parameterized.

(TV) Start

The time at which the TBR should start to be evaluated. This will mark the first running of the action associated with the TBR.

(TV) Period

The time to wait between running the action associated with the TBR. This value MUST be a relative time value.

(UVAST) Count

The number of times the TBR action may be run. The special value of 0 indicates the TBR should continue running the action indefinitely.

(AC) Action

The collection of Controls and/or Macros to run by the TBR. This is captured as a AC with the constraint that every ARI within the AC represent a Control or Macro.

(STR) Description

This represents the human-readable description of the TBR, as a string.

5.3.11. Variable (VAR)

Variables (VAR) may be statically defined in an ADM or dynamically by Managers in a network deployment. VARs differ from EDDs in that they are completely described by other known data in the system (either other VARs or other EDDs). For example, letting E# be a EDD item and V# be a VAR item, the following are examples of VAR definitions.

V1 = E1 * E2

V2 = V1 + E3

VARs are defined by an ARI, a type, an initializing expression, and a description, as follows.

(ARI) Identifier

The type of this ARI MUST be type VAR, and the ARI MUST NOT contain parameters.

(UINT) Type

This is the type of the VAR, and acts as a static cast for the result of the initializing EXPR. This type MUST be one of the data types defined in Table 2.

Note, it is possible to specify a type different than the resultant type of the initializing EXPR. For example, if an EXPR adds two single-precision floating point numbers, the VAR MAY have an integer type associated with it.

(EXPR) Initializer

The initial value of the VAR is given by an initializing EXPR. In the case where the type of the VAR itself is EXPR, the initializer is used as the value of the VAR. In the case where the type of the VAR is anything other than EXPR, then the initializer EXPR will be evaluated and the result of that evaluation will be the initial value of the VAR.

(STR) Description

This represents the human-readable description of the VAR, as a string.

5.3.12. Common Object Processing

This section describes the handling and exchange of AMM objects between Agents and Managers in a network.

Managers must:

- o Store the ARI and definitions for both network-specific and ADM-defined AMM Objects.
- o Send requests to Agents to add, list, describe, and remove custom AMM object definitions.
- o Verify and interpret reports against report templates and tables against table templates when receiving these objects from an Agent.
- o Encode ARIs in Objects to Agents, and decode ARIs from Agents.
- o Provide actual parameters when sending parameterized objects to an Agent.

Agents must:

- o Store the ARI for all ADM-defined AMM objects.
- o Calculate the value of an AMM object when required, such as when generating a Report or evaluating an Expression.
- o Implement Controls in firmware and run Controls and Macros with appropriate parameters when necessary in the context of Manager direction and Rule execution.
- o Communicate "return" values from Controls back to Managers as Reports where appropriate.
- o Persist custom AMM object definitions.
- o Add, remove, list, and describe custom AMM objects as requested by Managers.
- o Calculate the value of applying an Operator to a given set of operands, such as when evaluating an Expression.
- o Populate Reports and Tables for transmission to Managers when required.

- o Run the actions associated with SBRs and TBRs in accordance with their definitions.
- o Calculate the value of VARs when required, such as during Rule evaluation, calculating other VAR values, and generating Reports.

5.4. Data Type Mnemonics and Enumerations

While the AMM does not specify any encoding of data model elements, a common set of enumerations help to ensure that various encoding standards can interoperate.

This section defines string (mnemonic) and integer (enumeration) mechanisms for referring to AMM data and object types. Data types are separated into 4 major categories:

Category	Range
-----	-----
AMM Objects Types	0x00 - 0x0F
Primitive Types	0x10 - 0x1F
Compound Types	0x20 - 0x2F
Reserved	0x30 - 0xFF

Type Categories and Ranges

Within each category, the type of information, it's mnemonic, unique enumeration value, and whether it is considered a numeric value for expression evaluation are listed.

5.4.1. AMM Objects

AMM Objects include the set of objects identifiable using the ARI construct. The type field of the ARI MUST be one of these values. AMM Objects MUST be identified as follows.

Structure	Mnemonic	Enumeration	Numeric
Constant	CONST	0	No
Control	CTRL	1	No
Externally Defined Data	EDD	2	No
Literal	LIT	3	No
Macro	MAC	4	No
Operator	OPER	5	No
Report	RPT	6	No
Report Template	RPTT	7	No
State-Based Rule	SBR	8	No
Table	TBL	9	No
Table Template	TBLT	10	No
Time-Based Rule	TBR	11	No
Variable	VAR	12	No
Reserved		13-15	No

5.4.2. Primitive Data Types

Primitive data include the basic set of objects that must be encoded to transfer AMM objects. All AMM objects are built from combinations of these primitive types. Primitive types MUST be identified as follows.

Basic Data Type	Mnemonic	Enumeration	Numeric
-----	-----	-----	-----
Boolean	BOOL	16	No
BYTE	BYTE	17	No
Character String	STR	18	No
Signed 32-bit Integer	INT	19	Yes
Unsigned 32-bit Integer	UINT	20	Yes
Signed 64-bit Integer	VAST	21	Yes
Unsigned 64-bit Integer	UVAST	22	Yes
Single-Precision Floating Point	REAL32	23	Yes
Double-Precision Floating Point	REAL64	24	Yes
Reserved		25-31	No

5.4.3. Compound Data Types

Compound data include combinations of primitive data types, to include collections. Compound types MUST be identified as follows.

Compound/Special Data Type	Mnemonic	Enumeration	Numeric
-----	-----	-----	-----
Time Value	TV	32	No
Timestamp	TS	33	No
Type-Name-Value	TNV	34	No
Type-Name-Value Collection	TNVC	35	No
AMM Resource Identifier	ARI	36	No
ARI Collection	AC	37	No
Expression	EXPR	38	No
Byte String	BYTESTR	39	No
Reserved - Protocol		40-47	No

5.4.4. Numeric Promotions

When attempting to evaluate operators of different types, an Agent may need to promote operands until they are of the correct type. For example, if an Operator is given both an INT and a REAL32, the INT should be promoted to a REAL32 before the Operator is applied.

Listing legal promotion rules is mandatory for ensuring that behavior is similar across multiple implementations of Agents and Managers. The listing of legal promotions in the AMM are listed in Figure 2. In this Figure, operands are listed across the top row and down the first column. The resultant type of the promotion is listed in the table at their intersection.

	INT	UINT	VAST	UFAST	REAL32	REAL64
INT	INT	INT	VAST	UNK	REAL32	REAL64
UINT	INT	UINT	VAST	UFAST	REAL32	REAL64
VAST	VAST	VAST	VAST	VAST	REAL32	REAL64
UFAST	UNK	UFAST	VAST	UFAST	REAL32	REAL64
REAL32	REAL32	REAL32	REAL32	REAL32	REAL32	REAL64
REAL64	REAL64	REAL64	REAL64	REAL64	REAL64	REAL64

Figure 2: Numeric Promotions

The AMM does not permit promotions between non-numeric types, and numeric promotions not listed in this section are not allowed. Any attempt to perform an illegal promotion SHOULD result in an error.

5.4.5. Numeric Conversions

Variables, Expressions, and Predicates are typed values. When attempting to assign a value of a different type, a numeric conversion must be performed. Any numeric type may be converted to any other numeric type in accordance with the C rules for arithmetic type conversions.

6. JSON ADM Template

This section provides an ADM template in the form of a JSON document and describes the JSON representation of AMM objects that MUST be used to populate this JSON ADM template.

It is not required that these JSON encodings be used to encode the transmission of AMM information over the wire in the context of a network deployment. It is also not required that only these JSON encodings be used to document ADMs and other AMM information. Since

the AMM is designed to allow for multiple encodings, the expression of ADMs in the provided JSON format is intended to support translation to other encodings without loss of information.

6.1. ADM Inclusion

ADMs expressed in conformance with this template are captured as individual JSON files. AMM Objects defined in one ADM template MAY refer to objects defined in another ADM template file. To enable type checking of these cross-ADM references, the ADM template supports the "uses" keyword to identify other ADM files that contain objects referenced in the current ADM file.

The syntax of the uses statement is as follows.

```
"uses":["file1","file2",...,"fileN"]
```

Where file_# represents a JSON-formatted ADM file defining a namespace used in this ADM file.

6.2. ADMT Object Collections

The JSON ADM Template is defined as a JSON object containing a series of arrays - one for each type of information specified in the template. There are arrays for:

- o metadata constants
- o EDD definitions
- o VAR definitions
- o RPTT definitions
- o TBLT definitions
- o CTRL definitions
- o CONST definitions
- o MAC definitions
- o OP definitions

Where each array is named after the mnemonic for the particular AMM object, as defined in Section 5.4.1, with the exception of the metadata (MDAT) array which is unique to the ADM template itself.

In particular, the template does not provide definitions for RPT, TBL, SBR, or TBR objects as these are defined dynamically in the context of a network deployment.

The general format of the JSON ADM Template is as follows.

```
{
  "Mdat" : [],
  "Edd" : [],
  "Var" : [],
  "Rptt" : [],
  "Tbtl" : [],
  "Ctrl" : [],
  "Const" : [],
  "Mac" : [],
  "Oper" : []
}
```

6.3. ADM Metadata

The metadata array contains CONST objects that provide information about the ADM itself.

(STR) name

This is the human-readable name of the ADM that should appear in message logs, user-interfaces, and other human-facing applications.

(STR) namespace

This is the Moderated Namespace of the ADM, as defined in Section 5.1.1 and string-encoded in accordance with Section 5.1.5.1.

(STR) version

This is a string representation of the version of the ADM. ADM version representations are formatted at the discretion of the publishing organization.

(STR) organization

This is the name of the issuing organization for this ADM.

Metadata objects are encoded in the same way as CONST objects, in accordance with Section 6.6.2.

6.4. Type Encodings

This section describes the JSON encoding of AMM data types defined in Section 5.2.

6.4.1. Primitive Type Encoding

JSON data types generally have direct support for the AMM primitive data types. The mapping of AMM primitive types to JSON data types is provided in Table 3.

AMM Type	JSON Encoding
BYTE	number (0 <= # <= 256)
INT	number
UINT	number
VAST	number
UVAST	number
REAL32	number
REAL64	number
STRING	string
BOOL	boolean

Table 3: Primitive Type Encoding

6.4.2. Derived Type Encoding

In cases where an AMM derived type is simply a special interpretation of a primitive type, the JSON encoding of the derived type will be the same as the JSON encoding of the primitive type from which it derives.

6.4.2.1. Type-Name-Value

A TNV is encoded as a JSON object with three elements: "type", "name", and "value". For each item in a TNV, there are three acceptable formulations that can be used to represent the item, as

illustrated in the following table. For the examples in this table, consider the REAL32 value of PI as 3.14159.

Desc	Example
Full	<code>{"type": "REAL32", "name": "PI", "value": 3.14159}</code>
Named Type	<code>{"type": "REAL32", "name": "PI", "value": null}</code>
Anonymous Type	<code>{"type": "REAL32", "name": null, "value": null}</code>
Anonymous Type Value	<code>{"type": "REAL32", "name": null, "value": 3.14159}</code>
Anonymous Value	<code>{"type": null, "name": null, "value": 3.14159}</code>

Table 4: TNV Formulations

6.4.3. Collection Encoding

The TNVC and AC collections are encoded as JSON arrays, with each object in the array represented in accordance with the JSON encoding for that object type (TNV or ARI, respectively).

An Expression is encoded as a JSON object with two elements: a type and a postfix-expr. The description of these elements is as follows.

(UINT) type

The data type of the evaluation of the initializer expression.

(AC) postfix-expr

A JSON array of elements where each element is a JSON encoding of an ARI in conformance to Section 6.5.

The following is an example of a JSON encoding of an EXPR object.

```
"type": "UINT",
"postfix-expr": ["Edd.item1", "Edd.item2", "Oper.+UINT"]
```

6.5. ARI Encoding

An ARI may be encoded as either a string or as a JSON object, with the two representations being unambiguously interchangeable. Additionally, there exists a long-form and short-form encoding of the ARI.

String encodings provide a more compact and human-readable representation of the ARI. When an ARI is represented as a string in a JSON object, it MUST be encoded in accordance with Section 5.1.5.1. If the ARI references an object that is defined in the current ADM, then the shortform string encoding may be used, as described in Section 5.1.5.1.1. The object name to be used in the string encoding is the same as the "nm" value for the JSON object encoding, as described below.

JSON object encoding of the ARI provides additional structure that makes ARI information verification easier. An ARI is encoded as a JSON object with three keys: namespace, object name, and parameters, encoded as follows.

ns

This element identifies the namespace within which the ARI has been defined, and encoded as a string in accordance with Section 5.1.5.1. In cases where the ARI identifies an object defined in the ADM in which it is used, the ADM's namespace may be assumed as the namespace of the ADM and this element can be omitted from the ARI JSON object.

nm

The name of an object defined in an ADM is a string defined as the concatenation of the ADMT collection defining the object, the "." separator, and the string name of the object itself. For example, an EDD defined in the Edd array and named edd1 would have the string name "Edd.edd1".

fp

ARI formal parameters, if present, are defined as an array with each element in the array representing the JSON TNV encoding of the parameter. If a default value is not defined for the parameter, then the value of the TNV MUST be omitted.

The fp element is not used when AMM objects are defined in the context of an ADM, as the ADM template for defining objects already includes parameter information. This element is used when AMM objects are defined in accordance with the JSON ADM syntax, but by network operators as part of network-specific configuration.

If the ARI JSON object has the fp element, then it MUST NOT have the ap element. An ARI MUST NOT define both formal and actual parameters in the same object instance.

ap

ARI actual parameters, if present, are defined as an array with each element of the array representing the JSON TNV encoding of the parameter. In cases where an optional parameter is not present, an empty TNV object will be used in its place for that parameter. The name element of the TNV MUST NOT be present for actual parameters.

In cases where the actual parameter is by value, then the TNV value key will hold the JSON encoding of the value of the parameter.

In cases where the actual parameter is by name, then the TNV MUST have the type "ParmName" and the value MUST be the string name of the parameter whose value should be used to populate the value of this actual parameter, as described in Section 5.1.3.2.

If the ARI JSON object has the fp element, then it MUST NOT have the ap element. An ARI MUST NOT define both formal and actual parameters in the same object instance.

The following are examples of JSON encoded ARI objects.

String Encoding	JSON Encoding
"N1/N2/Edd.edd1"	{ "ns": "N1/N2", "nm": "Edd.edd1" }
"N1/N2/Edd.edd2(UINT num=3) "	{ "ns": "N1/N2", "nm": "Edd.edd2", "fp": [{ "type": "UINT", "name": "num", value": 3 }] }
"N1/N2/Edd.edd2() "	{ "ns": "N1/N2", "nm": "Edd.edd2", "ap": [{ }] }
"N1/N2/Edd.edd2(4) "	{ "ns": "N1/N2", "nm": "Edd.edd2", "ap": [{ "type": "UINT", "value": 4 }] }
"N1/N2/Edd.edd3(<input>) "	{ "ns": "N1/N2", "nm": "Edd.edd3", "ap": [{ "type": "ParmName", value": "input" }] }

Table 5: Formal Parameter Encoding

6.6. ADM Structures

6.6.1. General Notes

The following guidelines apply to the JSON encoding of AMM objects.

Identification

Objects do not include an ARI object as part of their definition. All of the contents of an ARI are derivable in the context of the ADM and adding an ARI encoding as part of the AMM object definition would be redundant and require maintaining naming information in two places in the ADM document.

Common Elements

Every JSON encoding of an AMM object MUST have the following elements:

- * Name
The identifier of the AMM Object. This MUST be unique across all name elements defined in the ADM collection of these types of objects.
- * Description
A string description of the kind of data represented by this data item.

Formal Parameters

If an AMM object may be parameterized, then an element MUST be present in the JSON object named "parmspec" which is defined as a JSON-encoded TNVC. Each element in the TNVC representing the JSON TNV encoding of the formal parameter. If a default value is not defined for the parameter, then the value of the TNV MUST be omitted.

6.6.2. Constant (CONST) Encoding

The CONST JSON object is comprised of four elements: "name", "type", "value", and "description". The description of these elements is as follows:

Name

The identifier of the constant. This MUST be unique across all name elements for CONSTs in the ADM.

Type

The strong typing of this data value. Types MUST be one of those defined in Section 5.4.

Value

The value of the constant, expressed in the JSON encoding of the data type.

Description

A string description of the kind of data represented by this data item.

The following is an example of a JSON encoding of a CONST object.

```
"name": "PI",  
"type": "REAL64",  
"value": 3.14159,  
"description": "The value of PI."
```

6.6.3. Control (CTRL) Encoding

The CTRL JSON object is comprised of three elements: "name", "parmspec", and "description". The description of these elements is as follows:

Name

The identifier of the control. This MUST be unique across all name elements for CTRLs in the ADM.

ParmSpec

This optional item describes parameters for this control. This is encoded as an array where each element in the array is encoded as a formal parameter in accordance with Paragraph 3.

Description

A string description of the kind of data represented by this data item.

The following is an example of a JSON encoding of an CTRL object.

```
"name": "reset_src_cnts",
"parmspec": [{ "type": "STR", "name": "src" }],
"description": "This control resets counts for the given source."
```

6.6.4. Externally Defined Data (EDD) Encoding

The EDD JSON object is comprised of four elements: "name", "type", "parmspec", and "description". The description of these elements is as follows:

Name

The identifier of the EDD data item. This MUST be unique across all name elements for EDDs in the ADM.

Type

The strong typing of this data value. Types MUST be one of those defined in Section 5.4.

ParmSpec

The optional array of formal parameters encoded in accordance with Paragraph 3.

Description

A string description of the kind of data represented by this data item.

The following is an example of a JSON encoding of an EDD object.

```
"name": "num_good_tx_bcb_blks_src",
"type": "UINT",
"parmspec": [{ "type": "STR", "name": "Src" }],
"description": "Successfully Tx BCB blocks from SRC"
```

6.6.5. Macro Encoding

The Macro JSON object is comprised of three elements: "name", "definition", and "description". The description of these elements is as follows:

Name

The identifier of the macro. This MUST be unique across all name elements for MACs in the ADM.

Definition

This is a JSON array whose elements are shorthand references are in accordance with Section 6.5 and are of the type CTRL or MAC.

Description

A string description of the kind of data represented by this data item.

The following is an example of a JSON encoding of an MAC object.

```
"name": "user_list",
"definition": [{
  "nm": "Ctrl.list_vars",
  "ap": []
}],
{
  "nm": "Ctrl.list_rptts"
  "ap": []
}],
"description": "List user defined data."
```

6.6.6. Operator (OP) Encoding

The OP JSON object is comprised of four elements: "name", "result-type", "in-type", and "description". The description of these elements is as follows.

Name

The identifier of the operator. This MUST be unique across all name elements for OPs in the ADM.

Result-Type

The numeric result of applying the operator to the series of operands. This must be one of the encodings for Table 2.

In-Type

This is an ordered JSON array of operands for the operator. Each operand is a data type encoded in accordance with Table 2.

Description

A string description of the kind of data represented by this data item.

The following is an example of a JSON encoding of an OP object.

```
"name": "plusINT",
"result-type": "INT",
"in-type": ["INT", "INT"],
"description": "Int32 addition"
```

6.6.7. Table Template (TBLT) Encoding

The TBLT JSON object is comprised of four elements: "name", "columns", and "description". The description of these elements is as follows:

Name

The identifier of the table template data item. This MUST be unique across all name elements for TBLTs in the ADM.

Columns

This is a JSON array of elements, with each element representing the definition of the type of information represented in each column. Each column is described using the same encoding as a TNV described in Table 4.

Description

A string description of the kind of data represented by this data item.

The following is an example of a JSON encoding of an TBLT object.

```
"name": "keys",
"columns": [{ "type": "STR", "name": "ciphersuite_names" }],
"description": "This table lists supported cipher suites."
```

6.6.8. Report Template Encoding

The RPTT JSON object is comprised of four elements: "name", "parmspec", "definition", and "description". The description of these elements is as follows:

Name

The identifier of the report template. This MUST be unique across all name elements for RPTTs in the ADM.

ParmSpec

This optional item describes parameters for this report. This is encoded as an array where each element in the array is encoded as a formal parameter in accordance with Paragraph 3.

Definition

This is an array of data elements that represent the ordered set of information associated with the report. Each element in the array is encoded as a data item shorthand in accordance with Section 6.5.

Report item elements MAY use reference parameters in their definition. In those cases, the reference parameters in the definition list MUST match report entry parameter names from the ParmSpec element in the report template definition.

Description

A string description of the kind of data represented by this data item.

The following is an example of a JSON encoding of an RPTT object.

```

{
  "name": "default_report",
  "parmspec": [{
    "type": "STR",
    "name": "endpoint_id"
  }],
  "definition": [
    {
      "ns": "DTN:bp",
      "nm": "Edd.edd_using_a_parm",
      "ap": [{
        "type": "PARMNAME",
        "value": "endpoint_id"
      }]
    },
    {
      "ns": "DTN:bp",
      "nm": "Edd.edd_with_default ",
      "ap": [{
        "type": "INT",
        "value": ""
      }],
      {
        "ns": "DTN:bp",
        "nm": "Edd.edd_with_no_parms ",
        "ap": []
      }
    ]
  ],
  "description": "A default report."
}

```

6.6.9. Variables Encoding

The VAR JSON object is comprised of four elements: "name", "type", "initializer", and "description". The description of these elements is as follows:

Name

The identifier of the variable data item. This MUST be unique across all name elements for VARs in the ADM.

Type

The strong typing of this data value. Types MUST be one of those defined in Section 5.4.

Initializer

The expression used to establish the initial value of the variable. This initializer is an expression encoded in conformance with Section 6.4.3.

Description

A string description of the kind of data represented by this data item.

The following is an example of a JSON encoding of an VAR object.

```
{
  "name": "total_bad_tx_blks",
  "type": "UINT",
  "initializer": {
    "type": "UINT",
    "postfix-expr": [{
      "nm": "Edd.item1",
      "ap": [{
        "type": "UINT",
        "value": 0
      }]
    }, {
      "nm": "Edd.item2",
      "ap": [{
        "type": "UINT",
        "value": 1
      }]
    }, {
      "nm": "Oper.plusUINT",
      "ap": []
    }
  ]
},
  "description": "# total items (# item1 + # item2)."
}
```

6.6.10. Exemptions

Certain AMM objects are not intended to be statically defined in the context of an ADM document. Literals, Reports, Tables, State-Based Rules, and Time-Based Rules all only have meaning in the context of an operational network. These objects are defined by network operators as part of network-specific configuration and therefore not present in the ADM Template.

7. ADM Author Considerations

The AMM model provides multiple ways to represent certain types of data. This section provides informative guidance on how to express application management constructs efficiently when authoring an ADM document.

Use Parameters for Dynamic Information.

Parameters provide a powerful mechanism for expressing associative look-ups of EDD data. EDDs SHOULD be parameterized when the definition of the EDD is dependent upon run-time information. For example, if requesting the number of bytes through a specific endpoint, the construct `num_bytes("endpoint_name")` is simpler to understand and more robust to new endpoint additions than attempting to enumerate the number and name of potential endpoints when defining the ADM.

Do Not Use Parameters for Static Information.

Parameters incur bandwidth and processing costs (such as type checking) and should only be used where necessary. If an EDD object can be parameterized, but the set of parameters is known and unchanging it may be more efficient to define multiple unparameterized EDD objects instead. For example, consider a single parameterized EDD object reporting the number of bytes of data received for a specific, known set of priorities and a request to report on those bytes for the "low", "med", and "high" priorities. Below are two ways to represent these data: using parameters and not using parameters.

Parameterized EDDs	Non-Parameterized EDDs
<code>num_bytes_by_pri(low)</code>	<code>num_bytes_by_low_pri</code>
<code>num_bytes_by_pri(med)</code>	<code>num_bytes_by_med_pri</code>
<code>num_bytes_by_pri(high)</code>	<code>num_bytes_by_high_pri</code>

The use of parameters in this case only incurs the overhead of type checking, parameter encoding/decoding, and associative lookups. This situation should be avoided when deciding when to parameterize ADM objects.

Use Tables for Related Data.

In cases where multiple EDD or VAR values are likely to be evaluated together, then that information SHOULD be placed in a Table Template rather than defining multiple EDD and VAR objects. By making a Table Template, the relationships amongst various data values are preserved. Otherwise, Managers would need to remember to query multiple EDD and/or VAR objects together which is burdensome, but also results in high bandwidth and processor utilization.

8. IANA Considerations

This document defines a moderated namespace registry in Section 5.1.1.1. This registry is envisioned to be moderated by IANA. Entries in this registry are to be made through Expert Review.

This document defines a new URI scheme, "ari", as defined in Section 5.1.5.

9. Security Considerations

This document does not describe any on-the-wire encoding or other messaging syntax. It is assumed that the exchange of AMM objects between Agents and Managers occurs within the context of an appropriate network environment.

This AMM model may be extended to include the concept of Access Control Lists (ACLs) to enforce roles and responsibilities amongst Managers in the network. This access control would be implemented separately from network security mechanisms.

10. References

10.1. Normative References

- [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>>.
- [RFC3339] Klyne, G. and C. Newman, "Date and Time on the Internet: Timestamps", RFC 3339, DOI 10.17487/RFC3339, July 2002, <<https://www.rfc-editor.org/info/rfc3339>>.
- [RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, DOI 10.17487/RFC3986, January 2005, <<https://www.rfc-editor.org/info/rfc3986>>.
- [RFC4648] Josefsson, S., "The Base16, Base32, and Base64 Data Encodings", RFC 4648, DOI 10.17487/RFC4648, October 2006, <<https://www.rfc-editor.org/info/rfc4648>>.

10.2. Informative References

- [I-D.birrane-dtn-ama] Birrane, E., "Asynchronous Management Architecture", draft-birrane-dtn-ama-06 (work in progress), October 2017.

Authors' Addresses

Edward J. Birrane
Johns Hopkins Applied Physics Laboratory

Email: Edward.Birrane@jhuapl.edu

Evana DiPietro
Johns Hopkins Applied Physics Laboratory

Email: Evana.DiPietro@jhuapl.edu

David Linko
Johns Hopkins Applied Physics Laboratory

Email: David.Linko@jhuapl.edu

Delay-Tolerant Networking Working Group
Internet Draft
Intended status: Standards Track
Expires: November 2018

S. Burleigh
JPL, Calif. Inst. Of Technology
May 20, 2018

Bundle-in-Bundle Encapsulation
draft-burleigh-dtn-bibect-01.txt

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), 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 November 21, 2018.

Copyright Notice

Copyright (c) 2018 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.

Abstract

This document describes Bundle-in-Bundle Encapsulation (BIBE), a Delay-Tolerant Networking (DTN) Bundle Protocol (BP) "convergence layer" protocol that tunnels BP "bundles" through encapsulating bundles. The services provided by the BIBE convergence-layer protocol adapter encapsulate an outbound BP "bundle" in a BIBE convergence-layer protocol data unit for transmission as the payload of a bundle. Security measures applied to the encapsulating bundle may augment those applied to the encapsulated bundle. The protocol includes a mechanism for recovery from loss of an encapsulating bundle, called "custody transfer". This mechanism is adapted from the custody transfer procedures described in 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.....	2
2. Conventions used in this document.....	4
3. BIBE Design Elements.....	4
3.1. BIBE Endpoints.....	4
3.2. BIBE Protocol Data Units.....	4
3.3. Custody Signals.....	6
3.4. Custody Transfer Status Reports.....	7
4. BIBE Procedures.....	8
4.1. BPDU Transmission.....	8
4.2. BPDU Reception.....	8
4.3. Retransmission Timer Expiration.....	10
4.4. Custody Signal Reception.....	10
5. Security Considerations.....	11
6. IANA Considerations.....	11
7. References.....	11
7.1. Normative References.....	11
7.2. Informative References.....	11
8. Acknowledgments.....	11
Appendix A. For More Information.....	13
Appendix B. CDDL expression.....	14

1. Introduction

This document describes Bundle-in-Bundle Encapsulation (BIBE), a Delay-Tolerant Networking (DTN) Bundle Protocol (BP) [RFC5050] "convergence layer" protocol that tunnels BP "bundles" through encapsulating bundles.

Conformance to the bundle-in-bundle encapsulation (BIBE) specification is OPTIONAL for BP nodes. Each BP node that conforms to the BIBE specification provides a BIBE convergence-layer adapter (CLA) that is implemented within the administrative element of the BP node's application agent. Like any convergence-layer adapter, the BIBE CLA provides:

- . A transmission service that sends an outbound bundle (from the bundle protocol agent) to a peer CLA. In the case of BIBE, the sending CLA and receiving peer CLA are both BP nodes.
- . A reception service that delivers to the bundle protocol agent an inbound bundle that was sent by a peer CLA (itself a BP node) via the BIBE convergence layer protocol.

The BIBE CLA performs these services by:

- . Encapsulating outbound bundles in BIBE protocol data units, which take the form of Bundle Protocol administrative records as described later.
- . Requesting that the bundle protocol agent transmit bundles whose payloads are BIBE protocol data units.
- . Taking delivery of BIBE protocol data units that are the payloads of bundles received by the bundle protocol agent.
- . Delivering to the bundle protocol agent the bundles that are encapsulated in delivered BIBE protocol data units.

Bundle-in-bundle encapsulation may have broad utility, but the principal motivating use case is the deployment of "cross domain solutions" in secure communications. Under some circumstances a bundle may arrive at a node that is on the frontier of a sector of network topology in which augmented security is required, from which the bundle must egress at some other designated node. In that case, the bundle may be encapsulated within a bundle to which the requisite additional BP Security (BPSEC) [bpsec] extension block(s) can be attached, whose source is the point of entry into the insecure region (the "security source") and whose destination is the point of egress from the insecure region (the "security destination").

Note that:

- . If the payload of the encapsulating bundle is protected by a Bundle Confidentiality Block (BCB), then the source and destination of the encapsulated bundle are encrypted, providing defense against traffic analysis that BPSEC alone cannot offer.
- . Bundles whose payloads are BIBE protocol data units may themselves be forwarded via a BIBE convergence-layer adapter,

enabling nested bundle encapsulation to arbitrary depth as required by security policy.

- . Moreover, in the event that no single point of egress from an insecure region of network topology can be determined at the moment a bundle is to be encapsulated, multiple copies of the bundle may be encapsulated individually and forwarded to all candidate points of egress.

The protocol includes a mechanism for recovery from loss of an encapsulating bundle, called "custody transfer". This mechanism is adapted from the custody transfer procedures described in the experimental Bundle Protocol specification developed by the Delay-Tolerant Networking Research group of the Internet Research Task Force and documented in RFC 5050. Custody transfer is a convention by which the loss or corruption of BIBE encapsulating bundles can be mitigated by the exchange of other bundles, which are termed "custody signals".

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. BIBE Design Elements

3.1. BIBE Endpoints

BIBE convergence-layer protocol endpoints, also known as BIBE convergence-layer adapters (BCLAs), are the Administrative Elements of Bundle Protocol nodes that conform to the BIBE protocol specification. The node of which a given BCLA is one component is termed the BCLA's "local node".

3.2. BIBE Protocol Data Units

Notionally, a BCLA is assumed to implement in some way, for each neighboring node to which the local node issues Bundle Protocol Data Units (BPDUs), the following two data resources:

1. A "custodial transmission count" (CTC). A CTC is a monotonically increasing integer indicating the number of "custodial" BPDUs - that is, BPDUs for which custody transfer

- was requested - that have been issued to the neighboring node by the local node since instantiation of the local node.
2. A "custodial transmission database" (CTDB), a notional array of "custodial transmission items" (CTIs). The CTDB contains one CTI for each custodial BPDU issued to the neighboring node, by the local node, for which (a) no custody disposition has yet been received in any custody signal (as discussed later) and (b) the bundle encapsulated in that BPDU has not yet been destroyed due to, e.g., time-to-live expiration. Each CTI notionally contains:
 - a. A reference to the bundle encapsulated in the corresponding BPDU.
 - b. The "transmission ID" of the corresponding BPDU, as discussed below.
 - c. A "retransmission time" indicating the time by which custody disposition for the corresponding BPDU is expected.

A BIBE protocol data unit is a Bundle Protocol administrative record whose record type code is 3 (i.e., bit pattern 0011), constructed as follows.

Each BPDU SHALL be represented as a CBOR array. The number of elements in the array SHALL be 3.

The first item of the BPDU array SHALL be the "transmission ID" for the BPDU, represented as a CBOR unsigned integer. The transmission ID for a BPDU for which custody transfer is NOT requested SHALL be zero. The transmission ID for a BPDU for which custody transfer IS requested SHALL be the current value of the local node's custodial transmission count, plus 1.

The second item of the BPDU array SHALL be the BPDU's retransmission time (i.e., the time by which custody disposition for this BPDU is expected), represented as a CBOR unsigned integer. Retransmission time for a BPDU for which custody transfer is NOT requested SHALL be zero. Retransmission time for a BPDU for which custody transfer IS requested SHALL take the form of a "DTN Time" as defined in the Bundle Protocol specification; determination of the value of retransmission time is an implementation matter that is beyond the scope of this specification and may be dynamically responsive to changes in connectivity.

The third item of the BPDU array SHALL be a single BP bundle, termed the "encapsulated bundle", represented as a CBOR byte string.

3.3. Custody Signals

A "custody signal" is defined as a Bundle Protocol administrative record whose record type code is 4 (i.e., bit pattern 0100) and whose content is constructed as follows.

The content of each custody signal SHALL be represented as a CBOR array. The number of elements in the array SHALL be 2.

The first item of the custody signal content array SHALL be a disposition code represented as a CBOR unsigned integer. Valid disposition codes are defined as follows:

Value	Meaning
0	Custody accepted.
1	No further information.
2	Reserved for future use.
3	Redundant (reception by a node that already has a copy of this bundle).
4	Depleted storage.
5	Destination endpoint ID unintelligible.
6	No known route destination from here.

7	No timely contact with next node on route.
8	Block unintelligible.
(other)	Reserved for future use.

Figure 1: Disposition Codes

The second item of the custody signal content array SHALL be a "disposition scope report", represented as a CBOR indefinite-length array. Each item of the disposition scope report array SHALL be a "disposition scope sequence", represented as a CBOR array of two elements. The first element of each disposition scope sequence array SHALL be the first transmission ID in a sequence of 1 or more consecutive transmission IDs corresponding to BPDUs to which the custody signal's disposition is declared to apply; the second element of each disposition scope sequence array SHALL be the number of transmission IDs in that sequence. Both are represented as CBOR unsigned integers.

A custody signal constitutes an assertion by the source of that administrative bundle that the indicated disposition code applies to all BPDUs identified by the transmission IDs enumerated in the custody signal's disposition scope report. If the disposition code is zero, then the source of the custody signal has accepted custody of all bundles that were encapsulated in the indicated BPDUs. Otherwise the source of the custody signal has refused custody of all bundles that were encapsulated in the indicated BPDUs, for the indicated reason.

3.4. Custody Transfer Status Reports

A "custody transfer status report" is a bundle status report with the "reporting node attempted custody transfer" flag set to 1.

4. BIBE Procedures

4.1. BPDU Transmission

When a BCLA is requested by the bundle protocol agent to send a bundle to the peer BCLA(s) included in the BP endpoint identified by a specified BP endpoint ID:

- . The BCLA SHALL generate, as defined in Section 6.2 of the Bundle Protocol specification (a work in progress), a BPDU for which the third element of the content array is the bundle that is to be transmitted. The destination of the bundle whose payload is the BPDU (termed the "encapsulating bundle") SHALL be the specified BP endpoint. Selection of the values of the parameters governing the forwarding of the encapsulating bundle, other than the destination endpoint ID, is an implementation matter. The parameter values governing the forwarding of the BPDU's encapsulated bundle MAY be consulted for this purpose.
- . Note that any transmission request presented to a BCLA MAY request that the transmission be subject to Custody Transfer, provided that the destination EID of the request identifies a singleton endpoint.
- . If Custody Transfer is requested:
 - o The first element of the BPDU's content array MUST be the BPDU's transmission ID, which SHALL be 1 more than the current value of the BCLA's CTC for the node that is the sole occupant of the BPDU's destination endpoint.
 - o The second element of the BPDU's content array MUST be the BPDU's retransmission time as discussed in 3.2 above.
 - o The bundle protocol agent MUST add the retention constraint "Custody accepted" to the encapsulated bundle.
 - o The BCLA MAY establish a retransmission timer for the encapsulated bundle. If a retransmission timer is established, it MUST be set to expire at the BPDU's retransmission time.
- . Otherwise, the first two elements of the BPDU's content array MUST both be zero.

Note that the custody transfer retransmission timer mechanism provides a means of recovering from loss of an encapsulating bundle as indicated by non-arrival of a responding custody signal.

4.2. BPDU Reception

When a BCLA receives a BPDU from the bundle protocol agent (that is, upon delivery of the payload of an encapsulating bundle):

- . If Custody Transfer was requested for this BPDU (as indicated by a non-zero value of transmission ID):
 - o If the encapsulated bundle has the same source node ID, creation timestamp, and (if that bundle is a fragment) fragment offset and payload length as another bundle that is currently retained at the BCLA's local node, then custody transfer redundancy MUST be handled as follows:
 - . The BCLA SHALL add the BPDU's transmission ID to the disposition scope report of a pending outbound custody signal, destined for the node that was the source of the encapsulating bundle, whose disposition is the reason code from Figure 1 for "Redundant reception".,
 - o Otherwise, if the BCLA determines that its local node can neither deliver nor forward the encapsulated bundle for any of the reasons listed in Figure 1, then custody transfer has failed. Custody transfer failure SHALL be handled as follows:
 - . The BCLA SHALL add the BPDU's transmission ID to the disposition scope report of a pending outbound custody signal, destined for the node that was the source of the encapsulating bundle, whose disposition is the reason code from Figure 2 that indicates the reason for the custody transfer failure.
 - o Otherwise, custody transfer has succeeded:
 - . The BCLA SHALL add the BPDU's transmission ID to the disposition scope report of a pending outbound custody signal, destined for the node that was the source of the encapsulating bundle, whose disposition is zero (indicating that custody was accepted).
 - o In each of these three cases:
 - . The pending outbound custody signal MAY then be issued immediately, but alternatively it MAY be issued at some time in the future, possibly enabling additional BPDUs' transmission IDs to be added to the same disposition scope report.
 - . If the "request reporting of custody transfer attempted" flag in the encapsulating bundle's status report request field is set to 1, and status reporting is enabled, a custody transfer status report whose reason code is the same as the pending outbound custody signal's disposition SHOULD be generated, destined for the report-to endpoint of the encapsulating bundle.
- . If Custody Transfer was NOT requested for this BPDU, or if Custody Transfer was requested for this BPDU and custody transfer succeeded, then the encapsulated bundle SHALL be

delivered from the convergence layer adapter to the bundle protocol agent, whereupon bundle reception SHALL be performed as defined in section 5.6 of the Bundle Protocol specification (a work in progress) as usual: the encapsulated bundle may be forwarded, delivered, etc.

Note that the manner in which pending outbound custody signals are managed, disposition scope reports are aggregated, and custody signal transmission is initiated is an implementation matter that is beyond the scope of this specification. Note, however, that failure to deliver a custody signal prior to the earliest value of retransmission time among all BPDUs enumerated in the custody signal's disposition scope report may result in unnecessary retransmission of one or more BPDUs.

4.3. Retransmission Timer Expiration

Upon expiration of a retransmission timer, the BCLA SHOULD remove the corresponding CTI from the CTDB (destroying the associated retransmission timer, if any) and notify the bundle protocol agent that custodial transmission of the indicated bundle failed. This notification may cause the indicated bundle to be re-forwarded (possibly on a different route).

4.4. Custody Signal Reception

When a BCLA receives a custody signal from the bundle protocol agent (that is, upon delivery of the payload of a custody-signal-bearing bundle):

- . If the custody signal's disposition is 0 (custody acceptance), then for each transmission ID in the custody signal's disposition scope report:
 - o The bundle protocol agent MUST remove the retention constraint "Custody accepted" on the bundle referenced by the corresponding CTI.
 - o The corresponding CTI MUST be removed from the CTDB (destroying the associated retransmission timer, if any).
- . Otherwise (custody refusal), for each transmission ID in the custody signal's disposition scope report:
 - o The corresponding CTI MUST be removed from the CTDB (destroying the associated retransmission timer, if any).
 - o Any further action taken by the BCLA is implementation-specific and may depend on the reason code cited for the refusal. For example, if the custody signal's reason code was "Depleted storage", the BCLA might choose to notify the bundle protocol agent that custodial transmission of

the indicated bundle failed. If the reason code was "Redundant reception", on the other hand, this might cause the BCLA simply to instruct the bundle protocol agent to remove the retention constraint "Custody accepted" on the bundle referenced by the corresponding CTI and to revise its algorithm for computing retransmission time.

5. Security Considerations

An adversary on a DTN-based network that can delete bundles could delete a BIBE custody signal in transit. This could result in unnecessary custodial retransmission, degrading network performance.

Alternatively, an adversary on a DTN-based network that can reorder bundles could cause bundles to be delivered to a BCLA in an order that complicates the efficient construction of disposition scope reports in pending outbound custody signals. This could result in inefficient custody transfer communications, again degrading network performance.

Custody transfer in BIBE may be contraindicated in environments characterized by such attacks.

6. IANA Considerations

The BIBE specification requires IANA registration of the new BIBE administrative records (type codes 3 and 4) defined above.

7. References

7.1. Normative References

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

7.2. Informative References

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

8. 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 Carnegie Mellon University, Stephen Farrell of Trinity College Dublin, Peter Lovell of SPARTA, Inc., Manikantan Ramadas of Ohio University, and Howard Weiss of SPARTA, Inc.

The custody transfer procedures defined in this specification are adapted from the Aggregate Custody Signals draft specification authored in 2010-2012 by Sebastian Kuzminsky and Andrew Jenkins, then of the University of Colorado at Boulder.

Although the BIBE specification diverges in some ways from the original Bundle-in-Bundle Encapsulation Internet Draft authored by Susan Symington, Bob Durst, and Keith Scott of The MITRE Corporation (draft-irtf-dtnrg-bundle-encapsulation-06, 2009), the influence of that earlier document is gratefully acknowledged.

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

Appendix A.

For More Information

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

Copyright (c) 2018 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>).

Appendix B.

CDDL expression

For informational purposes, Carsten Bormann has kindly provided an expression of the Bundle Protocol specification in the CBOR Data Definition Language (CDDL). Portions of CDDL expression that bear on the custody transfer extension are presented below, somewhat edited by the authors. Note that wherever the CDDL expression is in disagreement with the textual representation of the BP specification presented in the earlier sections of this document, the textual representation rules.

```
admin-record-choice /= BIBE-PDU
```

```
BIBE-PDU = [3, [transmission-ID: uint,  
                retransmission-time: uint,  
                encapsulated-bundle: bytes,  
                admin-common]]
```

```
admin-record-choice /= custody-signal
```

```
custody-signal = [4, [disposition-code: uint,  
                     disposition-scope-report,  
                     admin-common]]
```

```
disposition-scope-report = *disposition-scope-sequence
```

```
disposition-scope-sequence = [first-transmission-ID: uint,  
                              number-of-transmission-IDs: uint]
```

Authors' Address

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

Delay-Tolerant Networking
Internet-Draft
Intended status: Standards Track
Expires: August 19, 2021

E. Birrane
K. McKeever
JHU/APL
February 15, 2021

Bundle Protocol Security Specification
draft-ietf-dtn-bpsec-27

Abstract

This document defines a security protocol providing data integrity and confidentiality services for the Bundle Protocol.

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 August 19, 2021.

Copyright Notice

Copyright (c) 2021 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
1.1.	Supported Security Services	3
1.2.	Specification Scope	4
1.3.	Related Documents	5
1.4.	Terminology	6
2.	Design Decisions	7
2.1.	Block-Level Granularity	7
2.2.	Multiple Security Sources	8
2.3.	Mixed Security Policy	9
2.4.	User-Defined Security Contexts	9
2.5.	Deterministic Processing	9
3.	Security Blocks	10
3.1.	Block Definitions	10
3.2.	Uniqueness	10
3.3.	Target Multiplicity	12
3.4.	Target Identification	13
3.5.	Block Representation	13
3.6.	Abstract Security Block	13
3.7.	Block Integrity Block	16
3.8.	Block Confidentiality Block	17
3.9.	Block Interactions	18
3.10.	Parameter and Result Identification	19
3.11.	BSP Block Examples	20
3.11.1.	Example 1: Constructing a Bundle with Security	20
3.11.2.	Example 2: Adding More Security At A New Node	21
4.	Canonical Forms	23
5.	Security Processing	24
5.1.	Bundles Received from Other Nodes	24
5.1.1.	Receiving BCBs	24
5.1.2.	Receiving BIBs	25
5.2.	Bundle Fragmentation and Reassembly	26
6.	Key Management	27
7.	Security Policy Considerations	27
7.1.	Security Reason Codes	28
8.	Security Considerations	30
8.1.	Attacker Capabilities and Objectives	30
8.2.	Attacker Behaviors and BPSec Mitigations	31
8.2.1.	Eavesdropping Attacks	31
8.2.2.	Modification Attacks	32
8.2.3.	Topology Attacks	33
8.2.4.	Message Injection	34
9.	Security Context Considerations	34
9.1.	Mandating Security Contexts	34
9.2.	Identification and Configuration	35
9.3.	Authorship	37
10.	Defining Other Security Blocks	38

11. IANA Considerations	39
11.1. Bundle Block Types	39
11.2. Bundle Status Report Reason Codes	40
11.3. Security Context Identifiers	40
12. References	41
12.1. Normative References	41
12.2. Informative References	42
Appendix A. Acknowledgements	42
Authors' Addresses	42

1. Introduction

This document defines security features for the Bundle Protocol (BP) [I-D.ietf-dtn-bpbis] and is intended for use in Delay Tolerant Networks (DTNs) to provide security services between a security source and a security acceptor. When the security source is the bundle source and when the security acceptor is the bundle destination, the security service provides end-to-end protection.

The Bundle Protocol specification [I-D.ietf-dtn-bpbis] defines DTN as referring to "a networking architecture providing communications in and/or through highly stressed environments" where "BP may be viewed as sitting at the application layer of some number of constituent networks, forming a store-carry-forward overlay network". The term "stressed" environment refers to multiple challenging conditions including intermittent connectivity, large and/or variable delays, asymmetric data rates, and high bit error rates.

It should be presumed that the BP will be deployed such that the network cannot be trusted, posing the usual security challenges related to confidentiality and integrity. However, the stressed nature of the BP operating environment imposes unique conditions where usual transport security mechanisms may not be sufficient. For example, the store-carry-forward nature of the network may require protecting data at rest, preventing unauthorized consumption of critical resources such as storage space, and operating without regular contact with a centralized security oracle (such as a certificate authority).

An end-to-end security service is needed that operates in all of the environments where the BP operates.

1.1. Supported Security Services

BPSec provides integrity and confidentiality services for BP bundles, as defined in this section.

Integrity services ensure that changes to target data within a bundle can be discovered. Data changes may be caused by processing errors, environmental conditions, or intentional manipulation. In the context of BPsec, integrity services apply to plain text in the bundle.

Confidentiality services ensure that target data is unintelligible to nodes in the DTN, except for authorized nodes possessing special information. This generally means producing cipher text from plain text and generating authentication information for that cipher text. Confidentiality, in this context, applies to the contents of target data and does not extend to hiding the fact that confidentiality exists in the bundle.

NOTE: Hop-by-hop authentication is NOT a supported security service in this specification, for two reasons.

1. The term "hop-by-hop" is ambiguous in a BP overlay, as nodes that are adjacent in the overlay may not be adjacent in physical connectivity. This condition is difficult or impossible to detect and therefore hop-by-hop authentication is difficult or impossible to enforce.
2. Hop-by-hop authentication cannot be deployed in a network if adjacent nodes in the network have incompatible security capabilities.

1.2. Specification Scope

This document defines the security services provided by the BPsec. This includes the data specification for representing these services as BP extension blocks, and the rules for adding, removing, and processing these blocks at various points during the bundle's traversal of the DTN.

BPsec addresses only the security of data traveling over the DTN, not the underlying DTN itself. Furthermore, while the BPsec protocol can provide security-at-rest in a store-carry-forward network, it does not address threats which share computing resources with the DTN and/or BPsec software implementations. These threats may be malicious software or compromised libraries which intend to intercept data or recover cryptographic material. Here, it is the responsibility of the BPsec implementer to ensure that any cryptographic material, including shared secret or private keys, is protected against access within both memory and storage devices.

Completely trusted networks are extremely uncommon. Amongst untrusted networks, different networking conditions and operational

considerations require varying strengths of security mechanism. Mandating a single security context may result in too much security for some networks and too little security in others. It is expected that separate documents define different security contexts for use in different networks. A set of default security contexts are defined in ([I-D.ietf-dtn-bpsec-default-sc]) and provide basic security services for interoperability testing and for operational use on the terrestrial Internet.

This specification addresses neither the fitness of externally-defined cryptographic methods nor the security of their implementation.

This specification does not address the implementation of security policy and does not provide a security policy for the BPsec. Similar to cipher suites, security policies are based on the nature and capabilities of individual networks and network operational concepts. This specification does provide policy considerations when building a security policy.

With the exception of the Bundle Protocol, this specification does not address how to combine the BPsec security blocks with other protocols, other BP extension blocks, or other best practices to achieve security in any particular network implementation.

1.3. 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 DTNs and identifies certain security assumptions made by existing Internet protocols that are not valid in a DTN.

The Bundle Protocol [I-D.ietf-dtn-bpbis] defines the format and processing of bundles, defines the extension block format used to represent BPsec security blocks, and defines the canonical block structure used by this specification.

The Concise Binary Object Representation (CBOR) format [RFC8949] defines a data format that allows for small code size, fairly small message size, and extensibility without version negotiation. The block-specific-data associated with BPsec security blocks are encoded in this data format.

The Bundle Security Protocol [RFC6257] and Streamlined Bundle Security Protocol [I-D.birrane-dtn-sbsp] documents introduced the

concepts of using BP extension blocks for security services in a DTN. The BPsec is a continuation and refinement of these documents.

1.4. 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 BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

This section defines terminology either unique to the BPsec or otherwise necessary for understanding the concepts defined in this specification.

- o Bundle Destination - the node which receives a bundle and delivers the payload of the bundle to an application. Also, the Node ID of the Bundle Protocol Agent (BPA) receiving the bundle. The bundle destination acts as the security acceptor for every security target in every security block in every bundle it receives.
- o Bundle Source - the node which originates a bundle. Also, the Node ID of the BPA originating the bundle.
- o Cipher Suite - a set of one or more algorithms providing integrity and/or confidentiality services. Cipher suites may define user parameters (e.g. secret keys to use) but do not provide values for those parameters.
- o Forwarder - any node that transmits a bundle in the DTN. Also, the Node ID of the BPA that sent the bundle on its most recent hop.
- o Intermediate Receiver, Waypoint, or Next Hop - any node that receives a bundle from a Forwarder that is not the Bundle Destination. Also, the Node ID of the BPA at any such node.
- 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 in advance by the bundle or any BPAs in the DTN.
- o Security Acceptor - a bundle node that processes and dispositions one or more security blocks in a bundle. Security acceptors act as the endpoint of a security service represented in a security block. They remove the security blocks they act upon as part of processing and disposition. Also, the Node ID of that node.
- o Security Block - a BPsec extension block in a bundle.

- o Security Context - the set of assumptions, algorithms, configurations and policies used to implement security services.
- o Security Operation - the application of a given security service to a security target, notated as OP(security service, security target). For example, OP(bcb-confidentiality, payload). Every security operation in a bundle MUST be unique, meaning that a given security service can only be applied to a security target once in a bundle. A security operation is implemented by a security block.
- o Security Service - a process that gives some protection to a security target. For example, this specification defines security services for plain text integrity (bib-integrity), and authenticated plain text confidentiality with additional authenticated data (bcb-confidentiality).
- o Security Source - a bundle node that adds a security block to a bundle. Also, the Node ID of that node.
- o Security Target - the block within a bundle that receives a security service as part of a security operation.
- o Security Verifier - a bundle node that verifies the correctness of one or more security blocks in a bundle. Unlike security acceptors, security verifiers do not act as the endpoint of a security service and do not remove verified security blocks. Also, the Node ID of that node.

2. Design Decisions

The application of security services in a DTN is a complex endeavor that must consider physical properties of the network (such as connectivity and propagation times), policies at each node, application security requirements, and current and future threat environments. This section identifies those desirable properties that guide design decisions for this specification and are necessary for understanding the format and behavior of the BPsec protocol.

2.1. Block-Level Granularity

Security services within this specification must allow different blocks within a bundle to have different security services applied to them.

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 represent different types of information with different security needs.

For example, a payload block might be encrypted to protect its contents and an extension block containing summary information related to the payload might be integrity signed but unencrypted to provide waypoints access to payload-related data without providing access to the payload.

2.2. Multiple Security Sources

A bundle can have multiple security blocks and these blocks can have different security sources. BPsec implementations **MUST NOT** assume that all blocks in a bundle have the same security operations applied to them.

The Bundle Protocol allows extension blocks to be added to a bundle at any time during its existence in the DTN. When a waypoint adds a new extension block to a bundle, that extension block **MAY** have security services applied to it by that waypoint. Similarly, a waypoint **MAY** add a security service to an existing block, consistent with its security policy.

When a waypoint adds a security service to the bundle, the waypoint is the security source for that service. The security block(s) which represent that service in the bundle may need to record this security source as the bundle destination might need this information for processing.

For example, a bundle source may choose to apply an integrity service to its plain text payload. Later a waypoint node, representing a gateway to another portion of the DTN, may receive the bundle and choose to apply a confidentiality service. In this case, the integrity security source is the bundle source and the confidentiality security source is the waypoint node.

In cases where the security source and security acceptor are not the bundle source and bundle destination, it is possible that the bundle will reach the bundle destination prior to reaching a security acceptor. In cases where this may be a practical problem, it is recommended that solutions such as bundle encapsulation can be used to ensure that a bundle be delivered to a security acceptor prior to being delivered to the bundle destination. Generally, if a bundle reaches a waypoint that has the appropriate configuration and policy

to act as a security acceptor for a security service in the bundle, then the waypoint should act as that security acceptor.

2.3. Mixed Security Policy

The security policy enforced by nodes in the DTN may differ.

Some waypoints will have security policies that require evaluating security services even if they are not the bundle destination or the final intended acceptor of the service. For example, a waypoint could choose to verify an integrity service even though the waypoint is not the bundle destination and the integrity service will be needed by other nodes along the bundle's path.

Some waypoints will determine, through policy, that they are the intended recipient of the security service and terminate the security service in the bundle. For example, a gateway node could determine that, even though it is not the destination of the bundle, it should verify and remove a particular integrity service or attempt to decrypt a confidentiality service, before forwarding the bundle along its path.

Some waypoints could understand security blocks but refuse to process them unless they are the bundle destination.

2.4. User-Defined Security Contexts

A security context is the union of security algorithms (cipher suites), policies associated with the use of those algorithms, and configuration values. Different contexts may specify different algorithms, different policies, or different configuration values used in the implementation of their security services. BPsec provides a mechanism to define security contexts. Users may select from registered security contexts and customize those contexts through security context parameters.

For example, some users might prefer a SHA2 hash function for integrity whereas other users might prefer a SHA3 hash function. Providing either separate security contexts or a single, parameterized security context allows users flexibility in applying the desired cipher suite, policy, and configuration when populating a security block.

2.5. Deterministic Processing

Whenever a node determines that it must process more than one security block in a received bundle (either because the policy at a waypoint states that it should process security blocks or because the

node is the bundle destination) the order in which security blocks are processed must be deterministic. All nodes must impose this same deterministic processing order for all security blocks. This specification provides determinism in the application and evaluation of security services, even when doing so results in a loss of flexibility.

3. Security Blocks

3.1. Block Definitions

This specification defines two types of security block: the Block Integrity Block (BIB) and the Block Confidentiality Block (BCB).

The BIB is used to ensure the integrity of its plain text security target(s). The integrity information in the BIB MAY be verified by any node along the bundle path from the BIB security source to the bundle destination. Waypoints add or remove BIBs from bundles in accordance with their security policy. BIBs are never used for integrity protection of the cipher text provided by a BCB. Because security policy at BPsec nodes may differ regarding integrity verification, BIBs do not guarantee hop-by-hop authentication, as discussed in Section 1.1.

The BCB indicates that the security target(s) have been encrypted at the BCB security source in order to protect their content while in transit. The BCB is decrypted by security acceptor nodes in the network, up to and including the bundle destination, as a matter of security policy. BCBs additionally provide integrity protection mechanisms for the cipher text they generate.

3.2. Uniqueness

Security operations in a bundle MUST be unique; the same security service MUST NOT be applied to a security target more than once in a bundle. Since a security operation is represented by a security block, this means that multiple security blocks of the same type cannot share the same security targets. A new security block MUST NOT be added to a bundle if a pre-existing security block of the same type is already defined for the security target of the new security block.

This uniqueness requirement ensures that there is no ambiguity related to the order in which security blocks are processed or how security policy can be specified to require certain security services be present in a bundle.

Using the notation `OP(service, target)`, several examples illustrate this uniqueness requirement.

- o Signing the payload twice: The two operations `OP(bib-integrity, payload)` and `OP(bib-integrity, payload)` are redundant and MUST NOT both be present in the same bundle at the same time.
- o Signing different blocks: The two operations `OP(bib-integrity, payload)` and `OP(bib-integrity, extension_block_1)` are not redundant and both may be present in the same bundle at the same time. Similarly, the two operations `OP(bib-integrity, extension_block_1)` and `OP(bib-integrity, extension_block_2)` are also not redundant and may both be present in the bundle at the same time.
- o Different Services on same block: The two operations `OP(bib-integrity, payload)` and `OP(bcb-confidentiality, payload)` are not inherently redundant and may both be present in the bundle at the same time, pursuant to other processing rules in this specification.
- o Different services from different block types: The notation `OP(service, target)` refers specifically to a security block, as the security block is the embodiment of a security service applied to a security target in a bundle. Were some Other Security Block (OSB) to be defined providing an integrity service, then the operations `OP(bib-integrity, target)` and `OP(osb-integrity, target)` MAY both be present in the same bundle if so allowed by the definition of the OSB, as discussed in Section 10.

NOTES:

A security block may be removed from a bundle as part of security processing at a waypoint node with a new security block being added to the bundle by that node. In this case, conflicting security blocks never co-exist in the bundle at the same time and the uniqueness requirement is not violated.

A cipher text integrity mechanism (such as associated authenticated data) calculated by a cipher suite and transported in a BCB is considered part of the confidentiality service and, therefore, unique from the plain text integrity service provided by a BIB.

The security blocks defined in this specification (BIB and BCB) are designed with the intention that the BPA adding these blocks is the authoritative source of the security service. If a BPA adds a BIB on a security target, then the BIB is expected to be

the authoritative source of integrity for that security target. If a BPA adds a BCB to a security target, then the BCB is expected to be the authoritative source of confidentiality for that security target. More complex scenarios, such as having multiple nodes in a network sign the same security target, can be accommodated using the definition of custom security contexts (Section 9) and/or the definition of other security blocks (Section 10).

3.3. Target Multiplicity

A single security block MAY represent multiple security operations as a way of reducing the overall number of security blocks present in a bundle. In these circumstances, reducing the number of security blocks in the bundle reduces the amount of redundant information in the bundle.

A set of security operations can be represented by a single security block when all of the following conditions are true.

- o The security operations apply the same security service. For example, they are all integrity operations or all confidentiality operations.
- o The security context parameters for the security operations are identical.
- o The security source for the security operations is the same, meaning the set of operations are being added by the same node.
- o No security operations have the same security target, as that would violate the need for security operations to be unique.
- o None of the security operations conflict with security operations already present in the bundle.

When representing multiple security operations in a single security block, the information that is common across all operations is represented once in the security block, and the information which is different (e.g., the security targets) are represented individually.

It is RECOMMENDED that if a node processes any security operation in a security block that it process all security operations in the security block. This allows security sources to assert that the set of security operations in a security block are expected to be processed by the same security acceptor. However, the determination of whether a node actually is a security acceptor or not is a matter of the policy of the node itself. In cases where a receiving node

determines that it is the security acceptor of only a subset of the security operations in a security block, the node may choose to only process that subset of security operations.

3.4. Target Identification

A security target is a block in the bundle to which a security service applies. This target must be uniquely and unambiguously identifiable when processing a security block. The definition of the extension block header from [I-D.ietf-dtn-bpbis] provides a "Block Number" field suitable for this purpose. Therefore, a security target in a security block MUST be represented as the Block Number of the target block.

3.5. Block Representation

Each security block uses the Canonical Bundle Block Format as defined in [I-D.ietf-dtn-bpbis]. That is, each security block is comprised of the following elements:

- o block type code
- o block number
- o block processing control flags
- o CRC type
- o block-type-specific-data
- o CRC field (if present)

Security-specific information for a security block is captured in the block-type-specific-data field.

3.6. Abstract Security Block

The structure of the security-specific portions of a security block is identical for both the BIB and BCB Block Types. Therefore, this section defines an Abstract Security Block (ASB) data structure and discusses the definition, processing, and other constraints for using this structure. An ASB is never directly instantiated within a bundle, it is only a mechanism for discussing the common aspects of BIB and BCB security blocks.

The fields of the ASB SHALL be as follows, listed in the order in which they must appear. The encoding of these fields MUST be in accordance with the canonical forms provided in Section 4.

Security Targets:

This field identifies the block(s) targeted by the security operation(s) represented by this security block. Each target block is represented by its unique Block Number. This field SHALL be represented by a CBOR array of data items. Each target within this CBOR array SHALL be represented by a CBOR unsigned integer. This array MUST have at least 1 entry and each entry MUST represent the Block Number of a block that exists in the bundle. There MUST NOT be duplicate entries in this array. The order of elements in this list has no semantic meaning outside of the context of this block. Within the block, the ordering of targets must match the ordering of results associated with these targets.

Security Context Id:

This field identifies the security context used to implement the security service represented by this block and applied to each security target. This field SHALL be represented by a CBOR unsigned integer. The values for this Id should come from the registry defined in Section 11.3

Security Context Flags:

This field identifies which optional fields are present in the security block. This field SHALL be represented as a CBOR unsigned integer whose contents shall be interpreted as a bit field. Each bit in this bit field indicates the presence (bit set to 1) or absence (bit set to 0) of optional data in the security block. The association of bits to security block data is defined as follows.

Bit 0 (the least-significant bit, 0x01): Security Context Parameters Present Flag.

Bit >0 Reserved

Implementations MUST set reserved bits to 0 when writing this field and MUST ignore the values of reserved bits when reading this field. For unreserved bits, a value of 1 indicates that the associated security block field MUST be included in the security block. A value of 0 indicates that the associated security block field MUST NOT be in the security block.

Security Source:

This field identifies the Endpoint that inserted the security block in the bundle. This field SHALL be represented by a CBOR array in accordance with [I-D.ietf-dtn-bpbis] rules for representing Endpoint Identifiers (EIDs).

Security Context Parameters (Optional):

This field captures one or more security context parameters that should be used when processing the security service described by this security block. This field SHALL be represented by a CBOR array. Each entry in this array is a single security context parameter. A single parameter SHALL also be represented as a CBOR array comprising a 2-tuple of the id and value of the parameter, as follows.

- * **Parameter Id.** This field identifies which parameter is being specified. This field SHALL be represented as a CBOR unsigned integer. Parameter Ids are selected as described in Section 3.10.
- * **Parameter Value.** This field captures the value associated with this parameter. This field SHALL be represented by the applicable CBOR representation of the parameter, in accordance with Section 3.10.

The logical layout of the parameters array is illustrated in Figure 1.

Parameter 1		Parameter 2		...	Parameter N	
Id	Value	Id	Value		Id	Value

Figure 1: Security Context Parameters

Security Results:

This field captures the results of applying a security service to the security targets of the security block. This field SHALL be represented as a CBOR array of target results. Each entry in this array represents the set of security results for a specific security target. The target results MUST be ordered identically to the Security Targets field of the security block. This means that the first set of target results in this array corresponds to the first entry in the Security Targets field of the security block, and so on. There MUST be one entry in this array for each entry in the Security Targets field of the security block.

The set of security results for a target is also represented as a CBOR array of individual results. An individual result is represented as a 2-tuple of a result id and a result value, defined as follows.

- * Result Id. This field identifies which security result is being specified. Some security results capture the primary output of a cipher suite. Other security results contain additional annotative information from cipher suite processing. This field SHALL be represented as a CBOR unsigned integer. Security result Ids will be as specified in Section 3.10.
- * Result Value. This field captures the value associated with the result. This field SHALL be represented by the applicable CBOR representation of the result value, in accordance with Section 3.10.

The logical layout of the security results array is illustrated in Figure 2. In this figure there are N security targets for this security block. The first security target contains M results and the Nth security target contains K results.

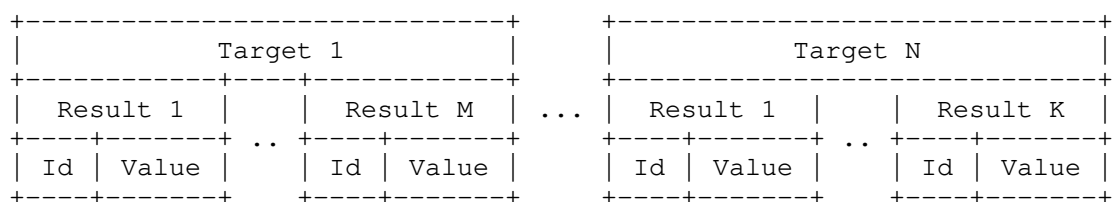


Figure 2: Security Results

3.7. Block Integrity Block

A BIB is a bundle extension block with the following characteristics.

The Block Type Code value is as specified in Section 11.1.

The block-type-specific-data field follows the structure of the ASB.

A security target listed in the Security Targets field MUST NOT reference a security block defined in this specification (e.g., a BIB or a BCB).

The Security Context MUST utilize an authentication mechanism or an error detection mechanism.

Notes:

- o 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.
- o Since OP(bib-integrity, target) is allowed only once in a bundle per target, it is RECOMMENDED that users wishing to support multiple integrity mechanisms for the same target define a multi-result security context. Such a context could generate multiple security results for the same security target using different integrity-protection mechanisms or different configurations for the same integrity-protection mechanism.
- o A BIB is used to verify the plain text integrity of its security target. However, a single BIB MAY include security results for blocks other than its security target when doing so establishes a needed relationship between the BIB security target and other blocks in the bundle (such as the primary block).
- o Security information MAY be checked at any hop on the way to the bundle destination that has access to the required keying information, in accordance with Section 3.9.

3.8. Block Confidentiality Block

A BCB is a bundle extension block with the following characteristics.

The Block Type Code value is as specified in Section 11.1.

The Block Processing Control flags value can be set to whatever values are required by local policy with the following exceptions. BCB blocks MUST have the "block must be replicated in every fragment" flag set if one of the targets is the payload block. Having that BCB in each fragment indicates to a receiving node that the payload portion of each fragment represents cipher text. BCB blocks MUST NOT have the "block must be removed from bundle if it can't be processed" flag set. Removing a BCB from a bundle without decrypting its security targets removes information from the bundle necessary for their later decryption.

The block-type-specific-data fields follow the structure of the ASB.

A security target listed in the Security Targets field can reference the payload block, a non-security extension block, or a BIB. A BCB MUST NOT include another BCB as a security target. A BCB MUST NOT target the primary block. A BCB MUST NOT target a BIB block unless it shares a security target with that BIB block.

Any Security Context used by a BCB MUST utilize a confidentiality cipher that provides authenticated encryption with associated data (AEAD).

Additional information created by a cipher suite (such as an authentication tag) can be placed either in a security result field or in the generated cipher text. The determination of where to place this information is a function of the cipher suite and security context used.

The BCB modifies the contents of its security target(s). When a BCB is applied, the security target body data are encrypted "in-place". Following encryption, the security target block-type-specific-data field contains cipher text, not plain text.

Notes:

- o It is RECOMMENDED that designers carefully consider the effect of setting flags that delete the bundle in the event that this block cannot be processed.
- o The BCB block processing control flags can be set independently from the processing control flags of the security target(s). The setting of such flags should be an implementation/policy decision for the encrypting node.

3.9. Block Interactions

The 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 a security target of a BCB is also a security target of a BIB, an undesirable condition occurs where a waypoint would be unable to validate the BIB because one of its security target's contents have been encrypted by a BCB. To address this situation the following processing rules MUST be followed.

- o When adding a BCB to a bundle, if some (or all) of the security targets of the BCB also match all of the security targets of an existing BIB, then the existing BIB MUST also be encrypted. This can be accomplished by either adding a new BCB that targets the existing BIB, or by adding the BIB to the list of security targets for the BCB. Deciding which way to represent this situation is a matter of security policy.

- o When adding a BCB to a bundle, if some (or all) of the security targets of the BCB match some (but not all) of the security targets of a BIB then that BIB MUST be altered in the following way. Any security results in the BIB associated with the BCB security targets MUST be removed from the BIB and placed in a new BIB. This newly created BIB MUST then be encrypted. The encryption of the new BIB can be accomplished by either adding a new BCB that targets the new BIB, or by adding the new BIB to the list of security targets for the BCB. Deciding which way to represent this situation is a matter of security policy.
- o A BIB MUST NOT be added for a security target that is already the security target of a BCB as this would cause ambiguity in block processing order.
- o A BIB integrity value MUST NOT be checked if the BIB is the security target of an existing BCB. In this case, the BIB data is encrypted.
- o A BIB integrity value MUST NOT be checked if the security target associated with that value is also the security target of a BCB. In such a case, the security target data contains cipher text as it has been encrypted.
- o As mentioned in Section 3.7, a BIB MUST NOT have a BCB as its security target.

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. 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.

In cases where a security source wishes to calculate both a plain text integrity mechanism and encrypt a security target, a BCB with a security context that generates an integrity-protection mechanism as one or more additional security results MUST be used instead of adding both a BIB and then a BCB for the security target at the security source.

3.10. Parameter and Result Identification

Each security context MUST define its own context parameters and results. Each defined parameter and result is represented as the tuple of an identifier and a value. Identifiers are always

represented as a CBOR unsigned integer. The CBOR encoding of values is as defined by the security context specification.

Identifiers MUST be unique for a given security context but do not need to be unique amongst all security contexts.

An example of a security context can be found at [I-D.ietf-dtn-bpsec-default-sc].

3.11. BSP Block Examples

This section provides two examples of BPsec blocks applied to a bundle. In the first example, a single node adds several security operations to a bundle. In the second example, a waypoint node received the bundle created in the first example and adds additional security operations. In both examples, the first column represents blocks within a bundle and the second column represents the Block Number for the block, using the terminology B1...Bn for the purpose of illustration.

3.11.1. Example 1: Constructing a Bundle with Security

In this example a bundle has four non-security-related blocks: the primary block (B1), two extension blocks (B4,B5), and a payload block (B6). The bundle source wishes to provide an integrity signature of the plain text associated with the primary block, the second extension block, and the payload. The bundle source also wishes to provide confidentiality for the first extension block. The resultant bundle is illustrated in Figure 3 and the security actions are described below.

Block in Bundle	ID
Primary Block	B1
BIB OP(bib-integrity, targets=B1, B5, B6)	B2
BCB OP(hcb-confidentiality, target=B4)	B3
Extension Block (encrypted)	B4
Extension Block	B5
Payload Block	B6

Figure 3: Security at Bundle Creation

The following security actions were applied to this bundle at its time of creation.

- o An integrity signature applied to the canonical form of the primary block (B1), the canonical form of the block-type-specific-data field of the second extension block (B5) and the canonical form of the payload block (B6). This is accomplished by a single BIB (B2) with multiple targets. A single BIB is used in this case because all three targets share a security source, security context, and security context parameters. Had this not been the case, multiple BIBs could have been added instead.
- o Confidentiality for the first extension block (B4). This is accomplished by a BCB (B3). Once applied, the block-type-specific-data field of extension block B4 is encrypted. The BCB MUST hold an authentication tag for the cipher text either in the cipher text that now populates the first extension block or as a security result in the BCB itself, depending on which security context is used to form the BCB. A plain text integrity signature may also exist as a security result in the BCB if one is provided by the selected confidentiality security context.

3.11.2. Example 2: Adding More Security At A New Node

Consider that the bundle as it is illustrated in Figure 3 is now received by a waypoint node that wishes to encrypt the second extension block and the bundle payload. The waypoint security policy is to allow existing BIBs for these blocks to persist, as they may be required as part of the security policy at the bundle destination.

The resultant bundle is illustrated in Figure 4 and the security actions are described below. Note that block IDs provided here are ordered solely for the purpose of this example and not meant to impose an ordering for block creation. The ordering of blocks added to a bundle MUST always be in compliance with [I-D.ietf-dtn-bpbis].

Block in Bundle	ID
Primary Block	B1
BIB OP(bib-integrity, targets=B1)	B2
BIB (encrypted) OP(bib-integrity, targets=B5, B6)	B7
BCB OP(bcb-confidentiality, targets=B5, B6, B7)	B8
BCB OP(bcb-confidentiality, target=B4)	B3
Extension Block (encrypted)	B4
Extension Block (encrypted)	B5
Payload Block (encrypted)	B6

Figure 4: Security At Bundle Forwarding

The following security actions were applied to this bundle prior to its forwarding from the waypoint node.

- o Since the waypoint node wishes to encrypt the block-type-specific-data field of blocks B5 and B6, it MUST also encrypt the block-type-specific-data field of the BIBs providing plain text integrity over those blocks. However, BIB B2 could not be encrypted in its entirety because it also held a signature for the primary block (B1). Therefore, a new BIB (B7) is created and security results associated with B5 and B6 are moved out of BIB B2 and into BIB B7.
- o Now that there is no longer confusion of which plain text integrity signatures must be encrypted, a BCB is added to the bundle with the security targets being the second extension block (B5) and the payload (B6) as well as the newly created BIB holding their plain text integrity signatures (B7). A single new BCB is

used in this case because all three targets share a security source, security context, and security context parameters. Had this not been the case, multiple BCBs could have been added instead.

4. Canonical Forms

Security services require consistency and determinism in how information is presented to cipher suites at security sources, verifiers, and acceptors. For example, integrity services require that the same target information (e.g., the same bits in the same order) is provided to the cipher suite when generating an original signature and when validating a signature. Canonicalization algorithms transcode the contents of a security target into a canonical form.

Canonical forms are used to generate input to a security context for security processing at a BP node. If the values of a security target are unchanged, then the canonical form of that target will be the same even if the encoding of those values for wire transmission is different.

BPsec operates on data fields within bundle blocks (e.g., the block-type-specific-data field). In their canonical form, these fields MUST include their own CBOR encoding and MUST NOT include any other encapsulating CBOR encoding. For example, the canonical form of the block-type-specific-data field is a CBOR byte string existing within the CBOR array containing the fields of the extension block. The entire CBOR byte string is considered the canonical block-type-specific-data field. The CBOR array framing is not considered part of the field.

The canonical form of the primary block is as specified in [I-D.ietf-dtn-bpbis] with the following constraint.

- o CBOR values from the primary block MUST be canonicalized using the rules for Deterministically Encoded CBOR, as specified in [RFC8949].

All non-primary blocks share the same block structure and are canonicalized as specified in [I-D.ietf-dtn-bpbis] with the following constraints.

- o CBOR values from the non-primary block MUST be canonicalized using the rules for Deterministically Encoded CBOR, as specified in [RFC8949].

- o Only the block-type-specific-data field may be provided to a cipher suite for encryption as part of a confidentiality security service. Other fields within a non-primary-block MUST NOT be encrypted or decrypted and MUST NOT be included in the canonical form used by the cipher suite for encryption and decryption. These other fields MAY have an integrity protection mechanism applied to them by treating them as associated authenticated data.
- o Reserved and unassigned flags in the block processing control flags field MUST be set to 0 in a canonical form as it is not known if those flags will change in transit.

Security contexts MAY define their own canonicalization algorithms and require the use of those algorithms over the ones provided in this specification. In the event of conflicting canonicalization algorithms, algorithms defined in a security context take precedence over this specification when constructing canonical forms for that security context.

5. Security Processing

This section describes the security aspects of bundle processing.

5.1. Bundles Received from Other Nodes

Security blocks must be processed in a specific order when received by a BP node. The processing order is as follows.

- o When BIBs and BCBs share a security target, BCBs MUST be evaluated first and BIBs second.

5.1.1. Receiving BCBs

If a received bundle contains a BCB, the receiving node MUST determine whether it is the security acceptor for any of the security operations in the BCB. If so, the node MUST process those operations and remove any operation-specific information from the BCB prior to delivering data to an application at the node or forwarding the bundle. If processing a security operation fails, the target SHALL be processed according to the security policy. A bundle status report indicating the failure MAY be generated. When all security operations for a BCB have been removed from the BCB, the BCB MUST be removed from the bundle.

If the receiving node is the destination of the bundle, the node MUST decrypt any BCBs remaining in the bundle. If the receiving node is not the destination of the bundle, the node MUST process the BCB if directed to do so as a matter of security policy.

If the security policy of a node specifies that a node should have applied confidentiality to a specific security target and no such BCB is present in the bundle, then the node MUST process this security target in accordance with the security policy. It is RECOMMENDED that the node remove the security target from the bundle because the confidentiality (and possibly the integrity) of the security target cannot be guaranteed. If the removed security target is the payload block, the bundle MUST be discarded.

If an encrypted payload block cannot be decrypted (i.e., the cipher text cannot be authenticated), then the bundle MUST be discarded and processed no further. If an encrypted security target other than the payload block cannot be decrypted then the associated security target and all security blocks associated with that target MUST be discarded and processed no further. In both cases, requested status reports (see [I-D.ietf-dtn-bpbis]) MAY be generated to reflect bundle or block deletion.

When a BCB is decrypted, the recovered plain text for each security target MUST replace the cipher text in each of the security targets' block-type-specific-data fields. If the plain text is of different size than the cipher text, the CBOR byte string framing of this field must be updated to ensure this field remains a valid CBOR byte string. The length of the recovered plain text is known by the decrypting security context.

If a BCB contains multiple security operations, each operation processed by the node MUST be treated as if the security operation has been represented by a single BCB with a single security operation for the purposes of report generation and policy processing.

5.1.2. Receiving BIBs

If a received bundle contains a BIB, the receiving node MUST determine whether it is the security acceptor for any of the security operations in the BIB. If so, the node MUST process those operations and remove any operation-specific information from the BIB prior to delivering data to an application at the node or forwarding the bundle. If processing a security operation fails, the target SHALL be processed according to the security policy. A bundle status report indicating the failure MAY be generated. When all security operations for a BIB have been removed from the BIB, the BIB MUST be removed from the bundle.

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 node specifies that a node should have applied 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. It is RECOMMENDED that the node remove the security target from the bundle if the security target is not the payload or primary block. If the security target is the payload or primary block, the bundle MAY be discarded. This action can occur at any node that has the ability to verify an integrity signature, not just the bundle destination.

If a receiving node is not the security acceptor of a security operation in a BIB it MAY attempt to verify the security operation anyway to prevent forwarding corrupt data. If the verification 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 bundle destination. If the check passes, the node MUST NOT remove the security operation from the BIB prior to forwarding.

If a BIB contains multiple security operations, each operation processed by the node MUST be treated as if the security operation has been represented by a single BIB with a single security operation for the purposes of report generation and policy processing.

5.2. Bundle Fragmentation and Reassembly

If it is necessary for a node to fragment a bundle payload, and security services have been applied to that bundle, the fragmentation rules described in [I-D.ietf-dtn-bpbis] MUST be followed. As defined there and summarized here for completeness, only the payload block can be fragmented; security blocks, like all extension blocks, can never be fragmented.

Due to the complexity of payload block fragmentation, including the possibility of fragmenting payload block fragments, integrity and confidentiality operations are not to be applied to a bundle representing a fragment. Specifically, a BCB or BIB MUST NOT be added to a bundle if the "Bundle is a Fragment" flag is set in the Bundle Processing Control Flags field.

Security processing in the presence of payload block fragmentation may be handled by other mechanisms outside of the BPsec protocol or by applying BPsec blocks in coordination with an encapsulation

mechanism. A node should apply any confidentiality protection prior to performing any fragmentation.

6. Key Management

There exist a myriad of ways to establish, communicate, and otherwise manage key information in a DTN. Certain DTN deployments might follow established protocols for key management whereas other DTN deployments might require new and novel approaches. BPSec assumes that key management is handled as a separate part of network management and this specification neither defines nor requires a specific key management strategy.

7. Security Policy Considerations

When implementing BPSec, 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. No single set of policy decisions is envisioned to work for all secure DTN deployments.

- o If a bundle is received that contains combinations of security operations that are disallowed by this specification the BPA must determine how to handle the 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 a waypoint 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 BPsec blocks, replace the security operation, or some other action.
- o It is RECOMMENDED that security operations be applied to every block in a bundle and that the default behavior of a bundle agent is to use the security services defined in this specification. Designers should only deviate from the use of security operations when the deviation can be justified – such as when doing so causes downstream errors when processing blocks whose contents must be inspected or changed at one or more hops along the path.

- o BCB security contexts can alter the size of extension blocks and the payload block. Security policy SHOULD consider how changes to the size of a block could negatively effect bundle processing (e.g., calculating storage needs and scheduling transmission times).
- o Adding a BIB to a security target that has already been encrypted by a BCB is not allowed. If this condition is likely to be encountered, there are (at least) three possible policies that could handle this situation.
 1. At the time of encryption, a security context can be selected which computes a plain text integrity-protection mechanism that is included as a security context result field.
 2. The encrypted block may be replicated as a new block with a new block number and given integrity protection.
 3. 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.
- o Security policy SHOULD address whether cipher suites whose cipher text is larger than the initial plain text are permitted and, if so, for what types of blocks. Changing the size of a block may cause processing difficulties for networks that calculate block offsets into bundles or predict transmission times or storage availability as a function of bundle size. In other cases, changing the size of a payload as part of encryption has no significant impact.

7.1. Security Reason Codes

Bundle protocol agents (BPAs) must process blocks and bundles in accordance with both BP policy and BPsec policy. The decision to receive, forward, deliver, or delete a bundle may be communicated to the report-to address of the bundle, in the form of a status report, as a method of tracking the progress of the bundle through the network. The status report for a bundle may be augmented with a "reason code" explaining why the particular action was taken on the bundle.

This section describes a set of reason codes associated with the security processing of a bundle. The communication of security-related status reports might reduce the security of a network if these reports are intercepted by unintended recipients. BPsec policy

SHOULD specify the conditions in which sending security reason codes are appropriate. Examples of appropriate conditions for the use of security reason codes could include the following.

- o When the report-to address is verified as unchanged from the bundle source. This can occur by placing an appropriate BIB on the bundle primary block.
- o When the block containing a status report with a security reason code is encrypted by a BCB.
- o When a status report containing a security reason code is only sent for security issues relating to bundles and/or blocks associated with non-operational user data or otherwise with test data.
- o When a status report containing a security reason code is only sent for security issues associated with non-operational security contexts, or security contexts using non-operational configurations, such as test keys.

Security reason codes are assigned in accordance with Section 11.2 and are as described below.

Missing Security Operation:

This reason code indicates that a bundle was missing one or more required security operations. This reason code is typically used by a security verifier or security acceptor.

Unknown Security Operation:

This reason code indicates that one or more security operations present in a bundle cannot be understood by the security verifier or security acceptor for the operation. For example, this reason code may be used if a security block references an unknown security context identifier or security context parameter. This reason code should not be used for security operations for which the node is not a security verifier or security acceptor; there is no requirement that all nodes in a network understand all security contexts, security context parameters, and security services for every bundle in a network.

Unexpected Security Operation:

This reason code indicates that a receiving node is neither a security verifier nor a security acceptor for at least one security operation in a bundle. This reason code should not be seen as an error condition; not every node is a security verifier or security acceptor for every security operation in

every bundle. In certain networks, this reason code may be useful in identifying misconfigurations of security policy.

Failed Security Operation:

This reason code indicates that one or more security operations in a bundle failed to process as expected for reasons other than misconfiguration. This may occur when a security-source is unable to add a security block to a bundle. This may occur if the target of a security operation fails to verify using the defined security context at a security verifier. This may also occur if a security operation fails to be processed without error at a security acceptor.

Conflicting Security Operations:

This reason code indicates that two or more security operations in a bundle are not conformant with the BPSec specification and that security processing was unable to proceed because of a BPSec protocol violation.

8. Security Considerations

Given the nature of DTN applications, it is expected that bundles may traverse a variety of environments and devices which each pose unique security risks and requirements on the implementation of security within BPSec. For these reasons, it is important to introduce key threat models and describe the roles and responsibilities of the BPSec protocol in protecting the confidentiality and integrity of the data against those threats. This section provides additional discussion on security threats that BPSec will face and describes how BPSec security mechanisms operate to mitigate these threats.

The threat model described here is assumed to have a set of capabilities identical to those described by the Internet Threat Model in [RFC3552], but the BPSec threat model is scoped to illustrate threats specific to BPSec operating within DTN environments and therefore focuses on on-path-attackers (OPAs). In doing so, it is assumed that the DTN (or significant portions of the DTN) are completely under the control of an attacker.

8.1. Attacker Capabilities and Objectives

BPSec was designed to protect against OPA threats which may have access to a bundle during transit from its source, Alice, to its destination, Bob. An OPA node, Olive, is a non-cooperative node operating on the DTN between Alice and Bob that has the ability to receive bundles, examine bundles, modify bundles, forward bundles, and generate bundles at will in order to compromise the confidentiality or integrity of data within the DTN. There are three

classes of OPA nodes which are differentiated based on their access to cryptographic material:

- o Unprivileged Node: Olive has not been provisioned within the secure environment and only has access to cryptographic material which has been publicly-shared.
- o Legitimate Node: Olive is within the secure environment and therefore has access to cryptographic material which has been provisioned to Olive (i.e., K_M) as well as material which has been publicly-shared.
- o Privileged Node: Olive is a privileged node within the secure environment and therefore has access to cryptographic material which has been provisioned to Olive, Alice and/or Bob (i.e. K_M , K_A , and/or K_B) as well as material which has been publicly-shared.

If Olive is operating as a privileged node, this is tantamount to compromise; BPsec does not provide mechanisms to detect or remove Olive from the DTN or BPsec secure environment. It is up to the BPsec implementer or the underlying cryptographic mechanisms to provide appropriate capabilities if they are needed. It should also be noted that if the implementation of BPsec uses a single set of shared cryptographic material for all nodes, a legitimate node is equivalent to a privileged node because $K_M == K_A == K_B$. For this reason, sharing cryptographic material in this way is not recommended.

A special case of the legitimate node is when Olive is either Alice or Bob (i.e., $K_M == K_A$ or $K_M == K_B$). In this case, Olive is able to impersonate traffic as either Alice or Bob, respectively, which means that traffic to and from that node can be decrypted and encrypted, respectively. Additionally, messages may be signed as originating from one of the endpoints.

8.2. Attacker Behaviors and BPsec Mitigations

8.2.1. Eavesdropping Attacks

Once Olive has received a bundle, she is able to examine the contents of that bundle and attempt to recover any protected data or cryptographic keying material from the blocks contained within. The protection mechanism that BPsec provides against this action is the BCB, which encrypts the contents of its security target, providing confidentiality of the data. Of course, it should be assumed that Olive is able to attempt offline recovery of encrypted data, so the

cryptographic mechanisms selected to protect the data should provide a suitable level of protection.

When evaluating the risk of eavesdropping attacks, it is important to consider the lifetime of bundles on a DTN. Depending on the network, bundles may persist for days or even years. Long-lived bundles imply that the data exists in the network for a longer period of time and, thus, there may be more opportunities to capture those bundles. Additionally, bundles that are long-lived imply that the information stored within them may remain relevant and sensitive for long enough that, once captured, there is sufficient time to crack encryption associated with the bundle. If a bundle does persist on the network for years and the cipher suite used for a BCB provides inadequate protection, Olive may be able to recover the protected data either before that bundle reaches its intended destination or before the information in the bundle is no longer considered sensitive.

NOTE: Olive is not limited by the bundle lifetime and may retain a given bundle indefinitely.

NOTE: Irrespective of whether BPsec is used, traffic analysis will be possible.

8.2.2. Modification Attacks

As a node participating in the DTN between Alice and Bob, Olive will also be able to modify the received bundle, including non-BPsec data such as the primary block, payload blocks, or block processing control flags as defined in [I-D.ietf-dtn-bpbis]. Olive will be able to undertake activities which include modification of data within the blocks, replacement of blocks, addition of blocks, or removal of blocks. Within BPsec, both the BIB and BCB provide integrity protection mechanisms to detect or prevent data manipulation attempts by Olive.

The BIB provides that protection to another block which is its security target. The cryptographic mechanisms used to generate the BIB should be strong against collision attacks and Olive should not have access to the cryptographic material used by the originating node to generate the BIB (e.g., K_A). If both of these conditions are true, Olive will be unable to modify the security target or the BIB and lead Bob to validate the security target as originating from Alice.

Since BPsec security operations are implemented by placing blocks in a bundle, there is no in-band mechanism for detecting or correcting certain cases where Olive removes blocks from a bundle. If Olive removes a BCB, but keeps the security target, the security target

remains encrypted and there is a possibility that there may no longer be sufficient information to decrypt the block at its destination. If Olive removes both a BCB (or BIB) and its security target there is no evidence left in the bundle of the security operation. Similarly, if Olive removes the BIB but not the security target there is no evidence left in the bundle of the security operation. In each of these cases, the implementation of BPSec must be combined with policy configuration at endpoints in the network which describe the expected and required security operations that must be applied on transmission and are expected to be present on receipt. This or other similar out-of-band information is required to correct for removal of security information in the bundle.

A limitation of the BIB may exist within the implementation of BIB validation at the destination node. If Olive is a legitimate node within the DTN, the BIB generated by Alice with K_A can be replaced with a new BIB generated with K_M and forwarded to Bob. If Bob is only validating that the BIB was generated by a legitimate user, Bob will acknowledge the message as originating from Olive instead of Alice. Validating a BIB indicates only that the BIB was generated by a holder of the relevant key; it does not provide any guarantee that the bundle or block was created by the same entity. In order to provide verifiable integrity checks BCB should require an encryption scheme that is Indistinguishable under adaptive Chosen Ciphertext Attack (IND-CCA2) secure. Such an encryption scheme will guard against signature substitution attempts by Olive. In this case, Alice creates a BIB with the protected data block as the security target and then creates a BCB with both the BIB and protected data block as its security targets.

8.2.3. Topology Attacks

If Olive is in a OPA position within the DTN, she is able to influence how any bundles that come to her may pass through the network. Upon receiving and processing a bundle that must be routed elsewhere in the network, Olive has three options as to how to proceed: not forward the bundle, forward the bundle as intended, or forward the bundle to one or more specific nodes within the network.

Attacks that involve re-routing the packets throughout the network are essentially a special case of the modification attacks described in this section where the attacker is modifying fields within the primary block of the bundle. Given that BPSec cannot encrypt the contents of the primary block, alternate methods must be used to prevent this situation. These methods may include requiring BIBs for primary blocks, using encapsulation, or otherwise strategically manipulating primary block data. The specifics of any such

mitigation technique are specific to the implementation of the deploying network and outside of the scope of this document.

Furthermore, routing rules and policies may be useful in enforcing particular traffic flows to prevent topology attacks. While these rules and policies may utilize some features provided by BPsec, their definition is beyond the scope of this specification.

8.2.4. Message Injection

Olive is also able to generate new bundles and transmit them into the DTN at will. These bundles may either be copies or slight modifications of previously-observed bundles (i.e., a replay attack) or entirely new bundles generated based on the Bundle Protocol, BPsec, or other bundle-related protocols. With these attacks Olive's objectives may vary, but may be targeting either the bundle protocol or application-layer protocols conveyed by the bundle protocol. The target could also be the storage and compute of the nodes running the bundle or application layer protocols (e.g., a denial of service to flood on the storage of the store-and-forward mechanism; or compute which would process the packets and perhaps prevent other activities).

BPsec relies on cipher suite capabilities to prevent replay or forged message attacks. A BCB used with appropriate cryptographic mechanisms may provide replay protection under certain circumstances. Alternatively, application data itself may be augmented to include mechanisms to assert data uniqueness and then protected with a BIB, a BCB, or both along with other block data. In such a case, the receiving node would be able to validate the uniqueness of the data.

For example, a BIB may be used to validate the integrity of a bundle's primary block, which includes a timestamp and lifetime for the bundle. If a bundle is replayed outside of its lifetime, then the replay attack will fail as the bundle will be discarded. Similarly, additional blocks such as the Bundle Age may be signed and validated to identify replay attacks. Finally, security context parameters within BIB and BCB blocks may include anti-replay mechanisms such as session identifiers, nonces, and dynamic passwords as supported by network characteristics.

9. Security Context Considerations

9.1. Mandating Security Contexts

Because of the diversity of networking scenarios and node capabilities that may utilize BPsec there is a risk that a single security context mandated for every possible BPsec implementation is

not feasible. For example, a security context appropriate for a resource-constrained node with limited connectivity may be inappropriate for use in a well-resourced, well connected node.

This does not mean that the use of BPsec in a particular network is meant to be used without security contexts for interoperability and default behavior. Network designers must identify the minimal set of security contexts necessary for functions in their network. For example, a default set of security contexts could be created for use over the terrestrial Internet and required by any BPsec implementation communicating over the terrestrial Internet.

To ensure interoperability among various implementations, all BPsec implementations MUST support at least the current IETF standards-track mandatory security context(s). As of this writing, that BCP mandatory security context is specified in [I-D.ietf-dtn-bpsec-default-sc], but the mandatory security context(s) might change over time in accordance with usual IETF processes. Such changes are likely to occur in the future if/when flaws are discovered in the applicable cryptographic algorithms, for example.

Additionally, BPsec implementations need to support the security contexts which are specified and/or used by the BP networks in which they are deployed.

If a node serves as a gateway amongst two or more networks, the BPsec implementation at that node needs to support the union of security contexts mandated in those networks.

BPsec has been designed to allow for a diversity of security contexts and for new contexts to be defined over time. The use of different security contexts does not change the BPsec protocol itself and the definition of new security contexts MUST adhere to the requirements of such contexts as presented in this section and generally in this specification.

Implementors should monitor the state of security context specifications to check for future updates and replacement.

9.2. Identification and Configuration

Security blocks uniquely identify the security context to be used in the processing of their security services. The security context for a security block MUST be uniquely identifiable and MAY use parameters for customization.

To reduce the number of security contexts used in a network, security context designers should make security contexts customizable through the definition of security context parameters. For example, a single security context could be associated with a single cipher suite and security context parameters could be used to configure the use of this security context with different key lengths and different key management options without needing to define separate security contexts for each possible option.

A single security context may be used in the application of more than one security service. This means that a security context identifier MAY be used with a BIB, with a BCB, or with any other BPsec-compliant security block. The definition of a security context MUST identify which security services may be used with the security context, how security context parameters are interpreted as a function of the security operation being supported, and which security results are produced for each security service.

Network operators must determine the number, type, and configuration of security contexts in a system. Networks with rapidly changing configurations may define relatively few security contexts with each context customized with multiple parameters. For networks with more stability, or an increased need for confidentiality, a larger number of contexts can be defined with each context supporting few, if any, parameters.

Security Context Examples

Context Type	Parameters	Definition
Key Exchange AES	Encrypted Key, IV	AES-GCM-256 cipher suite with provided ephemeral key encrypted with a predetermined key encryption key and clear text initialization vector.
Pre-shared Key AES	IV	AES-GCM-256 cipher suite with predetermined key and predetermined key rotation policy.
Out of Band AES	None	AES-GCM-256 cipher suite with all info predetermined.

Table 1

9.3. Authorship

Developers or implementers should consider the diverse performance and conditions of networks on which the Bundle Protocol (and therefore BPsec) will operate. Specifically, the delay and capacity of delay-tolerant networks can vary substantially. Developers should consider these conditions to better describe the conditions when those contexts will operate or exhibit vulnerability, and selection of these contexts for implementation should be made with consideration for this reality. There are key differences that may limit the opportunity for a security context to leverage existing cipher suites and technologies that have been developed for use in traditional, more reliable networks:

- o Data Lifetime: Depending on the application environment, bundles may persist on the network for extended periods of time, perhaps even years. Cryptographic algorithms should be selected to ensure protection of data against attacks for a length of time reasonable for the application.
- o One-Way Traffic: Depending on the application environment, it is possible that only a one-way connection may exist between two endpoints, or if a two-way connection does exist, the round-trip time may be extremely large. This may limit the utility of session key generation mechanisms, such as Diffie-Hellman, as a two-way handshake may not be feasible or reliable.
- o Opportunistic Access: Depending on the application environment, a given endpoint may not be guaranteed to be accessible within a certain amount of time. This may make asymmetric cryptographic architectures which rely on a key distribution center or other trust center impractical under certain conditions.

When developing security contexts for use with BPsec, the following information SHOULD be considered for inclusion in these specifications.

- o Security Context Parameters. Security contexts MUST define their parameter Ids, the data types of those parameters, and their CBOR encoding.
- o Security Results. Security contexts MUST define their security result Ids, the data types of those results, and their CBOR encoding.
- o New Canonicalizations. Security contexts may define new canonicalization algorithms as necessary.

- o Cipher-Text Size. Security contexts MUST state whether their associated cipher suites generate cipher text (to include any authentication information) that is of a different size than the input plain text.

If a security context does not wish to alter the size of the plain text it should place overflow bytes and authentication tags in security result fields.

- o Block Header Information. Security contexts SHOULD include block header information that is considered to be immutable for the block. This information MAY include the block type code, block number, CRC Type and CRC field (if present or if missing and unlikely to be added later), and possibly certain block processing control flags. Designers should input these fields as additional data for integrity protection when these fields are expected to remain unchanged over the path the block will take from the security source to the security acceptor. Security contexts considering block header information MUST describe expected behavior when these fields fail their integrity verification.
- o Handling CRC Fields. Security contexts may include algorithms that alter the contexts of their security target block, such as the case when encrypting the block-type-specific data of a target block as part of a BCB confidentiality service. Security context specifications SHOULD address how preexisting CRC-Type and CRC-Value fields be handled. For example, a BCB security context could remove the plain-text CRC value from its target upon encryption and replace or recalculate the value upon decryption.

10. Defining Other Security Blocks

Other security blocks (OSBs) may be defined and used in addition to the security blocks identified in this specification. Both the usage of BIB, BCB, and any future OSBs can co-exist within a bundle and can be considered in conformance with BPsec if each of the following requirements are met by any future identified security blocks.

- o Other security blocks (OSBs) MUST NOT reuse any enumerations identified in this specification, to include the block type codes for BIB and BCB.
- o An OSB definition MUST state whether it can be the target of a BIB or a BCB. The definition MUST also state whether the OSB can target a BIB or a BCB.
- o An OSB definition MUST provide a deterministic processing order in the event that a bundle is received containing BIBs, BCBs, and

OSBs. This processing order MUST NOT alter the BIB and BCB processing orders identified in this specification.

- o An OSB definition MUST provide a canonicalization algorithm if the default non-primary-block canonicalization algorithm cannot be used to generate a deterministic input for a cipher suite. This requirement can be waived if the OSB is defined so as to never be the security target of a BIB or a BCB.
- o An OSB definition MUST NOT require any behavior of a BPSEC-BPA that is in conflict with the behavior identified in this specification. In particular, the security processing requirements imposed by this specification must be consistent across all BPSEC-BPAs in a network.
- o The behavior of an OSB when dealing with fragmentation must be specified and MUST NOT lead to ambiguous processing states. In particular, an OSB definition should address how to receive and process an OSB in a bundle fragment that may or may not also contain its security target. An OSB definition should also address whether an OSB may be added to a bundle marked as a fragment.

Additionally, policy considerations for the management, monitoring, and configuration associated with blocks SHOULD be included in any OSB definition.

NOTE: The burden of showing compliance with processing rules is placed upon the specifications defining new security blocks and the identification of such blocks shall not, alone, require maintenance of this specification.

11. IANA Considerations

This specification includes fields requiring registries managed by IANA.

11.1. Bundle Block Types

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

Additional Entries for the Bundle Block-Type Codes Registry:

Value	Description	Reference
TBA	Block Integrity Block	This document
TBA	Block Confidentiality Block	This document

Table 2

The Bundle Block Types namespace notes whether a block type is meant for use in BP version 6, BP version 7, or both. The two block types defined in this specification are meant for use with BP version 7.

11.2. Bundle Status Report Reason Codes

This specification allocates five reason codes from the existing "Bundle Status Report Reason Codes" registry defined in [RFC6255].

Additional Entries for the Bundle Status Report Reason Codes Registry:

BP Version	Value	Description	Reference
7	TBD	Missing Security Operation	This document, Section 7.1
7	TBD	Unknown Security Operation	This document, Section 7.1
7	TBD	Unexpected Security Operation	This document, Section 7.1
7	TBD	Failed Security Operation	This document, Section 7.1
7	TBD	Conflicting Security Operation	This document, Section 7.1

11.3. Security Context Identifiers

BPsec has a Security Context Identifier field for which IANA is requested to create and maintain a new registry named "BPsec Security Context Identifiers". Initial values for this registry are given below.

The registration policy for this registry is: Specification Required.

The value range is: signed 16-bit integer.

BPsec Security Context Identifier Registry

Value	Description	Reference
< 0	Reserved	This document
0	Reserved	This document

Table 3

Negative security context identifiers are reserved for local/site-specific uses. The use of 0 as a security context identifier is for non-operational testing purposes only.

12. References

12.1. Normative References

[I-D.ietf-dtn-bpbis]

Burleigh, S., Fall, K., and E. Birrane, "Bundle Protocol Version 7", draft-ietf-dtn-bpbis-31 (work in progress), January 2021.

[I-D.ietf-dtn-bpsec-default-sc]

Birrane, E., "BPsec Default Security Contexts", draft-ietf-dtn-bpsec-default-sc-01 (work in progress), February 2021.

[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>>.

[RFC3552] Rescorla, E. and B. Korver, "Guidelines for Writing RFC Text on Security Considerations", BCP 72, RFC 3552, DOI 10.17487/RFC3552, July 2003, <<https://www.rfc-editor.org/info/rfc3552>>.

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

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

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

12.2. Informative References

- [I-D.birrane-dtn-sbsp]
 Birrane, E., Pierce-Mayer, J., and D. Iannicca,
 "Streamlined Bundle Security Protocol Specification",
 draft-birrane-dtn-sbsp-01 (work in progress), October 2015.
- [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, <<https://www.rfc-editor.org/info/rfc4838>>.
- [RFC6257] Symington, S., Farrell, S., Weiss, H., and P. Lovell, "Bundle Security Protocol Specification", RFC 6257, DOI 10.17487/RFC6257, May 2011, <<https://www.rfc-editor.org/info/rfc6257>>.

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, Angela Hennessy of the Laboratory for Telecommunications Sciences, and Amy Alford, 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

Kenneth McKeever
The Johns Hopkins University Applied
 Physics Laboratory
11100 Johns Hopkins Rd.
Laurel, MD 20723
US

Phone: +1 443 778 2237
Email: Ken.McKeever@jhuapl.edu

Delay-Tolerant Networking
Internet-Draft
Intended status: Standards Track
Expires: 9 April 2022

B. Sipos
RKF Engineering
M. Demmer
UC Berkeley
J. Ott
Aalto University
S. Perreault
6 October 2021

Delay-Tolerant Networking TCP Convergence Layer Protocol Version 4
draft-ietf-dtn-tcpclv4-28

Abstract

This document describes a TCP-based convergence layer (TCPCL) for Delay-Tolerant Networking (DTN). This version of the TCPCL protocol resolves implementation issues in the earlier TCPCL Version 3 of RFC7242 and updates to the Bundle Protocol (BP) contents, encodings, and convergence layer requirements in BP Version 7. Specifically, the TCPCLv4 uses CBOR-encoded BPv7 bundles as its service data unit being transported and provides a reliable transport of such bundles. This version of TCPCL also includes security and extensibility mechanisms.

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 9 April 2022.

Copyright Notice

Copyright (c) 2021 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	4
1.1. Scope	5
2. Requirements Language	5
2.1. Definitions Specific to the TCPCL Protocol	6
3. General Protocol Description	9
3.1. Convergence Layer Services	9
3.2. TCPCL Session Overview	12
3.3. TCPCL States and Transitions	13
3.4. PKIX Environments and CA Policy	19
3.5. Session Keeping Policies	20
3.6. Transfer Segmentation Policies	21
3.7. Example Message Exchange	22
4. Session Establishment	24
4.1. TCP Connection	24
4.2. Contact Header	25
4.3. Contact Validation and Negotiation	26
4.4. Session Security	28
4.4.1. Entity Identification	28
4.4.2. Certificate Profile for TCPCL	29
4.4.3. TLS Handshake	31
4.4.4. TLS Authentication	32
4.4.5. Policy Recommendations	34
4.4.6. Example TLS Initiation	34
4.5. Message Header	36
4.6. Session Initialization Message (SESS_INIT)	37
4.7. Session Parameter Negotiation	39
4.8. Session Extension Items	40
5. Established Session Operation	41
5.1. Upkeep and Status Messages	41
5.1.1. Session Upkeep (KEEPALIVE)	41
5.1.2. Message Rejection (MSG_REJECT)	42
5.2. Bundle Transfer	44
5.2.1. Bundle Transfer ID	45
5.2.2. Data Transmission (XFER_SEGMENT)	45
5.2.3. Data Acknowledgments (XFER_ACK)	47
5.2.4. Transfer Refusal (XFER_REFUSE)	49
5.2.5. Transfer Extension Items	51

6.	Session Termination	53
6.1.	Session Termination Message (SESS_TERM)	53
6.2.	Idle Session Shutdown	56
7.	Implementation Status	56
8.	Security Considerations	57
8.1.	Threat: Passive Leak of Node Data	57
8.2.	Threat: Passive Leak of Bundle Data	57
8.3.	Threat: TCPCL Version Downgrade	57
8.4.	Threat: Transport Security Stripping	57
8.5.	Threat: Weak TLS Configurations	58
8.6.	Threat: Untrusted End-Entity Certificate	58
8.7.	Threat: Certificate Validation Vulnerabilities	58
8.8.	Threat: Symmetric Key Limits	59
8.9.	Threat: BP Node Impersonation	59
8.10.	Threat: Denial of Service	60
8.11.	Mandatory-to-Implement TLS	61
8.12.	Alternate Uses of TLS	61
8.12.1.	TLS Without Authentication	61
8.12.2.	Non-Certificate TLS Use	61
8.13.	Predictability of Transfer IDs	62
9.	IANA Considerations	62
9.1.	Port Number	62
9.2.	Protocol Versions	63
9.3.	Session Extension Types	64
9.4.	Transfer Extension Types	64
9.5.	Message Types	65
9.6.	XFER_REFUSE Reason Codes	66
9.7.	SESS_TERM Reason Codes	67
9.8.	MSG_REJECT Reason Codes	68
9.9.	Object Identifier for PKIX Module Identifier	69
9.10.	Object Identifier for PKIX Other Name Forms	69
9.11.	Object Identifier for PKIX Extended Key Usage	70
10.	Acknowledgments	70
11.	References	70
11.1.	Normative References	70
11.2.	Informative References	72
	Appendix A. Significant changes from RFC7242	74
	Appendix B. ASN.1 Module	76
	Appendix C. Example of the BundleEID Other Name Form	78
	Authors' Addresses	78

1. Introduction

This document describes the TCP-based convergence-layer protocol for Delay-Tolerant Networking. Delay-Tolerant Networking is an end-to-end architecture providing communications in and/or through highly stressed environments, including those with intermittent connectivity, long and/or variable delays, and high bit error rates. More detailed descriptions of the rationale and capabilities of these networks can be found in "Delay-Tolerant Network Architecture" [RFC4838].

An important goal of the DTN architecture is to accommodate a wide range of networking technologies and environments. The protocol used for DTN communications is the Bundle Protocol Version 7 (BPv7) [I-D.ietf-dtn-bpbis], an application-layer protocol that is used to construct a store-and-forward overlay network. BPv7 requires the services of a "convergence-layer adapter" (CLA) to send and receive bundles using the service of some "native" link, network, or Internet protocol. This document describes one such convergence-layer adapter that uses the well-known Transmission Control Protocol (TCP). This convergence layer is referred to as TCP Convergence Layer Version 4 (TCPCLv4). For the remainder of this document, the abbreviation "BP" without the version suffix refers to BPv7. For the remainder of this document, the abbreviation "TCPCL" without the version suffix refers to TCPCLv4.

The locations of the TCPCL and the BP in the Internet model protocol stack (described in [RFC1122]) are shown in Figure 1. In particular, when BP is using TCP as its bearer with TCPCL as its convergence layer, both BP and TCPCL reside at the application layer of the Internet model.

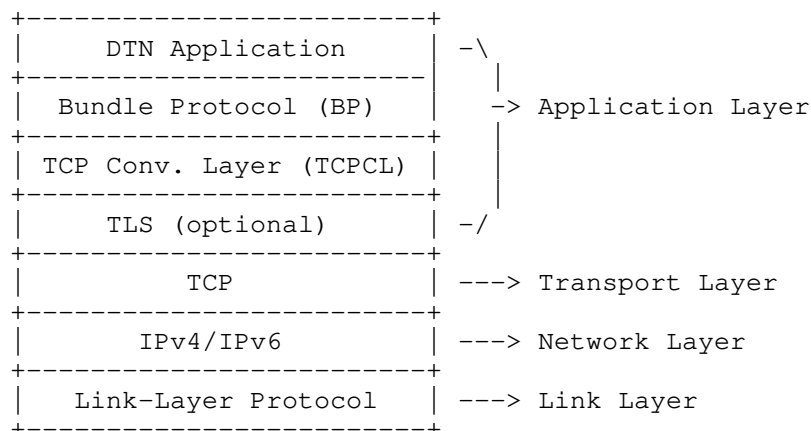


Figure 1: The Locations of the Bundle Protocol and the TCP Convergence-Layer Protocol above the Internet Protocol Stack

1.1. Scope

This document describes the format of the protocol data units passed between entities participating in TCPCL communications. This document does not address:

- * The format of protocol data units of the Bundle Protocol, as those are defined elsewhere in [I-D.ietf-dtn-bpbis]. This includes the concept of bundle fragmentation or bundle encapsulation. The TCPCL transfers bundles as opaque data blocks.
- * Mechanisms for locating or identifying other bundle entities (peers) within a network or across an internet. The mapping of Node ID to potential convergence layer (CL) protocol and network address is left to implementation and configuration of the BP Agent and its various potential routing strategies. The mapping of DNS name and/or address to a choice of end-entity certificate to authenticate a node to its peers.
- * Logic for routing bundles along a path toward a bundle's endpoint. This CL protocol is involved only in transporting bundles between adjacent entities in a routing sequence.
- * Policies or mechanisms for issuing Public Key Infrastructure Using X.509 (PKIX) certificates; provisioning, deploying, or accessing certificates and private keys; deploying or accessing certificate revocation lists (CRLs); or configuring security parameters on an individual entity or across a network.
- * Uses of TLS which are not based on PKIX certificate authentication (see Section 8.12.2) or in which authentication of both entities is not possible (see Section 8.12.1).

Any TCPCL implementation requires a BP agent to perform those above listed functions in order to perform end-to-end bundle delivery.

2. Requirements Language

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

2.1. Definitions Specific to the TCPCL Protocol

This section contains definitions specific to the TCPCL protocol.

Network Byte Order: Most significant byte first, a.k.a., big endian. All of the integer encodings in this protocol SHALL be transmitted in network byte order.

TCPCL Entity: This is the notional TCPCL application that initiates TCPCL sessions. This design, implementation, configuration, and specific behavior of such an entity is outside of the scope of this document. However, the concept of an entity has utility within the scope of this document as the container and initiator of TCPCL sessions. The relationship between a TCPCL entity and TCPCL sessions is defined as follows:

- * A TCPCL Entity MAY actively initiate any number of TCPCL Sessions and should do so whenever the entity is the initial transmitter of information to another entity in the network.
- * A TCPCL Entity MAY support zero or more passive listening elements that listen for connection requests from other TCPCL Entities operating on other entities in the network.
- * A TCPCL Entity MAY passively initiate any number of TCPCL Sessions from requests received by its passive listening element(s) if the entity uses such elements.

These relationships are illustrated in Figure 2. For most TCPCL behavior within a session, the two entities are symmetric and there is no protocol distinction between them. Some specific behavior, particularly during session establishment, distinguishes between the active entity and the passive entity. For the remainder of this document, the term "entity" without the prefix "TCPCL" refers to a TCPCL entity.

TCP Connection: The term Connection in this specification exclusively refers to a TCP connection and any and all behaviors, sessions, and other states associated with that TCP connection.

TCPCL Session: A TCPCL session (as opposed to a TCP connection) is a TCPCL communication relationship between two TCPCL entities. A TCPCL session operates within a single underlying TCP connection and the lifetime of a TCPCL session is bound to the lifetime of that TCP connection. A TCPCL session is terminated when the TCP connection ends, due either to one or both entities actively closing the TCP connection or due to network errors causing a failure of the TCP connection. Within a single TCPCL session

there are two possible transfer streams; one in each direction, with one stream from each entity being the outbound stream and the other being the inbound stream (see Figure 3). From the perspective of a TCPCL session, the two transfer streams do not logically interact with each other. The streams do operate over the same TCP connection and between the same BP agents, so there are logical relationships at those layers (message and bundle interleaving respectively). For the remainder of this document, the term "session" without the prefix "TCPCL" refers to a TCPCL session.

Session parameters: These are a set of values used to affect the operation of the TCPCL for a given session. The manner in which these parameters are conveyed to the bundle entity and thereby to the TCPCL is implementation dependent. However, the mechanism by which two entities exchange and negotiate the values to be used for a given session is described in Section 4.3.

Transfer Stream: A Transfer stream is a uni-directional user-data path within a TCPCL Session. Transfers sent over a transfer stream are serialized, meaning that one transfer must complete its transmission prior to another transfer being started over the same transfer stream. At the stream layer there is no logical relationship between transfers in that stream; it's only within the BP agent that transfers are fully decoded as bundles. Each uni-directional stream has a single sender entity and a single receiver entity.

Transfer: This refers to the procedures and mechanisms for conveyance of an individual bundle from one node to another. Each transfer within TCPCL is identified by a Transfer ID number which is guaranteed to be unique only to a single direction within a single Session.

Transfer Segment: A subset of a transfer of user data being communicated over a transfer stream.

Idle Session: A TCPCL session is idle while there is no transmission in-progress in either direction. While idle, the only messages being transmitted or received are KEEPALIVE messages.

Live Session: A TCPCL session is live while there is a transmission in-progress in either direction.

Reason Codes: The TCPCL uses numeric codes to encode specific reasons for individual failure/error message types.

The relationship between connections, sessions, and streams is shown in Figure 3.

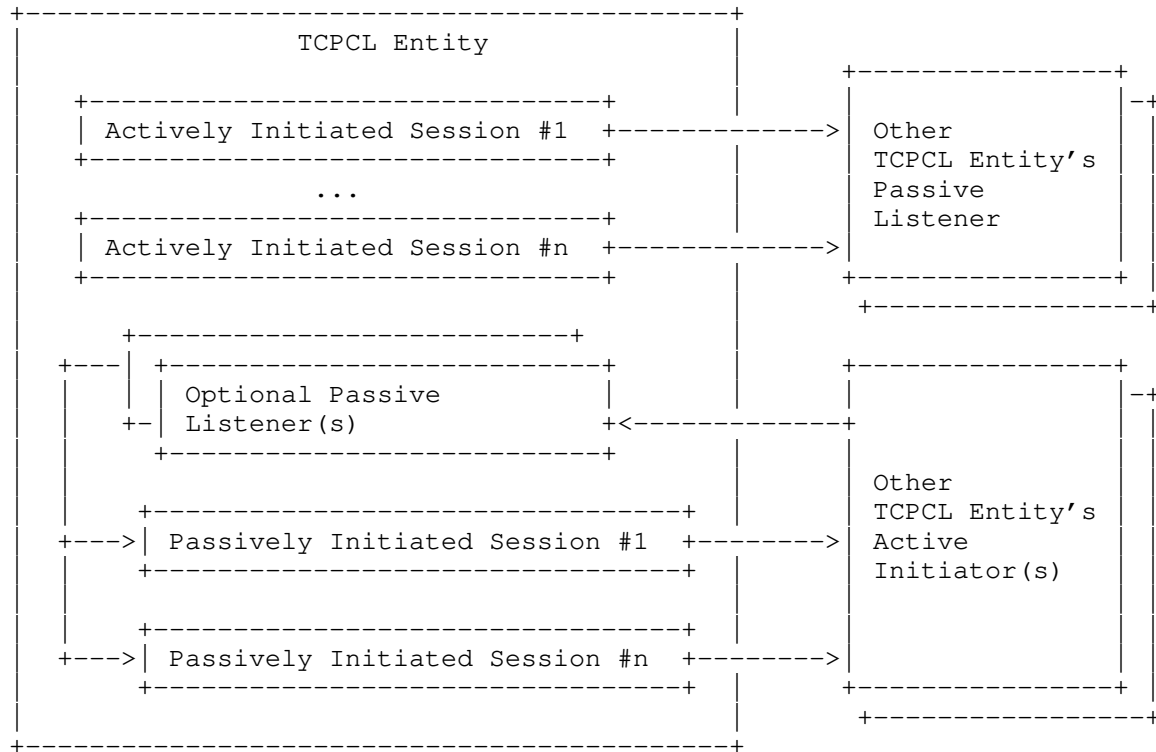


Figure 2: The relationships between TCPCL entities

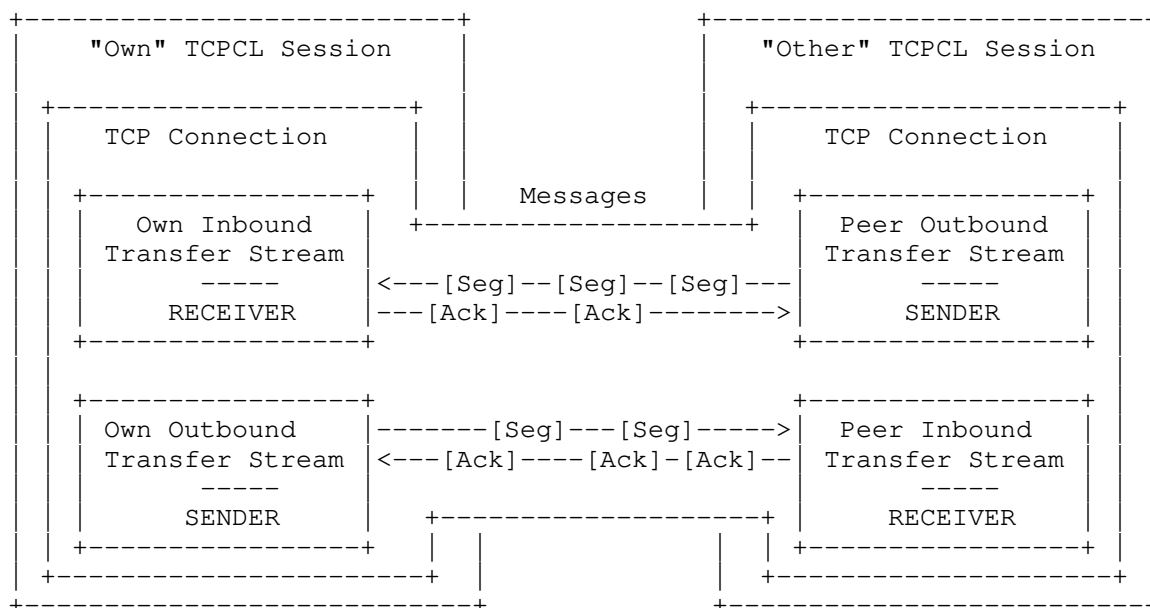


Figure 3: The relationship within a TCPCL Session of its two streams

3. General Protocol Description

The service of this protocol is the transmission of DTN bundles via the Transmission Control Protocol (TCP). This document specifies the encapsulation of bundles, procedures for TCP setup and teardown, and a set of messages and entity requirements. The general operation of the protocol is as follows.

3.1. Convergence Layer Services

This version of the TCPCL provides the following services to support the overlaying Bundle Protocol agent. In all cases, this is not an API definition but a logical description of how the CL can interact with the BP agent. Each of these interactions can be associated with any number of additional metadata items as necessary to support the operation of the CL or BP agent.

Attempt Session: The TCPCL allows a BP agent to preemptively attempt to establish a TCPCL session with a peer entity. Each session attempt can send a different set of session negotiation parameters as directed by the BP agent.

Terminate Session: The TCPCL allows a BP agent to preemptively

terminate an established TCPCL session with a peer entity. The terminate request is on a per-session basis.

Session State Changed: The TCPCL entity indicates to the BP agent when the session state changes. The top-level session states indicated are:

Connecting: A TCP connection is being established. This state only applies to the active entity.

Contact Negotiating: A TCP connection has been made (as either active or passive entity) and contact negotiation has begun.

Session Negotiating: Contact negotiation has been completed (including possible TLS use) and session negotiation has begun.

Established: The session has been fully established and is ready for its first transfer. When the session is established, the peer Node ID (along with indication of whether or not it was authenticated) and the negotiated session parameters (see Section 4.7) are also communicated to the BP agent.

Ending: The entity sent SESS_TERM message and is in the ending state.

Terminated: The session has finished normal termination sequencing.

Failed: The session ended without normal termination sequencing.

Session Idle Changed: The TCPCL entity indicates to the BP agent when the live/idle sub-state of the session changes. This occurs only when the top-level session state is "Established". The session transitions from Idle to Live at the at the start of a transfer in either transfer stream; the session transitions from Live to Idle at the end of a transfer when the other transfer stream does not have an ongoing transfer. Because TCPCL transmits serially over a TCP connection it suffers from "head of queue blocking," so a transfer in either direction can block an immediate start of a new transfer in the session.

Begin Transmission: The principal purpose of the TCPCL is to allow a BP agent to transmit bundle data over an established TCPCL session. Transmission request is on a per-session basis and the CL does not necessarily perform any per-session or inter-session queueing. Any queueing of transmissions is the obligation of the BP agent.

Transmission Success: The TCPCL entity indicates to the BP agent when a bundle has been fully transferred to a peer entity.

Transmission Intermediate Progress: The TCPCL entity indicates to the BP agent on intermediate progress of transfer to a peer entity. This intermediate progress is at the granularity of each transferred segment.

Transmission Failure: The TCPCL entity indicates to the BP agent on certain reasons for bundle transmission failure, notably when the peer entity rejects the bundle or when a TCPCL session ends before transfer success. The TCPCL itself does not have a notion of transfer timeout.

Reception Initialized: The TCPCL entity indicates to the receiving BP agent just before any transmission data is sent. This corresponds to reception of the XFER_SEGMENT message with the START flag of 1.

Interrupt Reception: The TCPCL entity allows a BP agent to interrupt an individual transfer before it has fully completed (successfully or not). Interruption can occur any time after the reception is initialized.

Reception Success: The TCPCL entity indicates to the BP agent when a bundle has been fully transferred from a peer entity.

Reception Intermediate Progress: The TCPCL entity indicates to the BP agent on intermediate progress of transfer from the peer entity. This intermediate progress is at the granularity of each transferred segment. Intermediate reception indication allows a BP agent the chance to inspect bundle header contents before the entire bundle is available, and thus supports the "Reception Interruption" capability.

Reception Failure: The TCPCL entity indicates to the BP agent on certain reasons for reception failure, notably when the local entity rejects an attempted transfer for some local policy reason or when a TCPCL session ends before transfer success. The TCPCL itself does not have a notion of transfer timeout.

3.2. TCPCL Session Overview

First, one entity establishes a TCPCL session to the other by initiating a TCP connection in accordance with [RFC0793]. After setup of the TCP connection is complete, an initial Contact Header is exchanged in both directions to establish a shared TCPCL version and negotiate the use of TLS security (as described in Section 4). Once contact negotiation is complete, TCPCL messaging is available and the session negotiation is used to set parameters of the TCPCL session. One of these parameters is a Node ID that each TCPCL Entity is acting as. This is used to assist in routing and forwarding messages by the BP Agent and is part of the authentication capability provided by TLS.

Once negotiated, the parameters of a TCPCL session cannot change and if there is a desire by either peer to transfer data under different parameters then a new session must be established. This makes CL logic simpler but relies on the assumption that establishing a TCP connection is lightweight enough that TCP connection overhead is negligible compared to TCPCL data sizes.

Once the TCPCL session is established and configured in this way, bundles can be transferred in either direction. Each transfer is performed by segmenting the transfer data into one or more XFER_SEGMENT messages. Multiple bundles can be transmitted consecutively in a single direction on a single TCPCL connection. Segments from different bundles are never interleaved. Bundle interleaving can be accomplished by fragmentation at the BP layer or by establishing multiple TCPCL sessions between the same peers. There is no fundamental limit on the number of TCPCL sessions which a single entity can establish beyond the limit imposed by the number of available (ephemeral) TCP ports of the active entity.

A feature of this protocol is for the receiving entity to send acknowledgment (XFER_ACK) messages as bundle data segments arrive. The rationale behind these acknowledgments is to enable the transmitting entity to determine how much of the bundle has been received, so that in case the session is interrupted, it can perform reactive fragmentation to avoid re-sending the already transmitted part of the bundle. In addition, there is no explicit flow control on the TCPCL layer.

A TCPCL receiver can interrupt the transmission of a bundle at any point in time by replying with a XFER_REFUSE message, which causes the sender to stop transmission of the associated bundle (if it hasn't already finished transmission). Note: This enables a cross-layer optimization in that it allows a receiver that detects that it already has received a certain bundle to interrupt transmission as early as possible and thus save transmission capacity for other bundles.

For sessions that are idle, a KEEPALIVE message is sent at a negotiated interval. This is used to convey entity live-ness information during otherwise message-less time intervals.

A SESS_TERM message is used to initiate the ending of a TCPCL session (see Section 6.1). During termination sequencing, in-progress transfers can be completed but no new transfers can be initiated. A SESS_TERM message can also be used to refuse a session setup by a peer (see Section 4.3). Regardless of the reason, session termination is initiated by one of the entities and responded-to by the other as illustrated by Figure 13 and Figure 14. Even when there are no transfers queued or in-progress, the session termination procedure allows each entity to distinguish between a clean end to a session and the TCP connection being closed because of some underlying network issue.

Once a session is established, TCPCL is a symmetric protocol between the peers. Both sides can start sending data segments in a session, and one side's bundle transfer does not have to complete before the other side can start sending data segments on its own. Hence, the protocol allows for a bi-directional mode of communication. Note that in the case of concurrent bidirectional transmission, acknowledgment segments MAY be interleaved with data segments.

3.3. TCPCL States and Transitions

The states of a normal TCPCL session (i.e., without session failures) are indicated in Figure 4.

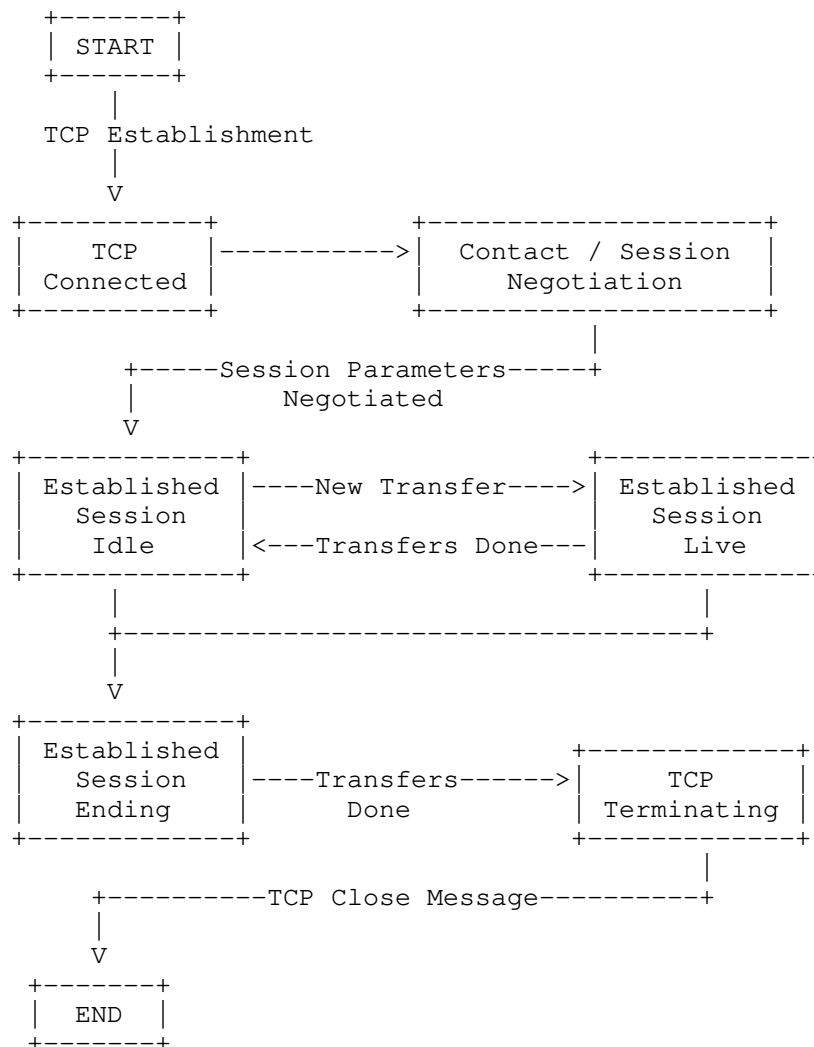


Figure 4: Top-level states of a TCPCL session

Notes on Established Session states:

Session "Live" means transmitting or receiving over a transfer stream.

Session "Idle" means no transmission/reception over a transfer stream.

Session "Ending" means no new transfers will be allowed.

Contact negotiation involves exchanging a Contact Header (CH) in both directions and deriving a negotiated state from the two headers. The contact negotiation sequencing is performed either as the active or passive entity, and is illustrated in Figure 5 and Figure 6 respectively which both share the data validation and negotiation of the Processing of Contact Header "[PCH]" activity of Figure 7 and the "[TCPCLOSE]" activity which indicates TCP connection close. Successful negotiation results in one of the Session Initiation "[SI]" activities being performed. To avoid data loss, a Session Termination "[ST]" exchange allows cleanly finishing transfers before a session is ended.

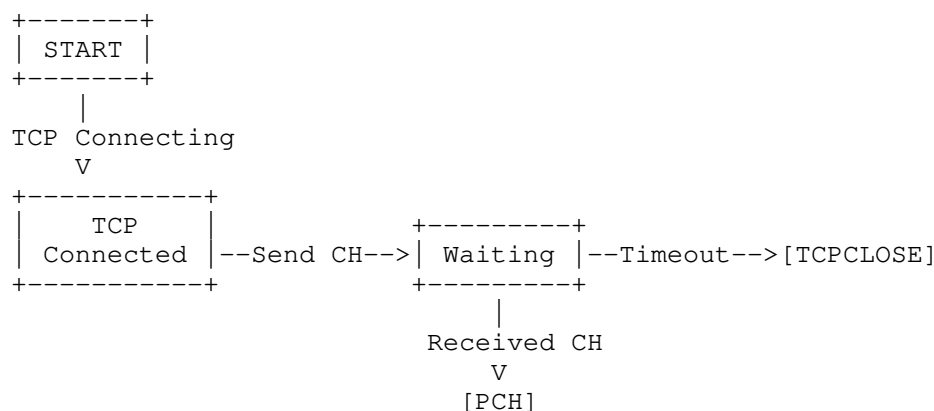


Figure 5: Contact Initiation as Active Entity

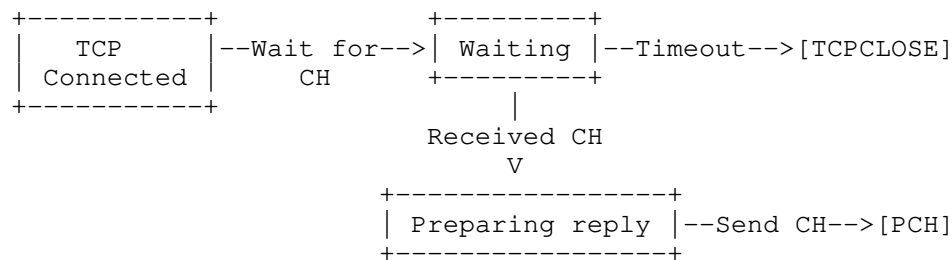


Figure 6: Contact Initiation as Passive Entity

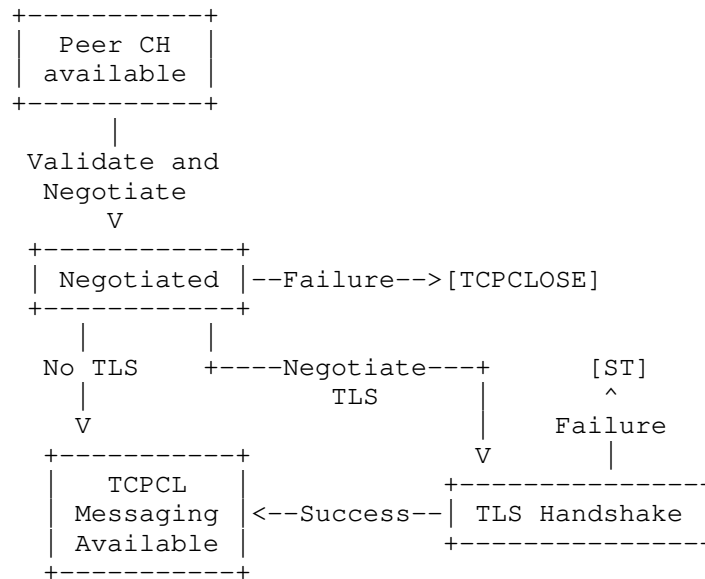


Figure 7: Processing of Contact Header [PCH]

Session negotiation involves exchanging a session initialization (SESS_INIT) message in both directions and deriving a negotiated state from the two messages. The session negotiation sequencing is performed either as the active or passive entity, and is illustrated in Figure 8 and Figure 9 respectively which both share the data validation and negotiation of Figure 10. The validation here includes certificate validation and authentication when TLS is used for the session.

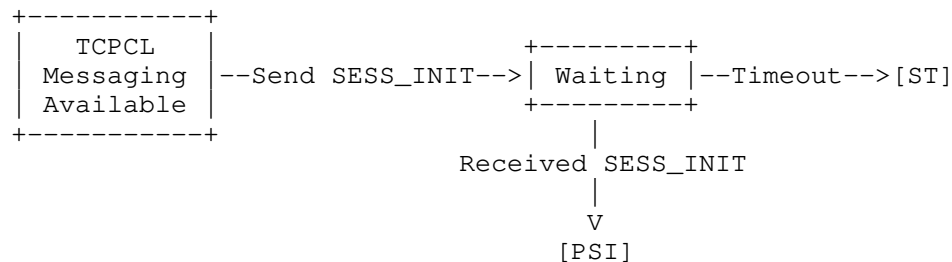


Figure 8: Session Initiation [SI] as Active Entity

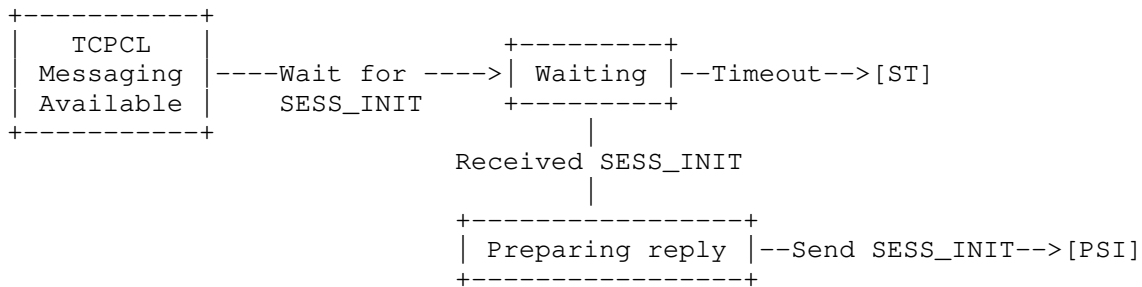


Figure 9: Session Initiation [SI] as Passive Entity

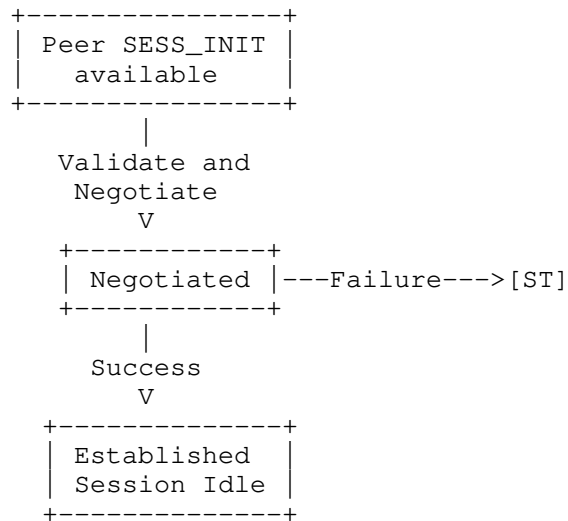


Figure 10: Processing of Session Initiation [PSI]

Transfers can occur after a session is established and it's not in the Ending state. Each transfer occurs within a single logical transfer stream between a sender and a receiver, as illustrated in Figure 11 and Figure 12 respectively.

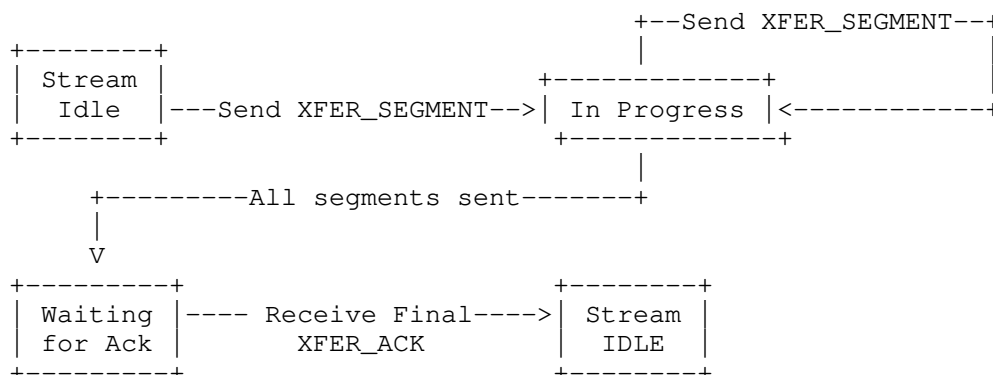


Figure 11: Transfer sender states

Notes on transfer sending:

Pipelining of transfers can occur when the sending entity begins a new transfer while in the "Waiting for Ack" state.

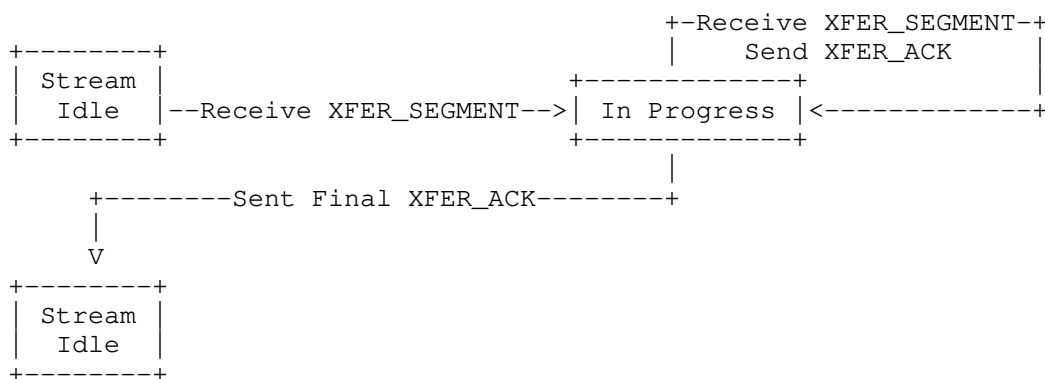


Figure 12: Transfer receiver states

Session termination involves one entity initiating the termination of the session and the other entity acknowledging the termination. For either entity, it is the sending of the SESS_TERM message which transitions the session to the Ending substate. While a session is in the Ending state only in-progress transfers can be completed and no new transfers can be started.

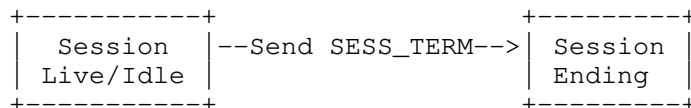


Figure 13: Session Termination [ST] from the Initiator

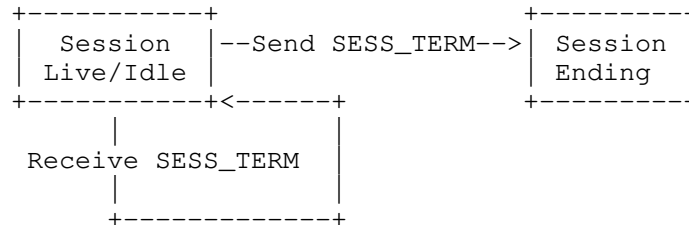


Figure 14: Session Termination [ST] from the Responder

3.4. PKIX Environments and CA Policy

This specification gives requirements about how to use PKIX certificates issued by a Certificate Authority (CA), but does not define any mechanisms for how those certificates come to be. The requirements about TCPCL certificate use are broad to support two quite different PKIX environments:

DTN-Aware CAs: In the ideal case, the CA(s) issuing certificates for TCPCL entities are aware of the end use of the certificate, have a mechanism for verifying ownership of a Node ID, and are issuing certificates directly for that Node ID. In this environment, the ability to authenticate a peer entity Node ID directly avoids the need to authenticate a network name or address and then implicitly trust Node ID of the peer. The TCPCL authenticates the Node ID whenever possible and this is preferred over lower-level PKIX identities.

DTN-Ignorant CAs: It is expected that Internet-scale "public" CAs will continue to focus on DNS names as the preferred PKIX identifier. There are large infrastructures already in-place for managing network-level authentication and protocols to manage identity verification in those environments [RFC8555]. The TCPCL allows for this type of environment by authenticating a lower-level identifier for a peer and requiring the entity to trust that the Node ID given by the peer (during session initialization) is valid. This situation is not ideal, as it allows vulnerabilities described in Section 8.9, but still provides some amount of mutual authentication to take place for a TCPCL session.

Even within a single TCPCL session, each entity may operate within different PKI environments and with different identifier limitations. The requirements related to identifiers in a PKIX certificate are in Section 4.4.1.

It is important for interoperability that a TCPCL entity have its own security policy tailored to accommodate the peers with which it is expected to operate. Some security policy recommendations are given in Section 4.4.5 but these are meant as a starting point for tailoring. A strict TLS security policy is appropriate for a private network with a single shared CA. Operation on the Internet (such as inter-site BP gateways) could trade more lax TCPCL security with the use of encrypted bundle encapsulation [I-D.ietf-dtn-bibect] to ensure strong bundle security.

By using the Server Name Indication (SNI) DNS name (see Section 4.4.3) a single passive entity can act as a convergence layer for multiple BP agents with distinct Node IDs. When this "virtual host" behavior is used, the DNS name is used as the indication of which BP Node the active entity is attempting to communicate with. A virtual host CL entity can be authenticated by a certificate containing all of the DNS names and/or Node IDs being hosted or by several certificates each authenticating a single DNS name and/or Node ID, using the SNI value from the peer to select which certificate to use. The logic for mapping an SNI DNS name to an end-entity certificate is an implementation matter, and can involve correlating DNS name with Node ID or other certificate attributes.

3.5. Session Keeping Policies

This specification gives requirements about how to initiate, sustain, and terminate a TCPCL session but does not impose any requirements on how sessions need to be managed by a BP agent. It is a network administration matter to determine an appropriate session keeping policy, but guidance given here can be used to steer policy toward performance goals.

Persistent Session: This policy preemptively establishes a single session to known entities in the network and keeps the session active using KEEPALIVES. Benefits of this policy include reducing the total amount of TCP data needing to be exchanged for a set of transfers (assuming KEEPALIVE size is significantly smaller than transfer size), and allowing the session state to indicate peer connectivity. Drawbacks include wasted network resources when a session is mostly idle or when the network connectivity is inconsistent (which requires re-establishing failed sessions), and potential queueing issues when multiple transfers are requested simultaneously. This policy assumes that there is agreement between pairs of entities as to which of the peers will initiate sessions; if there is no such agreement, there is potential for duplicate sessions to be established between peers.

Ephemeral Sessions: This policy only establishes a session when an

outgoing transfer is needed to be sent. Benefits of this policy include not wasting network resources on sessions which are idle for long periods of time, and avoids queueing issues of a persistent session. Drawbacks include the TCP and TLS overhead of establish a new session for each transfer. This policy assumes that each entity can function in a passive role to listen for session requests from any peer which needs to send a transfer; when that is not the case the Polling behavior below needs to happen. This policy can be augmented to keep the session established as long as any transfers are queued.

Active-Only Polling Sessions: When naming and/or addressing of one entity is variable (i.e. dynamically assigned IP address or domain name) or when firewall or routing rules prevent incoming TCP connections, that entity can only function in the active role. In these cases, sessions also need to be established when an incoming transfer is expected from a peer or based on a periodic schedule. This polling behavior causes inefficiencies compared to as-needed ephemeral sessions.

Many other policies can be established in a TCPCL network between the two extremes of single persistent sessions and only ephemeral sessions. Different policies can be applied to each peer entity and to each bundle as it needs to be transferred (e.g for quality of service). Additionally, future session extension types can apply further nuance to session policies and policy negotiation.

3.6. Transfer Segmentation Policies

Each TCPCL session allows a negotiated transfer segmentation policy to be applied in each transfer direction. A receiving entity can set the Segment MRU in its SESS_INIT message to determine the largest acceptable segment size, and a transmitting entity can segment a transfer into any sizes smaller than the receiver's Segment MRU. It is a network administration matter to determine an appropriate segmentation policy for entities operating TCPCL, but guidance given here can be used to steer policy toward performance goals. It is also advised to consider the Segment MRU in relation to chunking/packetization performed by TLS, TCP, and any intermediate network-layer nodes.

Minimum Overhead: For a simple network expected to exchange relatively small bundles, the Segment MRU can be set to be identical to the Transfer MRU which indicates that all transfers can be sent with a single data segment (i.e., no actual segmentation). If the network is closed and all transmitters are known to follow a single-segment transfer policy, then receivers can avoid the necessity of segment reassembly. Because this CL

operates over a TCP stream, which suffers from a form of head-of-queue blocking between messages, while one entity is transmitting a single XFER_SEGMENT message it is not able to transmit any XFER_ACK or XFER_REFUSE for any associated received transfers.

Predictable Message Sizing: In situations where the maximum message size is desired to be well-controlled, the Segment MRU can be set to the largest acceptable size (the message size less XFER_SEGMENT header size) and transmitters can always segment a transfer into maximum-size chunks no larger than the Segment MRU. This guarantees that any single XFER_SEGMENT will not monopolize the TCP stream for too long, which would prevent outgoing XFER_ACK and XFER_REFUSE associated with received transfers.

Dynamic Segmentation: Even after negotiation of a Segment MRU for each receiving entity, the actual transfer segmentation only needs to guarantee that any individual segment is no larger than that MRU. In a situation where TCP throughput is dynamic, the transfer segmentation size can also be dynamic in order to control message transmission duration.

Many other policies can be established in a TCPCL network between the two extremes of minimum overhead (large MRU, single-segment) and predictable message sizing (small MRU, highly segmented). Different policies can be applied to each transfer stream to and from any particular entity. Additionally, future session extension and transfer extension types can apply further nuance to transfer policies and policy negotiation.

3.7. Example Message Exchange

The following figure depicts the protocol exchange for a simple session, showing the session establishment and the transmission of a single bundle split into three data segments (of lengths "L1", "L2", and "L3") from Entity A to Entity B.

Note that the sending entity can transmit multiple XFER_SEGMENT messages without waiting for the corresponding XFER_ACK responses. This enables pipelining of messages on a transfer stream. Although this example only demonstrates a single bundle transmission, it is also possible to pipeline multiple XFER_SEGMENT messages for different bundles without necessarily waiting for XFER_ACK messages to be returned for each one. However, interleaving data segments from different bundles is not allowed.

No errors or rejections are shown in this example.

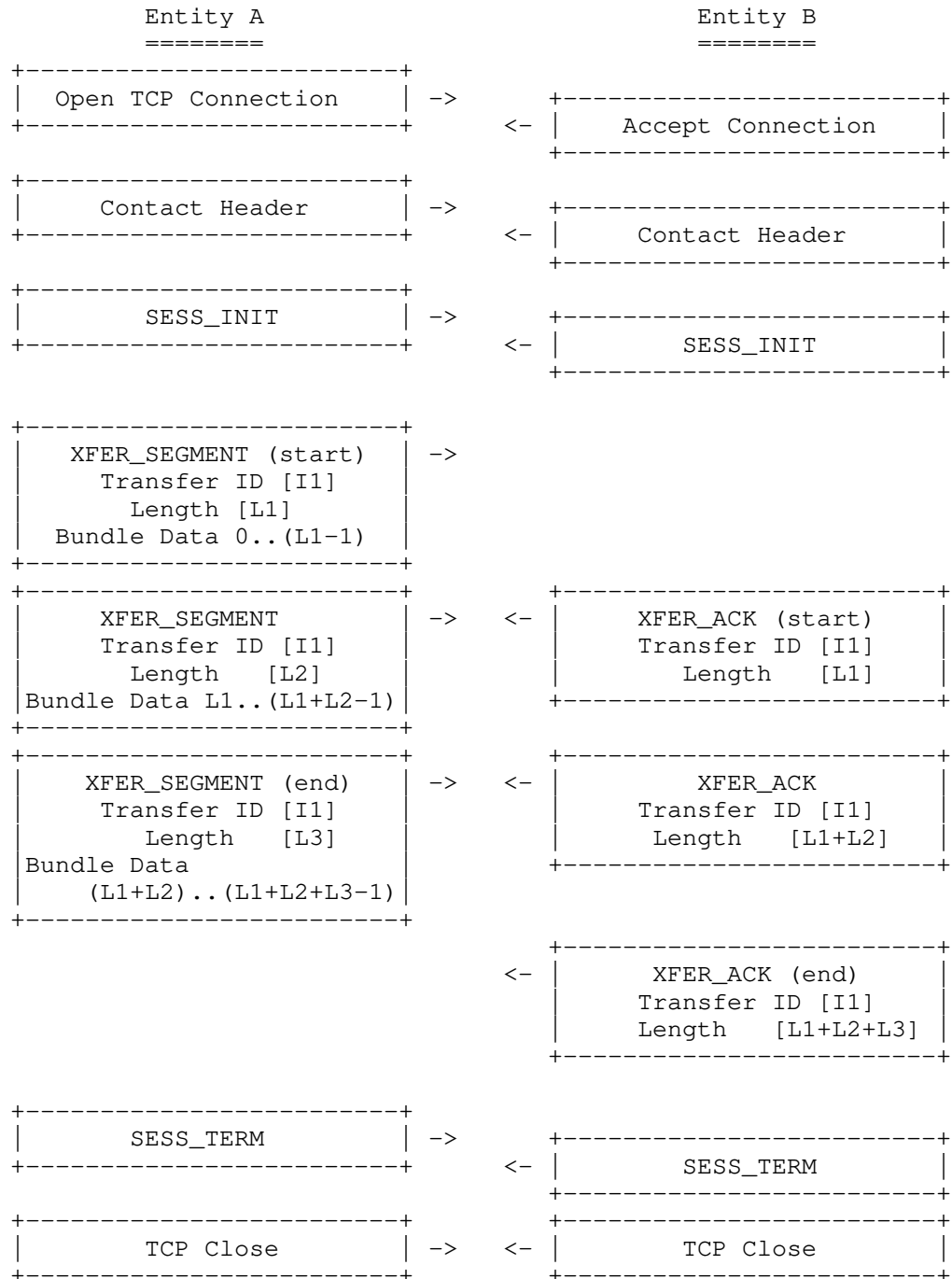


Figure 15: An example of the flow of protocol messages on a single TCP Session between two entities

4. Session Establishment

For bundle transmissions to occur using the TCPCL, a TCPCL session MUST first be established between communicating entities. It is up to the implementation to decide how and when session setup is triggered. For example, some sessions can be opened proactively and maintained for as long as is possible given the network conditions, while other sessions are opened only when there is a bundle that is queued for transmission and the routing algorithm selects a certain next-hop node.

4.1. TCP Connection

To establish a TCPCL session, an entity MUST first establish a TCP connection with the intended peer entity, typically by using the services provided by the operating system. Destination port number 4556 has been assigned by IANA as the Registered Port number for the TCP convergence layer. Other destination port numbers MAY be used per local configuration. Determining a peer's destination port number (if different from the registered TCPCL port number) is up to the implementation. Any source port number MAY be used for TCPCL sessions. Typically an operating system assigned number in the TCP Ephemeral range (49152-65535) is used.

If the entity is unable to establish a TCP connection for any reason, then it is an implementation matter to determine how to handle the connection failure. An entity MAY decide to re-attempt to establish the connection. If it does so, it MUST NOT overwhelm its target with repeated connection attempts. Therefore, the entity MUST NOT retry the connection setup earlier than some delay time from the last attempt, and it SHOULD use a (binary) exponential back-off mechanism to increase this delay in case of repeated failures. The upper limit on a re-attempt back-off is implementation defined but SHOULD be no longer than one minute (60 seconds) before signaling to the BP agent that a connection cannot be made.

Once a TCP connection is established, the active entity SHALL immediately transmit its Contact Header. Once a TCP connection is established, the passive entity SHALL wait for the peer's Contact Header. If the passive entity does not receive a Contact Header after some implementation-defined time duration after TCP connection is established, the entity SHALL close the TCP connection. Entities SHOULD choose a Contact Header reception timeout interval no longer than one minute (60 seconds). Upon reception of a Contact Header, the passive entity SHALL transmit its Contact Header. The ordering

of the Contact Header exchange allows the passive entity to avoid allocating resources to a potential TCPCL session until after a valid Contact Header has been received from the active entity. This ordering also allows the passive peer to adapt to alternate TCPCL protocol versions.

The format of the Contact Header is described in Section 4.2. Because the TCPCL protocol version in use is part of the initial Contact Header, entities using TCPCL version 4 can coexist on a network with entities using earlier TCPCL versions (with some negotiation needed for interoperation as described in Section 4.3).

Within this specification when an entity is said to "close" a TCP connection the entity SHALL use the TCP FIN mechanism and not the RST mechanism. Either mechanism, however, when received will cause a TCP connection to become closed.

4.2. Contact Header

This section describes the format of the Contact Header and the meaning of its fields.

If the entity is configured to enable exchanging messages according to TLS 1.3 [RFC8446] or any successors which are compatible with that TLS ClientHello, the the CAN_TLS flag within its Contact Header SHALL be set to 1. The RECOMMENDED policy is to enable TLS for all sessions, even if security policy does not allow or require authentication. This follows the opportunistic security model of [RFC7435], though an active attacker could interfere with the exchange in such cases (see Section 8.4).

Upon receipt of the Contact Header, both entities perform the validation and negotiation procedures defined in Section 4.3. After receiving the Contact Header from the other entity, either entity MAY refuse the session by sending a SESS_TERM message with an appropriate reason code.

The format for the Contact Header is as follows:

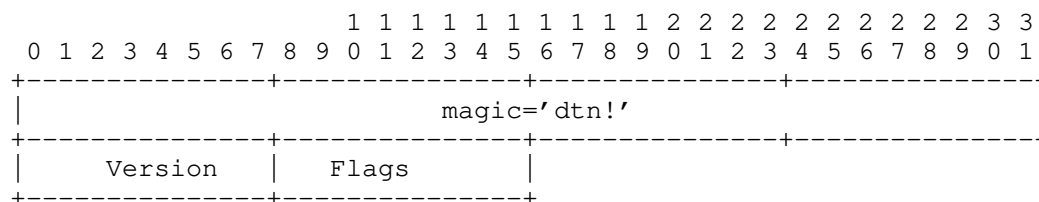


Figure 16: Contact Header Format

See Section 4.3 for details on the use of each of these Contact Header fields.

The fields of the Contact Header are:

magic: A four-octet field that always contains the octet sequence 0x64 0x74 0x6E 0x21, i.e., the text string "dtn!" in US-ASCII (and UTF-8).

Version: A one-octet field value containing the value 4 (current version of the TCPCL).

Flags: A one-octet field of single-bit flags, interpreted according to the descriptions in Table 1. All reserved header flag bits SHALL be set to 0 by the sender. All reserved header flag bits SHALL be ignored by the receiver.

Name	Code	Description
CAN_TLS	0x01	If bit is set, indicates that the sending peer has enabled TLS security.
Reserved	others	

Table 1: Contact Header Flags

4.3. Contact Validation and Negotiation

Upon reception of the Contact Header, each entity follows the following procedures to ensure the validity of the TCPCL session and to negotiate values for the session parameters.

If the magic string is not present or is not valid, the connection MUST be terminated. The intent of the magic string is to provide some protection against an inadvertent TCP connection by a different protocol than the one described in this document. To prevent a flood of repeated connections from a misconfigured application, a passive entity MAY deny new TCP connections from a specific peer address for a period of time after one or more connections fail to provide a decodable Contact Header.

The first negotiation is on the TCPCL protocol version to use. The active entity always sends its Contact Header first and waits for a response from the passive entity. During contact initiation, the active TCPCL entity SHALL send the highest TCPCL protocol version on a first session attempt for a TCPCL peer. If the active entity

receives a Contact Header with a lower protocol version than the one sent earlier on the TCP connection, the TCP connection SHALL be closed. If the active entity receives a SESS_TERM message with reason of "Version Mismatch", that entity MAY attempt further TCPCL sessions with the peer using earlier protocol version numbers in decreasing order. Managing multi-TCPCL-session state such as this is an implementation matter.

If the passive entity receives a Contact Header containing a version that is not a version of the TCPCL that the entity implements, then the entity SHALL send its Contact Header and immediately terminate the session with a reason code of "Version mismatch". If the passive entity receives a Contact Header with a version that is lower than the latest version of the protocol that the entity implements, the entity MAY either terminate the session (with a reason code of "Version mismatch") or adapt its operation to conform to the older version of the protocol. The decision of version fall-back is an implementation matter.

The negotiated contact parameters defined by this specification are described in the following paragraphs.

TCPCL Version: Both Contact Headers of a successful contact negotiation have identical TCPCL Version numbers as described above. Only upon response of a Contact Header from the passive entity is the TCPCL protocol version established and session negotiation begun.

Enable TLS: Negotiation of the Enable TLS parameter is performed by taking the logical AND of the two Contact Headers' CAN_TLS flags. A local security policy is then applied to determine if the negotiated value of Enable TLS is acceptable. It can be a reasonable security policy to require or disallow the use of TLS depending upon the desired network flows. The RECOMMENDED policy is to require TLS for all sessions, even if security policy does not allow or require authentication. Because this state is negotiated over an unsecured medium, there is a risk of a TLS Stripping as described in Section 8.4.

If the Enable TLS state is unacceptable, the entity SHALL terminate the session with a reason code of "Contact Failure". Note that this contact failure reason is different than a failure of TLS handshake or TLS authentication after an agreed-upon and acceptable Enable TLS state. If the negotiated Enable TLS value is true and acceptable then TLS negotiation feature (described in Section 4.4) begins immediately following the Contact Header exchange.

4.4. Session Security

This version of the TCPCL supports establishing a Transport Layer Security (TLS) session within an existing TCP connection. When TLS is used within the TCPCL it affects the entire session. Once TLS is established, there is no mechanism available to downgrade the TCPCL session to non-TLS operation.

Once established, the lifetime of a TLS connection SHALL be bound to the lifetime of the underlying TCP connection. Immediately prior to actively ending a TLS connection after TCPCL session termination, the peer which sent the original (non-reply) SESS_TERM message SHOULD follow the Closure Alert procedure of [RFC8446] to cleanly terminate the TLS connection. Because each TCPCL message is either fixed-length or self-indicates its length, the lack of a TLS Closure Alert will not cause data truncation or corruption.

Subsequent TCPCL session attempts to the same passive entity MAY attempt to use the TLS session resumption feature. There is no guarantee that the passive entity will accept the request to resume a TLS session, and the active entity cannot assume any resumption outcome.

4.4.1. Entity Identification

The TCPCL uses TLS for certificate exchange in both directions to identify each entity and to allow each entity to authenticate its peer. Each certificate can potentially identify multiple entities and there is no problem using such a certificate as long as the identifiers are sufficient to meet authentication policy (as described in later sections) for the entity which presents it.

Because the PKIX environment of each TCPCL entity are likely not controlled by the certificate end users (see Section 3.4), the TCPCL defines a prioritized list of what a certificate can identify about a TCPCL entity:

Node ID: The ideal certificate identity is the Node ID of the entity using the NODE-ID definition below. When the Node ID is identified, there is no need for any lower-level identification to be present (though it can still be present, and if so it is also validated).

DNS Name: If CA policy forbids a certificate to contain an arbitrary

NODE-ID but allows a DNS-ID to be identified then one or more stable DNS names can be identified in the certificate. The use of wildcard DNS-ID is discouraged due to the complex rules for matching and dependence on implementation support for wildcard matching (see Section 6.4.3 of [RFC6125]).

Network Address: If no stable DNS name is available but a stable network address is available and CA policy allows a certificate to contain a IPADDR-ID (as defined below) then one or more network addresses can be identified in the certificate.

This specification defines a NODE-ID of a certificate as being the subjectAltName entry of type otherName with a name form of BundleEID (see Section 4.4.2.1) and a value limited to a Node ID. An entity SHALL ignore any otherName with a name form of BundleEID and a value which is some URI other than a Node ID. The NODE-ID is similar to the URI-ID of [RFC6125] but restricted to a Node ID rather than a URI with a qualified-name authority part. Unless specified otherwise by the definition of the URI scheme being authenticated, URI matching of a NODE-ID SHALL use the URI comparison logic of [RFC3986] and scheme-based normalization of those schemes specified in [I-D.ietf-dtn-bpbis]. A URI scheme can refine this "exact match" logic with rules about how Node IDs within that scheme are to be compared with the certificate-authenticated NODE-ID.

This specification reuses the DNS-ID definition of Section 1.8 of [RFC6125], which is the subjectAltName entry of type dNSName whose value is encoded according to [RFC5280].

This specification defines a IPADDR-ID of a certificate as being the subjectAltName entry of type iPAddress whose value is encoded according to [RFC5280].

4.4.2. Certificate Profile for TCPCL

All end-entity certificates used by a TCPCL entity SHALL conform to [RFC5280], or any updates or successors to that profile. When an end-entity certificate is supplied, the full certification chain SHOULD be included unless security policy indicates that is unnecessary. An entity SHOULD omit the root CA certificate (the last item of the chain) when sending a certification chain, as the recipient already has the root CA to anchor its validation.

The TCPCL requires Version 3 certificates due to the extensions used by this profile. TCPCL entities SHALL reject as invalid Version 1 and Version 2 end-entity certificates.

TCPCL entities SHALL accept certificates that contain an empty Subject field or contain a Subject without a Common Name. Identity information in end-entity certificates is contained entirely in the subjectAltName extension as defined in Section 4.4.1 and below.

All end-entity and CA certificates used for TCPCL SHOULD contain both a Subject Key Identifier and an Authority Key Identifier extension in accordance with [RFC5280]. TCPCL entities SHOULD NOT rely on either a Subject Key Identifier and an Authority Key Identifier being present in any received certificate. Including key identifiers simplifies the work of an entity needing to assemble a certification chain.

Unless prohibited by CA policy, a TCPCL end-entity certificate SHALL contain a NODE-ID which authenticates the Node ID of the peer. When assigned one or more stable DNS names, a TCPCL end-entity certificate SHOULD contain DNS-ID which authenticates those (fully qualified) names. When assigned one or more stable network addresses, a TCPCL end-entity certificate MAY contain IPADDR-ID which authenticates those addresses.

When allowed by CA policy, a BPSec end-entity certificate SHOULD contain a PKIX Extended Key Usage extension in accordance with Section 4.2.1.12 of [RFC5280]. When the PKIX Extended Key Usage extension is present, it SHOULD contain a key purpose id-kp-bundleSecurity (see Section 4.4.2.1). Although not specifically required by TCPCL, some networks or TLS implementations assume the use of id-kp-clientAuth and id-kp-serverAuth are needed for, respectively, the client-side and server-side of TLS authentication. For interoperability, a TCPCL end-entity certificate MAY contain an Extended Key Usage with both id-kp-clientAuth and id-kp-serverAuth values.

When allowed by CA policy, a TCPCL end-entity certificate SHOULD contain a PKIX Key Usage extension in accordance with Section 4.2.1.3 of [RFC5280]. The PKIX Key Usage bit which is consistent with TCPCL security using TLS 1.3 is digitalSignature. The specific algorithms used during the TLS handshake will determine which of those key uses are exercised. Earlier versions of TLS can mandate use of the bits keyEncipherment or keyAgreement.

When allowed by CA policy, a TCPCL end-entity certificate SHOULD contain an Online Certificate Status Protocol (OCSP) URI within an Authority Information Access extension in accordance with Section 4.2.2.1 of [RFC5280].

4.4.2.1. PKIX OID Allocations

This document defines a PKIX Other Name Form identifier of id-on-bundleEID in Appendix B which can be used as the type-id in a subjectAltName entry of type otherName. The BundleEID value associated with otherName type-id id-on-bundleEID SHALL be a URI, encoded as an IA5String, with a scheme which is present in the IANA "Bundle Protocol URI Scheme Type" registry [IANA-BUNDLE]. Although this otherName form allows any Endpoint ID to be present, the NODE-ID defined in Section 4.4.1 limits its use to contain only a Node ID.

This document defines a PKIX Extended Key Usage key purpose id-kp-bundleSecurity in Appendix B which can be used to restrict a certificate's use. The id-kp-bundleSecurity purpose can be combined with other purposes in the same certificate.

4.4.3. TLS Handshake

The use of TLS is negotiated using the Contact Header as described in Section 4.3. After negotiating an Enable TLS parameter of true, and before any other TCPCL messages are sent within the session, the session entities SHALL begin a TLS handshake in accordance with [RFC8446]. By convention, this protocol uses the entity which initiated the underlying TCP connection (the active peer) as the "client" role of the TLS handshake request.

The TLS handshake, if it occurs, is considered to be part of the contact negotiation before the TCPCL session itself is established. Specifics about sensitive data exposure are discussed in Section 8.

The parameters within each TLS negotiation are implementation dependent but any TCPCL entity SHALL follow all recommended practices of BCP 195 [RFC7525], or any updates or successors that become part of BCP 195. Within each TLS handshake, the following requirements apply (using the rough order in which they occur):

Client Hello: When a resolved DNS name was used to establish the TCP connection, the TLS ClientHello SHOULD include a "server_name" extension in accordance with [RFC6066]. When present, the "server_name" extension SHALL contain a "HostName" value taken from the DNS name (of the passive entity) which was resolved. Note: The "HostName" in the "server_name" extension is the network name for the passive entity, not the Node ID of that entity.

Server Certificate: The passive entity SHALL supply a certificate

within the TLS handshake to allow authentication of its side of the session. The supplied end-entity certificate SHALL conform to the profile of Section 4.4.2. The passive entity MAY use the SNI DNS name to choose an appropriate server-side certificate which authenticates that DNS name.

Certificate Request: During TLS handshake, the passive entity SHALL request a client-side certificate.

Client Certificate: The active entity SHALL supply a certificate chain within the TLS handshake to allow authentication of its side of the session. The supplied end-entity certificate SHALL conform to the profile of Section 4.4.2.

If a TLS handshake cannot negotiate a TLS connection, both entities of the TCPCL session SHALL close the TCP connection. At this point the TCPCL session has not yet been established so there is no TCPCL session to terminate.

After a TLS connection is successfully established, the active entity SHALL send a SESS_INIT message to begin session negotiation. This session negotiation and all subsequent messaging are secured.

4.4.4. TLS Authentication

Using PKIX certificates exchanged during the TLS handshake, each of the entities can authenticate a peer Node ID directly or authenticate the peer DNS name or network address. The logic for handling certificates and certificate data is separated into the following phases:

1. Validating the certification path from the end-entity certificate up to a trusted root CA.
2. Validating the Extended Key Usage (EKU) and other properties of the end-entity certificate.
3. Authenticating identities from a valid end-entity certificate.
4. Applying security policy to the result of each identity type authentication.

The result of validating a peer identity (see Section 4.4.1) against one or more type of certificate claim is one of the following:

Absent: Indicating that no such claims are present in the certificate and the identity cannot be authenticated.

Success: Indicating that one or more such claims are present and at least one matches the peer identity value.

Failure: Indicating that one or more such claims are present and none match the peer identity.

4.4.4.1. Certificate Path and Purpose Validation

For any peer end-entity certificate received during TLS handshake, the entity SHALL perform the certification path validation of [RFC5280] up to one of the entity's trusted CA certificates. If enabled by local policy, the entity SHALL perform an OCSP check of each certificate providing OCSP authority information in accordance with [RFC6960]. If certificate validation fails or if security policy disallows a certificate for any reason, the entity SHALL fail the TLS handshake with a "bad_certificate" alert. Leaving out part of the certification chain can cause the entity to fail to validate a certificate if the left-out certificates are unknown to the entity (see Section 8.6).

For the end-entity peer certificate received during TLS handshake, the entity SHALL apply security policy to the Key Usage extension (if present) and Extended Key Usage extension (if present) in accordance with Section 4.2.1.12 of [RFC5280] and the profile in Section 4.4.2.

4.4.4.2. Network-Level Authentication

Either during or immediately after the TLS handshake, if required by security policy each entity SHALL validate the following certificate identifiers together in accordance with Section 6 of [RFC6125]:

- * If the active entity resolved a DNS name (of the passive entity) in order to initiate the TCP connection that DNS name SHALL be used as a DNS-ID reference identifier.
- * The IP address of the other side of the TCP connection SHALL be used as an IPADDR-ID reference identifier.

If the network-level identifiers authentication result is Failure or if the result is Absent and security policy requires an authenticated network-level identifier, the entity SHALL terminate the session (with a reason code of "Contact Failure").

4.4.4.3. Node ID Authentication

Immediately before Session Parameter Negotiation, if required by security policy each entity SHALL validate the certificate NODE-ID in accordance with Section 6 of [RFC6125] using the Node ID of the peer's SESS_INIT message as the NODE-ID reference identifier. If the NODE-ID validation result is Failure or if the result is Absent and security policy requires an authenticated Node ID, the entity SHALL terminate the session (with a reason code of "Contact Failure").

4.4.5. Policy Recommendations

A RECOMMENDED security policy is to enable the use of OCSP checking during TLS handshake. A RECOMMENDED security policy is that if an Extended Key Usage is present that it needs to contain id-kp-bundleSecurity (of Section 4.4.2.1) to be usable with TCPCL security. A RECOMMENDED security policy is to require a validated Node ID (of Section 4.4.4.3) and to ignore any network-level identifier (of Section 4.4.4.2).

This policy relies on and informs the certificate requirements in Section 4.4.3. This policy assumes that a DTN-aware CA (see Section 3.4) will only issue a certificate for a Node ID when it has verified that the private key holder actually controls the DTN node; this is needed to avoid the threat identified in Section 8.9. This policy requires that a certificate contain a NODE-ID and allows the certificate to also contain network-level identifiers. A tailored policy on a more controlled network could relax the requirement on Node ID validation and allow just network-level identifiers to authenticate a peer.

4.4.6. Example TLS Initiation

A summary of a typical TLS use is shown in the sequence in Figure 17 below. In this example the active peer terminates the session but termination can be initiated from either peer.

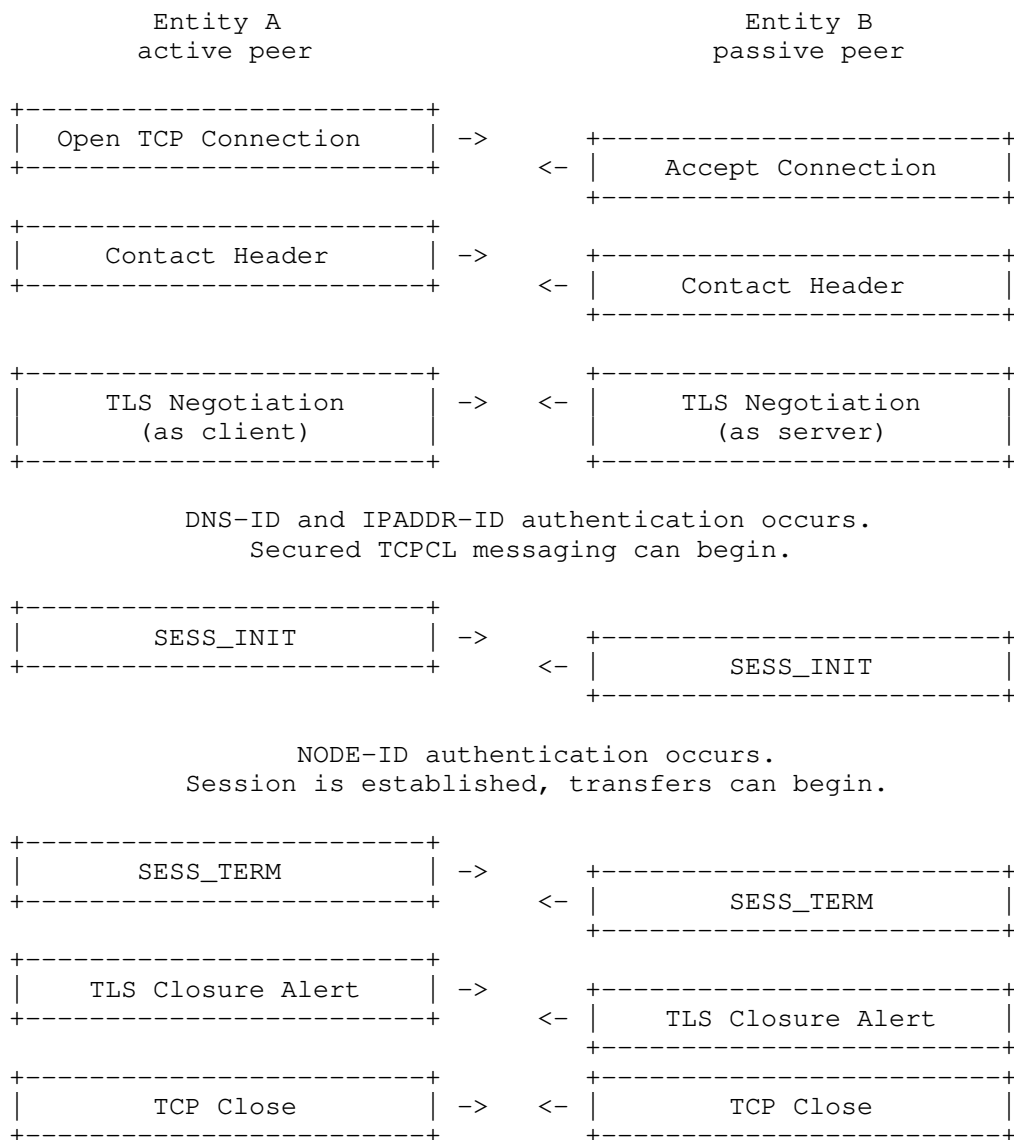


Figure 17: A simple visual example of TCPCL TLS Establishment between two entities

4.5. Message Header

After the initial exchange of a Contact Header and (if TLS is negotiated to be used) the TLS handshake, all messages transmitted over the session are identified by a one-octet header with the following structure:

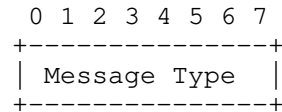


Figure 18: Format of the Message Header

The message header fields are as follows:

Message Type: Indicates the type of the message as per Table 2 below. Encoded values are listed in Section 9.5.

Name	Code	Description
SESS_INIT	0x07	Contains the session parameter inputs from one of the entities, as described in Section 4.6.
SESS_TERM	0x05	Indicates that one of the entities participating in the session wishes to cleanly terminate the session, as described in Section 6.1.
XFER_SEGMENT	0x01	Indicates the transmission of a segment of bundle data, as described in Section 5.2.2.
XFER_ACK	0x02	Acknowledges reception of a data segment, as described in Section 5.2.3.
XFER_REFUSE	0x03	Indicates that the transmission of the current bundle SHALL be stopped, as described in Section 5.2.4.
KEEPAIVE	0x04	Used to keep TCPCL session active, as described in Section 5.1.1.
MSG_REJECT	0x06	Contains a TCPCL message rejection, as described in Section 5.1.2.

Table 2: TCPCL Message Types

4.6. Session Initialization Message (SESS_INIT)

Before a session is established and ready to transfer bundles, the session parameters are negotiated between the connected entities. The SESS_INIT message is used to convey the per-entity parameters which are used together to negotiate the per-session parameters as described in Section 4.7.

The format of a SESS_INIT message is as follows in Figure 19.

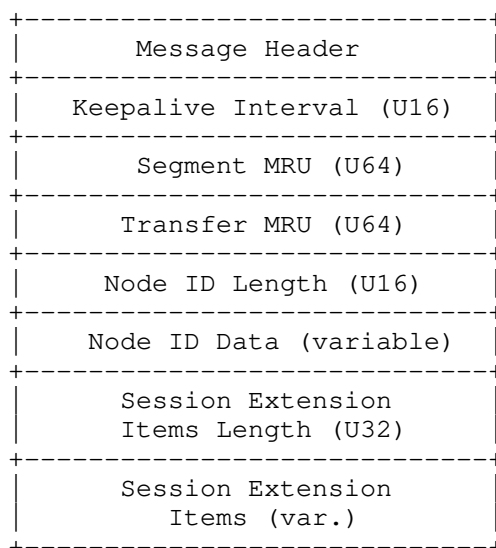


Figure 19: SESS_INIT Format

The fields of the SESS_INIT message are:

Keepalive Interval: A 16-bit unsigned integer indicating the minimum interval, in seconds, to negotiate as the Session Keepalive using the method of Section 4.7.

Segment MRU: A 64-bit unsigned integer indicating the largest allowable single-segment data payload size to be received in this session. Any XFER_SEGMENT sent to this peer SHALL have a data payload no longer than the peer's Segment MRU. The two entities of a single session MAY have different Segment MRUs, and no relation between the two is required.

Transfer MRU: A 64-bit unsigned integer indicating the largest allowable total-bundle data size to be received in this session. Any bundle transfer sent to this peer SHALL have a Total Bundle Length payload no longer than the peer's Transfer MRU. This value can be used to perform proactive bundle fragmentation. The two entities of a single session MAY have different Transfer MRUs, and no relation between the two is required.

Node ID Length and Node ID Data: Together these fields represent a variable-length text string. The Node ID Length is a 16-bit unsigned integer indicating the number of octets of Node ID Data to follow. A zero-length Node ID SHALL be used to indicate the lack of Node ID rather than a truly empty Node ID. This case

allows an entity to avoid exposing Node ID information on an untrusted network. A non-zero-length Node ID Data SHALL contain the UTF-8 encoded Node ID of the Entity which sent the SESS_INIT message. Every Node ID SHALL be a URI consistent with the requirements of [RFC3986] and the URI schemes of the IANA "Bundle Protocol URI Scheme Type" registry [IANA-BUNDLE]. The Node ID itself can be authenticated as described in Section 4.4.4.

Session Extension Length and Session Extension Items: Together these fields represent protocol extension data not defined by this specification. The Session Extension Length is the total number of octets to follow which are used to encode the Session Extension Item list. The encoding of each Session Extension Item is within a consistent data container as described in Section 4.8. The full set of Session Extension Items apply for the duration of the TCPCL session to follow. The order and multiplicity of these Session Extension Items is significant, as defined in the associated type specification(s). If the content of the Session Extension Items data disagrees with the Session Extension Length (e.g., the last Item claims to use more octets than are present in the Session Extension Length), the reception of the SESS_INIT is considered to have failed.

If an entity receives a peer Node ID which is not authenticated (by the procedure of Section 4.4.4.3) that Node ID SHOULD NOT be used by a BP agent for any discovery or routing functions. Trusting an unauthenticated Node ID can lead to the threat described in Section 8.9.

When the active entity initiates a TCPCL session, it is likely based on routing information which binds a Node ID to CL parameters used to initiate the session. If the active entity receives a SESS_INIT with different Node ID than was intended for the TCPCL session, the session MAY be allowed to be established. If allowed, such a session SHALL be associated with the Node ID provided in the SESS_INIT message rather than any intended value.

4.7. Session Parameter Negotiation

An entity calculates the parameters for a TCPCL session by negotiating the values from its own preferences (conveyed by the SESS_INIT it sent to the peer) with the preferences of the peer entity (expressed in the SESS_INIT that it received from the peer). The negotiated parameters defined by this specification are described in the following paragraphs.

Transfer MTU and Segment MTU: The maximum transmit unit (MTU) for

whole transfers and individual segments are identical to the Transfer MRU and Segment MRU, respectively, of the received SESS_INIT message. A transmitting peer can send individual segments with any size smaller than the Segment MTU, depending on local policy, dynamic network conditions, etc. Determining the size of each transmitted segment is an implementation matter. If either the Transfer MRU or Segment MRU is unacceptable, the entity SHALL terminate the session with a reason code of "Contact Failure".

Session Keepalive: Negotiation of the Session Keepalive parameter is performed by taking the minimum of the two Keepalive Interval values from the two SESS_INIT messages. The Session Keepalive interval is a parameter for the behavior described in Section 5.1.1. If the Session Keepalive interval is unacceptable, the entity SHALL terminate the session with a reason code of "Contact Failure". Note: a negotiated Session Keepalive of zero indicates that KEEPALIVES are disabled.

Once this process of parameter negotiation is completed, this protocol defines no additional mechanism to change the parameters of an established session; to effect such a change, the TCPCL session MUST be terminated and a new session established.

4.8. Session Extension Items

Each of the Session Extension Items SHALL be encoded in an identical Type-Length-Value (TLV) container form as indicated in Figure 20.

The fields of the Session Extension Item are:

Item Flags: A one-octet field containing generic bit flags about the Item, which are listed in Table 3. All reserved header flag bits SHALL be set to 0 by the sender. All reserved header flag bits SHALL be ignored by the receiver. If a TCPCL entity receives a Session Extension Item with an unknown Item Type and the CRITICAL flag of 1, the entity SHALL terminate the TCPCL session with SESS_TERM reason code of "Contact Failure". If the CRITICAL flag is 0, an entity SHALL skip over and ignore any item with an unknown Item Type.

Item Type: A 16-bit unsigned integer field containing the type of the extension item. This specification does not define any extension types directly, but does create an IANA registry for such codes (see Section 9.3).

Item Length: A 16-bit unsigned integer field containing the number of Item Value octets to follow.

Item Value: A variable-length data field which is interpreted according to the associated Item Type. This specification places no restrictions on an extension's use of available Item Value data. Extension specifications SHOULD avoid the use of large data lengths, as no bundle transfers can begin until the full extension data is sent.

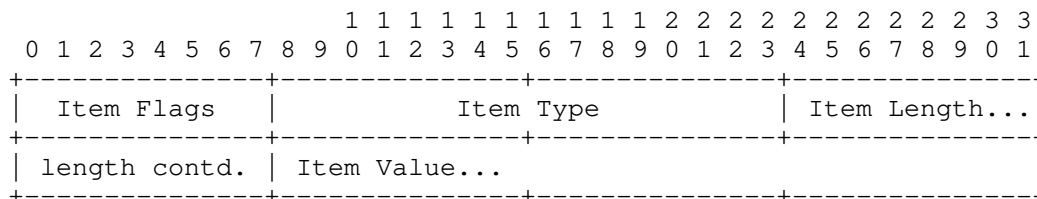


Figure 20: Session Extension Item Format

Name	Code	Description
CRITICAL	0x01	If bit is set, indicates that the receiving peer must handle the extension item.
Reserved	others	

Table 3: Session Extension Item Flags

5. Established Session Operation

This section describes the protocol operation for the duration of an established session, including the mechanism for transmitting bundles over the session.

5.1. Upkeep and Status Messages

5.1.1. Session Upkeep (KEEPALIVE)

The protocol includes a provision for transmission of KEEPALIVE messages over the TCPCL session to help determine if the underlying TCP connection has been disrupted.

As described in Section 4.3, a negotiated parameter of each session is the Session Keepalive interval. If the negotiated Session Keepalive is zero (i.e., one or both SESS_INIT messages contains a zero Keepalive Interval), then the keepalive feature is disabled. There is no logical minimum value for the keepalive interval (within the minimum imposed by the positive-value encoding), but when used

for many sessions on an open, shared network a short interval could lead to excessive traffic. For shared network use, entities SHOULD choose a keepalive interval no shorter than 30 seconds. There is no logical maximum value for the keepalive interval (within the maximum imposed by the fixed-size encoding), but an idle TCP connection is liable for closure by the host operating system if the keepalive time is longer than tens-of-minutes. Entities SHOULD choose a keepalive interval no longer than 10 minutes (600 seconds).

Note: The Keepalive Interval SHOULD NOT be chosen too short as TCP retransmissions MAY occur in case of packet loss. Those will have to be triggered by a timeout (TCP retransmission timeout (RTO)), which is dependent on the measured RTT for the TCP connection so that KEEPALIVE messages can experience noticeable latency.

The format of a KEEPALIVE message is a one-octet message type code of KEEPALIVE (as described in Table 2) with no additional data. Both sides SHALL send a KEEPALIVE message whenever the negotiated interval has elapsed with no transmission of any message (KEEPALIVE or other).

If no message (KEEPALIVE or other) has been received in a session after some implementation-defined time duration, then the entity SHALL terminate the session by transmitting a SESS_TERM message (as described in Section 6.1) with reason code "Idle Timeout". If configurable, the idle timeout duration SHOULD be no shorter than twice the keepalive interval. If not configurable, the idle timeout duration SHOULD be exactly twice the keepalive interval.

5.1.2. Message Rejection (MSG_REJECT)

This message type is not expected to be seen in a well-functioning session. Its purpose is to aid in troubleshooting bad entity behavior by allowing the peer to observe why an entity is not responding as expected to its messages.

If a TCPCL entity receives a message type which is unknown to it (possibly due to an unhandled protocol version mismatch or a incorrectly-negotiated session extension which defines a new message type), the entity SHALL send a MSG_REJECT message with a Reason Code of "Message Type Unknown" and close the TCP connection. If a TCPCL entity receives a message type which is known but is inappropriate for the negotiated session parameters (possibly due to incorrectly-negotiated session extension), the entity SHALL send a MSG_REJECT message with a Reason Code of "Message Unsupported". If a TCPCL entity receives a message which is inappropriate for the current session state (e.g., a SESS_INIT after the session has already been established or an XFER_ACK message with an unknown Transfer ID), the entity SHALL send a MSG_REJECT message with a Reason Code of "Message Unexpected".

The format of a MSG_REJECT message is as follows in Figure 21.

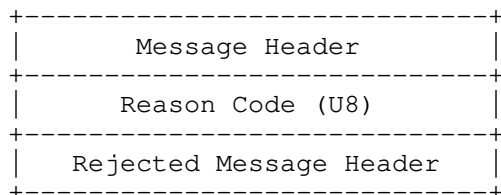


Figure 21: Format of MSG_REJECT Messages

The fields of the MSG_REJECT message are:

Reason Code: A one-octet refusal reason code interpreted according to the descriptions in Table 4.

Rejected Message Header: The Rejected Message Header is a copy of the Message Header to which the MSG_REJECT message is sent as a response.

Name	Code	Description
Message Type Unknown	0x01	A message was received with a Message Type code unknown to the TCPCL entity.
Message Unsupported	0x02	A message was received but the TCPCL entity cannot comply with the message contents.
Message Unexpected	0x03	A message was received while the session is in a state in which the message is not expected.

Table 4: MSG_REJECT Reason Codes

5.2. Bundle Transfer

All of the messages in this section are directly associated with transferring a bundle between TCPCL entities.

A single TCPCL transfer results in a bundle (handled by the convergence layer as opaque data) being exchanged from one entity to the other. In TCPCL a transfer is accomplished by dividing a single bundle up into "segments" based on the receiving-side Segment MRU (see Section 4.2). The choice of the length to use for segments is an implementation matter, but each segment MUST NOT be larger than the receiving entity's maximum receive unit (MRU) (see the field Segment MRU of Section 4.2). The first segment for a bundle is indicated by the 'START' flag and the last segment is indicated by the 'END' flag.

A single transfer (and by extension a single segment) SHALL NOT contain data of more than a single bundle. This requirement is imposed on the agent using the TCPCL rather than TCPCL itself.

If multiple bundles are transmitted on a single TCPCL connection, they MUST be transmitted consecutively without interleaving of segments from multiple bundles.

5.2.1. Bundle Transfer ID

Each of the bundle transfer messages contains a Transfer ID which is used to correlate messages (from both sides of a transfer) for each bundle. A Transfer ID does not attempt to address uniqueness of the bundle data itself and has no relation to concepts such as bundle fragmentation. Each invocation of TCPCL by the bundle protocol agent, requesting transmission of a bundle (fragmentary or otherwise), results in the initiation of a single TCPCL transfer. Each transfer entails the sending of a sequence of some number of XFER_SEGMENT and XFER_ACK messages; all are correlated by the same Transfer ID. The sending entity originates a transfer ID and the receiving entity uses that same Transfer ID in acknowledgements.

Transfer IDs from each entity SHALL be unique within a single TCPCL session. Upon exhaustion of the entire 64-bit Transfer ID space, the sending entity SHALL terminate the session with SESS_TERM reason code "Resource Exhaustion". For bidirectional bundle transfers, a TCPCL entity SHOULD NOT rely on any relation between Transfer IDs originating from each side of the TCPCL session.

Although there is not a strict requirement for Transfer ID initial values or ordering (see Section 8.13), in the absence of any other mechanism for generating Transfer IDs an entity SHALL use the following algorithm: The initial Transfer ID from each entity is zero and subsequent Transfer ID values are incremented from the prior Transfer ID value by one.

5.2.2. Data Transmission (XFER_SEGMENT)

Each bundle is transmitted in one or more data segments. The format of a XFER_SEGMENT message follows in Figure 22.

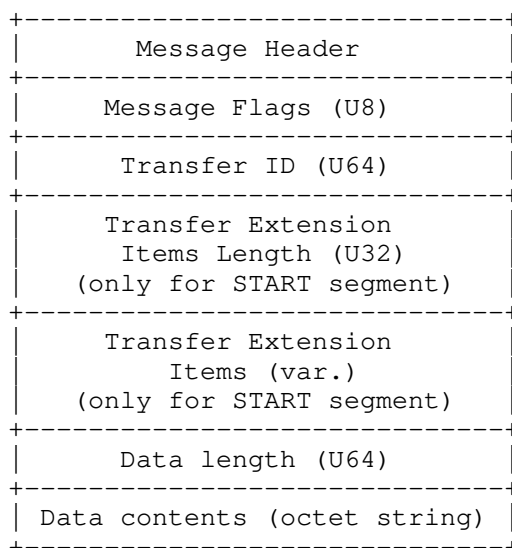


Figure 22: Format of XFER_SEGMENT Messages

The fields of the XFER_SEGMENT message are:

Message Flags: A one-octet field of single-bit flags, interpreted according to the descriptions in Table 5. All reserved header flag bits SHALL be set to 0 by the sender. All reserved header flag bits SHALL be ignored by the receiver.

Transfer ID: A 64-bit unsigned integer identifying the transfer being made.

Transfer Extension Length and Transfer Extension Items: Together these fields represent protocol extension data for this specification. The Transfer Extension Length and Transfer Extension Item fields SHALL only be present when the 'START' flag is set to 1 on the message. The Transfer Extension Length is the total number of octets to follow which are used to encode the Transfer Extension Item list. The encoding of each Transfer Extension Item is within a consistent data container as described in Section 5.2.5. The full set of transfer extension items apply only to the associated single transfer. The order and multiplicity of these transfer extension items is significant, as defined in the associated type specification(s). If the content of the Transfer Extension Items data disagrees with the Transfer Extension Length (e.g., the last Item claims to use more octets than are present in the Transfer Extension Length), the reception of the XFER_SEGMENT is considered to have failed.

Data length: A 64-bit unsigned integer indicating the number of octets in the Data contents to follow.

Data contents: The variable-length data payload of the message.

Name	Code	Description
END	0x01	If bit is set, indicates that this is the last segment of the transfer.
START	0x02	If bit is set, indicates that this is the first segment of the transfer.
Reserved	others	

Table 5: XFER_SEGMENT Flags

The flags portion of the message contains two flag values in the two low-order bits, denoted 'START' and 'END' in Table 5. The 'START' flag SHALL be set to 1 when transmitting the first segment of a transfer. The 'END' flag SHALL be set to 1 when transmitting the last segment of a transfer. In the case where an entire transfer is accomplished in a single segment, both the 'START' and 'END' flags SHALL be set to 1.

Once a transfer of a bundle has commenced, the entity MUST only send segments containing sequential portions of that bundle until it sends a segment with the 'END' flag set to 1. No interleaving of multiple transfers from the same entity is possible within a single TCPCL session. Simultaneous transfers between two entities MAY be achieved using multiple TCPCL sessions.

5.2.3. Data Acknowledgments (XFER_ACK)

Although the TCP transport provides reliable transfer of data between transport peers, the typical BSD sockets interface provides no means to inform a sending application of when the receiving application has processed some amount of transmitted data. Thus, after transmitting some data, the TCPCL needs an additional mechanism to determine whether the receiving agent has successfully received and fully processed the segment. To this end, the TCPCL protocol provides feedback messaging whereby a receiving entity transmits acknowledgments of reception of data segments.

The format of an XFER_ACK message follows in Figure 23.

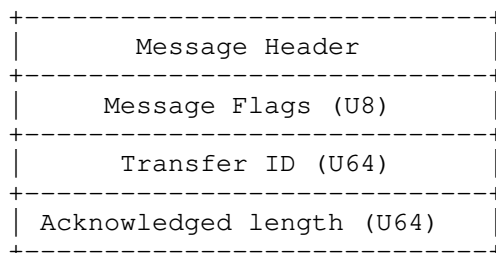


Figure 23: Format of XFER_ACK Messages

The fields of the XFER_ACK message are:

Message Flags: A one-octet field of single-bit flags, interpreted according to the descriptions in Table 5. All reserved header flag bits SHALL be set to 0 by the sender. All reserved header flag bits SHALL be ignored by the receiver.

Transfer ID: A 64-bit unsigned integer identifying the transfer being acknowledged.

Acknowledged length: A 64-bit unsigned integer indicating the total number of octets in the transfer which are being acknowledged.

A receiving TCPCL entity SHALL send an XFER_ACK message in response to each received XFER_SEGMENT message after the segment has been fully processed. The flags portion of the XFER_ACK header SHALL be set to match the corresponding XFER_SEGMENT message being acknowledged (including flags not decodable to the entity). The acknowledged length of each XFER_ACK contains the sum of the data length fields of all XFER_SEGMENT messages received so far in the course of the indicated transfer. The sending entity SHOULD transmit multiple XFER_SEGMENT messages without waiting for the corresponding XFER_ACK responses. This enables pipelining of messages on a transfer stream.

For example, suppose the sending entity transmits four segments of bundle data with lengths 100, 200, 500, and 1000, respectively. After receiving the first segment, the entity sends an acknowledgment of length 100. After the second segment is received, the entity sends an acknowledgment of length 300. The third and fourth acknowledgments are of length 800 and 1800, respectively.

5.2.4. Transfer Refusal (XFER_REFUSE)

The TCPCL supports a mechanism by which a receiving entity can indicate to the sender that it does not want to receive the corresponding bundle. To do so, upon receiving an XFER_SEGMENT message, the entity MAY transmit a XFER_REFUSE message. As data segments and acknowledgments can cross on the wire, the bundle that is being refused SHALL be identified by the Transfer ID of the refusal.

There is no required relation between the Transfer MRU of a TCPCL entity (which is supposed to represent a firm limitation of what the entity will accept) and sending of a XFER_REFUSE message. A XFER_REFUSE can be used in cases where the agent's bundle storage is temporarily depleted or somehow constrained. A XFER_REFUSE can also be used after the bundle header or any bundle data is inspected by an agent and determined to be unacceptable.

A transfer receiver MAY send an XFER_REFUSE message as soon as it receives any XFER_SEGMENT message. The transfer sender MUST be prepared for this and MUST associate the refusal with the correct bundle via the Transfer ID fields.

The TCPCL itself does not have any required behavior to respond to an XFER_REFUSE based on its Reason Code; the refusal is passed up as an indication to the BP agent that the transfer has been refused. If a transfer refusal has a Reason Code which is not decodable to the BP agent, the agent SHOULD treat the refusal as having an Unknown reason.

The format of the XFER_REFUSE message is as follows in Figure 24.

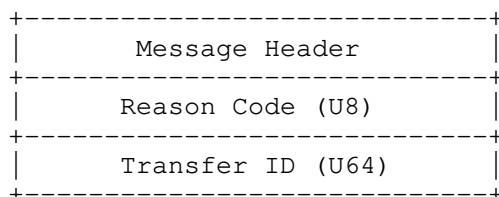


Figure 24: Format of XFER_REFUSE Messages

The fields of the XFER_REFUSE message are:

Reason Code: A one-octet refusal reason code interpreted according to the descriptions in Table 6.

Transfer ID: A 64-bit unsigned integer identifying the transfer

being refused.

Name	Code	Description
Unknown	0x00	Reason for refusal is unknown or not specified.
Completed	0x01	The receiver already has the complete bundle. The sender MAY consider the bundle as completely received.
No Resources	0x02	The receiver's resources are exhausted. The sender SHOULD apply reactive bundle fragmentation before retrying.
Retransmit	0x03	The receiver has encountered a problem that requires the bundle to be retransmitted in its entirety.
Not Acceptable	0x04	Some issue with the bundle data or the transfer extension data was encountered. The sender SHOULD NOT retry the same bundle with the same extensions.
Extension Failure	0x05	A failure processing the Transfer Extension Items has occurred.
Session Terminating	0x06	The receiving entity is in the process of terminating the session. The sender MAY retry the same bundle at a later time in a different session.

Table 6: XFER_REFUSE Reason Codes

The receiver MUST, for each transfer preceding the one to be refused, have either acknowledged all XFER_SEGMENT messages or refused the bundle transfer.

The bundle transfer refusal MAY be sent before an entire data segment is received. If a sender receives a XFER_REFUSE message, the sender MUST complete the transmission of any partially sent XFER_SEGMENT message. There is no way to interrupt an individual TCPCL message partway through sending it. The sender MUST NOT commence transmission of any further segments of the refused bundle subsequently. Note, however, that this requirement does not ensure that an entity will not receive another XFER_SEGMENT for the same

bundle after transmitting a XFER_REFUSE message since messages can cross on the wire; if this happens, subsequent segments of the bundle SHALL also be refused with a XFER_REFUSE message.

Note: If a bundle transmission is aborted in this way, the receiver does not receive a segment with the 'END' flag set to 1 for the aborted bundle. The beginning of the next bundle is identified by the 'START' flag set to 1, indicating the start of a new transfer, and with a distinct Transfer ID value.

5.2.5. Transfer Extension Items

Each of the Transfer Extension Items SHALL be encoded in an identical Type-Length-Value (TLV) container form as indicated in Figure 25.

The fields of the Transfer Extension Item are:

Item Flags: A one-octet field containing generic bit flags about the Item, which are listed in Table 7. All reserved header flag bits SHALL be set to 0 by the sender. All reserved header flag bits SHALL be ignored by the receiver. If a TCPCL entity receives a Transfer Extension Item with an unknown Item Type and the CRITICAL flag is 1, the entity SHALL refuse the transfer with an XFER_REFUSE reason code of "Extension Failure". If the CRITICAL flag is 0, an entity SHALL skip over and ignore any item with an unknown Item Type.

Item Type: A 16-bit unsigned integer field containing the type of the extension item. This specification creates an IANA registry for such codes (see Section 9.4).

Item Length: A 16-bit unsigned integer field containing the number of Item Value octets to follow.

Item Value: A variable-length data field which is interpreted according to the associated Item Type. This specification places no restrictions on an extension's use of available Item Value data. Extension specifications SHOULD avoid the use of large data lengths, as the associated transfer cannot begin until the full extension data is sent.

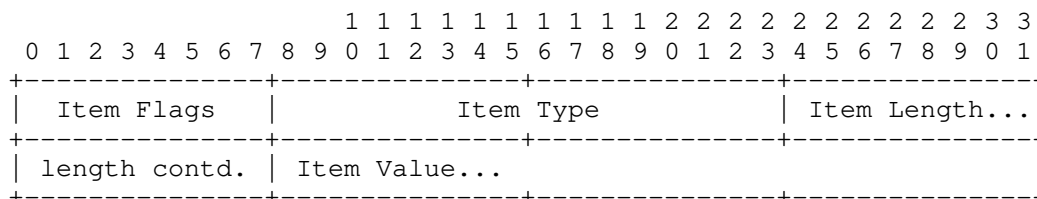


Figure 25: Transfer Extension Item Format

Name	Code	Description
CRITICAL	0x01	If bit is set, indicates that the receiving peer must handle the extension item.
Reserved	others	

Table 7: Transfer Extension Item Flags

5.2.5.1. Transfer Length Extension

The purpose of the Transfer Length extension is to allow entities to preemptively refuse bundles that would exceed their resources or to prepare storage on the receiving entity for the upcoming bundle data.

Multiple Transfer Length extension items SHALL NOT occur within the same transfer. The lack of a Transfer Length extension item in any transfer SHALL NOT imply anything about the potential length of the transfer. The Transfer Length extension SHALL be assigned transfer extension type ID 0x0001.

If a transfer occupies exactly one segment (i.e., both START and END flags are 1) the Transfer Length extension SHOULD NOT be present. The extension does not provide any additional information for single-segment transfers.

The format of the Transfer Length data is as follows in Figure 26.

Total Length (U64)

Figure 26: Format of Transfer Length data

The fields of the Transfer Length extension are:

Total Length: A 64-bit unsigned integer indicating the size of the data-to-be-transferred. The Total Length field SHALL be treated as authoritative by the receiver. If, for whatever reason, the actual total length of bundle data received differs from the value indicated by the Total Length value, the receiver SHALL treat the transmitted data as invalid and send an XFER_REFUSE with a Reason Code of "Not Acceptable".

6. Session Termination

This section describes the procedures for terminating a TCPCL session. The purpose of terminating a session is to allow transfers to complete before the TCP connection is closed but not allow any new transfers to start. A session state change is necessary for this to happen because transfers can be in-progress in either direction (transfer stream) within a session. Waiting for a transfer to complete in one direction does not control or influence the possibility of a transfer in the other direction. Either peer of a session can terminate an established session at any time.

6.1. Session Termination Message (SESS_TERM)

To cleanly terminate a session, a SESS_TERM message SHALL be transmitted by either entity at any point following complete transmission of any other message. When sent to initiate a termination, the REPLY flag of a SESS_TERM message SHALL be 0. Upon receiving a SESS_TERM message after not sending a SESS_TERM message in the same session, an entity SHALL send an acknowledging SESS_TERM message. When sent to acknowledge a termination, a SESS_TERM message SHALL have identical data content from the message being acknowledged except for the REPLY flag, which is set to 1 to indicate acknowledgement.

Once a SESS_TERM message is sent the state of that TCPCL session changes to Ending. While the session is in the Ending state, an entity MAY finish an in-progress transfer in either direction. While the session is in the Ending state, an entity SHALL NOT begin any new outgoing transfer for the remainder of the session. While the session is in the Ending state, an entity SHALL NOT accept any new incoming transfer for the remainder of the session. If a new incoming transfer is attempted while in the Ending state, the receiving entity SHALL send an XFER_REFUSE with a Reason Code of "Session Terminating".

There are circumstances where an entity has an urgent need to close a TCP connection associated with a TCPCL session, without waiting for transfers to complete but also in a way which doesn't force timeouts to occur; for example, due to impending shutdown of the underlying data link layer. Instead of following a clean termination sequence, after transmitting a SESS_TERM message an entity MAY perform an unclean termination by immediately closing the associated TCP connection. When performing an unclean termination, an entity SHOULD acknowledge all received XFER_SEGMENTS with an XFER_ACK before closing the TCP connection. Not acknowledging received segments can result in unnecessary bundle or bundle fragment retransmission. Any delay between request to close the TCP connection and actual closing

of the connection (a "half-closed" state) MAY be ignored by the TCPCL entity. If the underlying TCP connection is closed during a transmission (in either transfer stream), the transfer SHALL be indicated to the BP agent as failed (see the transmission failure and reception failure indications of Section 3.1).

The TCPCL itself does not have any required behavior to respond to an SESS_TERM based on its Reason Code; the termination is passed up as an indication to the BP agent that the session state has changed. If a termination has a Reason Code which is not decodable to the BP agent, the agent SHOULD treat the termination as having an Unknown reason.

The format of the SESS_TERM message is as follows in Figure 27.

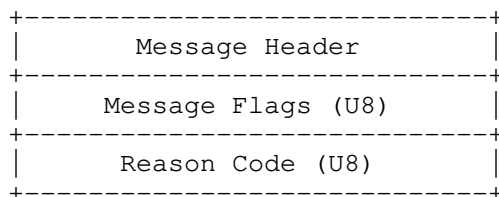


Figure 27: Format of SESS_TERM Messages

The fields of the SESS_TERM message are:

Message Flags: A one-octet field of single-bit flags, interpreted according to the descriptions in Table 8. All reserved header flag bits SHALL be set to 0 by the sender. All reserved header flag bits SHALL be ignored by the receiver.

Reason Code: A one-octet refusal reason code interpreted according to the descriptions in Table 9.

Name	Code	Description
REPLY	0x01	If bit is set, indicates that this message is an acknowledgement of an earlier SESS_TERM message.
Reserved	others	

Table 8: SESS_TERM Flags

Name	Code	Description
Unknown	0x00	A termination reason is not available.
Idle timeout	0x01	The session is being terminated due to idleness.
Version mismatch	0x02	The entity cannot conform to the specified TCPCL protocol version.
Busy	0x03	The entity is too busy to handle the current session.
Contact Failure	0x04	The entity cannot interpret or negotiate a Contact Header or SESS_INIT option.
Resource Exhaustion	0x05	The entity has run into some resource limit and cannot continue the session.

Table 9: SESS_TERM Reason Codes

The earliest a TCPCL session termination MAY occur is immediately after transmission of a Contact Header (and prior to any further message transmit). This can, for example, be used to notify that the entity is currently not able or willing to communicate. However, an entity MUST always send the Contact Header to its peer before sending a SESS_TERM message.

Termination of the TCP connection MAY occur prior to receiving the Contact header as discussed in Section 4.1. If reception of the Contact Header itself somehow fails (e.g., an invalid "magic string" is received), an entity SHALL close the TCP connection without sending a SESS_TERM message.

If a session is to be terminated before a protocol message has completed being sent, then the entity MUST NOT transmit the SESS_TERM message but still SHALL close the TCP connection. Each TCPCL message is contiguous in the octet stream and has no ability to be cut short and/or preempted by an other message. This is particularly important when large segment sizes are being transmitted; either entire XFER_SEGMENT is sent before a SESS_TERM message or the connection is simply terminated mid-XFER_SEGMENT.

6.2. Idle Session Shutdown

The protocol includes a provision for clean termination of idle sessions. Determining the length of time to wait before terminating idle sessions, if they are to be terminated at all, is an implementation and configuration matter.

If there is a configured time to terminate idle sessions and if no TCPCL messages (other than KEEPALIVE messages) has been received for at least that amount of time, then either entity MAY terminate the session by transmitting a SESS_TERM message indicating the reason code of "Idle timeout" (as described in Table 9).

7. Implementation Status

This section is to be removed before publishing as an RFC.

[NOTE to the RFC Editor: please remove this section before publication, as well as the reference to [RFC7942], [github-dtn-demo-agent], and [github-dtn-wireshark].]

This section records the status of known implementations of the protocol defined by this specification at the time of posting of this Internet-Draft, and is based on a proposal described in [RFC7942]. The description of implementations in this section is intended to assist the IETF in its decision processes in progressing drafts to RFCs. Please note that the listing of any individual implementation here does not imply endorsement by the IETF. Furthermore, no effort has been spent to verify the information presented here that was supplied by IETF contributors. This is not intended as, and must not be construed to be, a catalog of available implementations or their features. Readers are advised to note that other implementations can exist.

An example implementation of the this draft of TCPCLv4 has been created as a GitHub project [github-dtn-demo-agent] and is intended to use as a proof-of-concept and as a possible source of interoperability testing. This example implementation uses D-Bus as the CL-BP Agent interface, so it only runs on hosts which provide the Python "dbus" library.

A wireshark dissector for TCPCLv4 has been created as a GitHub project [github-dtn-wireshark] and has been kept in-sync with the latest encoding of this specification.

8. Security Considerations

This section separates security considerations into threat categories based on guidance of BCP 72 [RFC3552].

8.1. Threat: Passive Leak of Node Data

When used without TLS security, the TCPCL exposes the Node ID and other configuration data to passive eavesdroppers. This occurs even when no transfers occur within a TCPCL session. This can be avoided by always using TLS, even if authentication is not available (see Section 8.12).

8.2. Threat: Passive Leak of Bundle Data

TCPCL can be used to provide point-to-point transport security, but does not provide security of data-at-rest and does not guarantee end-to-end bundle security. The bundle security mechanisms defined in [I-D.ietf-dtn-bpsec] are to be used instead.

When used without TLS security, the TCPCL exposes all bundle data to passive eavesdroppers. This can be avoided by always using TLS, even if authentication is not available (see Section 8.12).

8.3. Threat: TCPCL Version Downgrade

When a TCPCL entity supports multiple versions of the protocol it is possible for a malicious or misconfigured peer to use an older version of TCPCL which does not support transport security. A on-path attacker can also manipulate a Contact Header to present a lower protocol version than desired.

It is up to security policies within each TCPCL entity to ensure that the negotiated TCPCL version meets transport security requirements.

8.4. Threat: Transport Security Stripping

When security policy allows non-TLS sessions, TCPCL does not protect against active network attackers. It is possible for a on-path attacker to set the CAN_TLS flag to 0 on either side of the Contact Header exchange, which will cause the negotiation of Section 4.3 to disable TLS. This leads to the "SSL Stripping" attack described in [RFC7457].

The purpose of the CAN_TLS flag is to allow the use of TCPCL on entities which simply do not have a TLS implementation available. When TLS is available on an entity, it is strongly encouraged that the security policy disallow non-TLS sessions. This requires that the TLS handshake occurs, regardless of the policy-driven parameters of the handshake and policy-driven handling of the handshake outcome.

One mechanism to mitigate the possibility of TLS stripping is the use of DNS-based Authentication of Named Entities (DANE) [RFC6698] toward the passive peer. This mechanism relies on DNS and is unidirectional, so it doesn't help with applying policy toward the active peer, but it can be useful in an environment using opportunistic security. The configuration and use of DANE are outside of the scope of this document.

The negotiated use of TLS is identical behavior to STARTTLS use in [RFC2595], [RFC4511], and others.

8.5. Threat: Weak TLS Configurations

Even when using TLS to secure the TCPCL session, the actual ciphersuite negotiated between the TLS peers can be insecure. Recommendations for ciphersuite use are included in BCP 195 [RFC7525]. It is up to security policies within each TCPCL entity to ensure that the negotiated TLS ciphersuite meets transport security requirements.

8.6. Threat: Untrusted End-Entity Certificate

The profile in Section 4.4.4 uses end-entity certificates chained up to a trusted root CA. During TLS handshake, either entity can send a certificate set which does not contain the full chain, possibly excluding intermediate or root CAs. In an environment where peers are known to already contain needed root and intermediate CAs there is no need to include those CAs, but this has a risk of an entity not actually having one of the needed CAs.

8.7. Threat: Certificate Validation Vulnerabilities

Even when TLS itself is operating properly an attacker can attempt to exploit vulnerabilities within certificate check algorithms or configuration to establish a secure TCPCL session using an invalid certificate. A BP agent treats the peer Node ID within a TCPCL session as authoritative and an invalid certificate exploit could lead to bundle data leaking and/or denial of service to the Node ID being impersonated.

There are many reasons, described in [RFC5280] and [RFC6125], why a certificate can fail to validate, including using the certificate outside of its valid time interval, using purposes for which it was not authorized, or using it after it has been revoked by its CA. Validating a certificate is a complex task and can require network connectivity outside of the primary TCPCL network path(s) if a mechanism such as OCSP [RFC6960] is used by the CA. The configuration and use of particular certificate validation methods are outside of the scope of this document.

8.8. Threat: Symmetric Key Limits

Even with a secure block cipher and securely-established session keys, there are limits to the amount of plaintext which can be safely encrypted with a given set of keys as described in [AEAD-LIMITS]. When permitted by the negotiated TLS version (see [RFC8446]), it is advisable to take advantage of session key updates to avoid those limits.

8.9. Threat: BP Node Impersonation

The certificates exchanged by TLS enable authentication of peer DNS name and Node ID, but it is possible that a peer either not provide a valid certificate or that the certificate does not validate either the DNS-ID/IPADDR-ID or NODE-ID of the peer (see Section 3.4). Having a CA-validated certificate does not alone guarantee the identity of the network host or BP node from which the certificate is provided; additional validation procedures in Section 4.4.3 bind the DNS-ID/IPADDR-ID or NODE-ID based on the contents of the certificate.

The DNS-ID/IPADDR-ID validation is a weaker form of authentication, because even if a peer is operating on an authenticated network DNS name or IP address it can provide an invalid Node ID and cause bundles to be "leaked" to an invalid node. Especially in DTN environments, network names and addresses of nodes can be time-variable so binding a certificate to a Node ID is a more stable identity.

NODE-ID validation ensures that the peer to which a bundle is transferred is in fact the node which the BP Agent expects it to be. In circumstances where certificates can only be issued to DNS names, Node ID validation is not possible but it could be reasonable to assume that a trusted host is not going to present an invalid Node ID. Determining when a DNS-ID/IPADDR-ID authentication can be trusted to validate a Node ID is also a policy matter outside of the scope of this document.

One mitigation to arbitrary entities with valid PKIX certificates impersonating arbitrary Node IDs is the use of the PKIX Extended Key Usage key purpose id-kp-bundleSecurity (see Section 4.4.2.1). When this Extended Key Usage is present in the certificate, it represents a stronger assertion that the private key holder should in fact be trusted to operate as a DTN Node.

8.10. Threat: Denial of Service

The behaviors described in this section all amount to a potential denial-of-service to a TCPCL entity. The denial-of-service could be limited to an individual TCPCL session, could affect other well-behaving sessions on an entity, or could affect all sessions on a host.

A malicious entity can continually establish TCPCL sessions and delay sending of protocol-required data to trigger timeouts. The victim entity can block TCP connections from network peers which are thought to be incorrectly behaving within TCPCL.

An entity can send a large amount of data over a TCPCL session, requiring the receiving entity to handle the data. The victim entity can attempt to stop the flood of data by sending an XFER_REFUSE message, or forcibly terminate the session.

There is the possibility of a "data dribble" attack in which an entity presents a very small Segment MRU which causes transfers to be split among an large number of very small segments and causes the segmentation overhead to overwhelm the actual bundle data segments. Similarly, an entity can present a very small Transfer MRU which will cause resources to be wasted on establishment and upkeep of a TCPCL session over which a bundle could never be transferred. The victim entity can terminate the session during the negotiation of Section 4.7 if the MRUs are unacceptable.

The keepalive mechanism can be abused to waste throughput within a network link which would otherwise be usable for bundle transmissions. Due to the quantization of the Keepalive Interval parameter the smallest Session Keepalive is one second, which should be long enough to not flood the link. The victim entity can terminate the session during the negotiation of Section 4.7 if the Keepalive Interval is unacceptable.

Finally, an attacker or a misconfigured entity can cause issues at the TCP connection which will cause unnecessary TCP retransmissions or connection resets, effectively denying the use of the overlying TCPCL session.

8.11. Mandatory-to-Implement TLS

Following IETF best current practice, TLS is mandatory to implement for all TCPCL implementations but TLS is optional to use for a given TCPCL session. The recommended configuration of Section 4.2 is to always enable TLS, but entities are permitted to disable TLS based on local configuration. The configuration to enable or disable TLS for an entity or a session is outside of the scope of this document. The configuration to disable TLS is different from the threat of TLS stripping described in Section 8.4.

8.12. Alternate Uses of TLS

This specification makes use of PKIX certificate validation and authentication within TLS. There are alternate uses of TLS which are not necessarily incompatible with the security goals of this specification, but are outside of the scope of this document. The following subsections give examples of alternate TLS uses.

8.12.1. TLS Without Authentication

In environments where PKI is available but there are restrictions on the issuance of certificates (including the contents of certificates), it may be possible to make use of TLS in a way which authenticates only the passive entity of a TCPCL session or which does not authenticate either entity. Using TLS in a way which does not successfully authenticate some claim of both peer entities of a TCPCL session is outside of the scope of this document but does have similar properties to the opportunistic security model of [RFC7435].

8.12.2. Non-Certificate TLS Use

In environments where PKI is unavailable, alternate uses of TLS which do not require certificates such as pre-shared key (PSK) authentication [RFC5489] and the use of raw public keys [RFC7250] are available and can be used to ensure confidentiality within TCPCL. Using non-PKI node authentication methods is outside of the scope of this document.

8.13. Predictability of Transfer IDs

The only requirement on Transfer IDs is that they be unique with each session from the sending peer only. The trivial algorithm of the first transfer starting at zero and later transfers incrementing by one causes absolutely predictable Transfer IDs. Even when a TCPCL session is not TLS secured and there is a on-path attacker causing denial of service with XFER_REFUSE messages, it is not possible to preemptively refuse a transfer so there is no benefit in having unpredictable Transfer IDs within a session.

9. IANA Considerations

Registration procedures referred to in this section are defined in [RFC8126].

Some of the registries have been defined as version specific to TCPCLv4, and imports some or all codepoints from TCPCLv3. This was done to disambiguate the use of these codepoints between TCPCLv3 and TCPCLv4 while preserving the semantics of some of the codepoints.

9.1. Port Number

Within the port registry of [IANA-PORTS], TCP port number 4556 has been previously assigned as the default port for the TCP convergence layer in [RFC7242]. This assignment is unchanged by TCPCL version 4, but the assignment reference is updated to this specification. Each TCPCL entity identifies its TCPCL protocol version in its initial contact (see Section 9.2), so there is no ambiguity about what protocol is being used. The related assignments for UDP and DCCP port 4556 (both registered by [RFC7122]) are unchanged.

Parameter	Value
Service Name:	dtn-bundle
Transport Protocol(s):	TCP
Assignee:	IESG <iesg@ietf.org>
Contact:	IESG <iesg@ietf.org>
Description:	DTN Bundle TCP CL Protocol
Reference:	This specification.
Port Number:	4556

Table 10

9.2. Protocol Versions

IANA has created, under the "Bundle Protocol" registry [IANA-BUNDLE], a sub-registry titled "Bundle Protocol TCP Convergence-Layer Version Numbers". The version number table is updated to include this specification. The registration procedure is RFC Required.

Value	Description	Reference
0	Reserved	[RFC7242]
1	Reserved	[RFC7242]
2	Reserved	[RFC7242]
3	TCPCL	[RFC7242]
4	TCPCLv4	This specification.
5-255	Unassigned	

Table 11

9.3. Session Extension Types

EDITOR NOTE: sub-registry to-be-created upon publication of this specification.

IANA will create, under the "Bundle Protocol" registry [IANA-BUNDLE], a sub-registry titled "Bundle Protocol TCP Convergence-Layer Version 4 Session Extension Types" and initialize it with the contents of Table 12. The registration procedure is Expert Review within the lower range 0x0001--0x7FFF. Values in the range 0x8000--0xFFFF are reserved for use on private networks for functions not published to the IANA.

Specifications of new session extension types need to define the encoding of the Item Value data as well as any meaning or restriction on the number of or order of instances of the type within an extension item list. Specifications need to define how the extension functions when no instance of the new extension type is received during session negotiation.

Expert(s) are encouraged to be biased towards approving registrations unless they are abusive, frivolous, or actively harmful (not merely aesthetically displeasing, or architecturally dubious).

Code	Session Extension Type
0x0000	Reserved
0x0001--0x7FFF	Unassigned
0x8000--0xFFFF	Private/Experimental Use

Table 12: Session Extension Type Codes

9.4. Transfer Extension Types

EDITOR NOTE: sub-registry to-be-created upon publication of this specification.

IANA will create, under the "Bundle Protocol" registry [IANA-BUNDLE], a sub-registry titled "Bundle Protocol TCP Convergence-Layer Version 4 Transfer Extension Types" and initialize it with the contents of Table 13. The registration procedure is Expert Review within the lower range 0x0001--0x7FFF. Values in the range 0x8000--0xFFFF are reserved for use on private networks for functions not published to the IANA.

Specifications of new transfer extension types need to define the encoding of the Item Value data as well as any meaning or restriction on the number of or order of instances of the type within an extension item list. Specifications need to define how the extension functions when no instance of the new extension type is received in a transfer.

Expert(s) are encouraged to be biased towards approving registrations unless they are abusive, frivolous, or actively harmful (not merely aesthetically displeasing, or architecturally dubious).

Code	Transfer Extension Type
0x0000	Reserved
0x0001	Transfer Length Extension
0x0002--0x7FFF	Unassigned
0x8000--0xFFFF	Private/Experimental Use

Table 13: Transfer Extension Type Codes

9.5. Message Types

EDITOR NOTE: sub-registry to-be-created upon publication of this specification.

IANA will create, under the "Bundle Protocol" registry [IANA-BUNDLE], a sub-registry titled "Bundle Protocol TCP Convergence-Layer Version 4 Message Types" and initialize it with the contents of Table 14. The registration procedure is RFC Required within the lower range 0x01--0xEF. Values in the range 0xF0--0xFF are reserved for use on private networks for functions not published to the IANA.

Specifications of new message types need to define the encoding of the message data as well as the purpose and relationship of the new message to existing session/transfer state within the baseline message sequencing. The use of new message types need to be negotiated between TCPCL entities within a session (using the session extension mechanism) so that the receiving entity can properly decode all message types used in the session.

Expert(s) are encouraged to favor new session/transfer extension types over new message types. TCPCL messages are not self-delimiting, so care must be taken in introducing new message types.

If an entity receives an unknown message type the only thing that can be done is to send a MSG_REJECT and close the TCP connection; not even a clean termination can be done at that point.

Code	Message Type
0x00	Reserved
0x01	XFER_SEGMENT
0x02	XFER_ACK
0x03	XFER_REFUSE
0x04	KEEPALIVE
0x05	SESS_TERM
0x06	MSG_REJECT
0x07	SESS_INIT
0x08--0xEF	Unassigned
0xF0--0xFF	Private/Experimental Use

Table 14: Message Type Codes

9.6. XFER_REFUSE Reason Codes

EDITOR NOTE: sub-registry to-be-created upon publication of this specification.

IANA will create, under the "Bundle Protocol" registry [IANA-BUNDLE], a sub-registry titled "Bundle Protocol TCP Convergence-Layer Version 4 XFER_REFUSE Reason Codes" and initialize it with the contents of Table 15. The registration procedure is Specification Required within the lower range 0x00--0xEF. Values in the range 0xF0--0xFF are reserved for use on private networks for functions not published to the IANA.

Specifications of new XFER_REFUSE reason codes need to define the meaning of the reason and disambiguate it with pre-existing reasons. Each refusal reason needs to be usable by the receiving BP Agent to make retransmission or re-routing decisions.

Expert(s) are encouraged to be biased towards approving registrations unless they are abusive, frivolous, or actively harmful (not merely aesthetically displeasing, or architecturally dubious).

Code	Refusal Reason
0x00	Unknown
0x01	Completed
0x02	No Resources
0x03	Retransmit
0x04	Not Acceptable
0x05	Extension Failure
0x06	Session Terminating
0x07--0xEF	Unassigned
0xF0--0xFF	Private/Experimental Use

Table 15: XFER_REFUSE Reason Codes

9.7. SESS_TERM Reason Codes

EDITOR NOTE: sub-registry to-be-created upon publication of this specification.

IANA will create, under the "Bundle Protocol" registry [IANA-BUNDLE], a sub-registry titled "Bundle Protocol TCP Convergence-Layer Version 4 SESS_TERM Reason Codes" and initialize it with the contents of Table 16. The registration procedure is Specification Required within the lower range 0x00--0xEF. Values in the range 0xF0--0xFF are reserved for use on private networks for functions not published to the IANA.

Specifications of new SESS_TERM reason codes need to define the meaning of the reason and disambiguate it with pre-existing reasons. Each termination reason needs to be usable by the receiving BP Agent to make re-connection decisions.

Expert(s) are encouraged to be biased towards approving registrations unless they are abusive, frivolous, or actively harmful (not merely aesthetically displeasing, or architecturally dubious).

Code	Termination Reason
0x00	Unknown
0x01	Idle timeout
0x02	Version mismatch
0x03	Busy
0x04	Contact Failure
0x05	Resource Exhaustion
0x06--0xEF	Unassigned
0xF0--0xFF	Private/Experimental Use

Table 16: SESS_TERM Reason Codes

9.8. MSG_REJECT Reason Codes

EDITOR NOTE: sub-registry to-be-created upon publication of this specification.

IANA will create, under the "Bundle Protocol" registry [IANA-BUNDLE], a sub-registry titled "Bundle Protocol TCP Convergence-Layer Version 4 MSG_REJECT Reason Codes" and initialize it with the contents of Table 17. The registration procedure is Specification Required within the lower range 0x01--0xEF. Values in the range 0xF0--0xFF are reserved for use on private networks for functions not published to the IANA.

Specifications of new MSG_REJECT reason codes need to define the meaning of the reason and disambiguate it with pre-existing reasons. Each rejection reason needs to be usable by the receiving TCPCL Entity to make message sequencing and/or session termination decisions.

Expert(s) are encouraged to be biased towards approving registrations unless they are abusive, frivolous, or actively harmful (not merely aesthetically displeasing, or architecturally dubious).

Code	Rejection Reason
0x00	reserved
0x01	Message Type Unknown
0x02	Message Unsupported
0x03	Message Unexpected
0x04--0xEF	Unassigned
0xF0--0xFF	Private/Experimental Use

Table 17: MSG_REJECT Reason Codes

9.9. Object Identifier for PKIX Module Identifier

IANA has created, under the "Structure of Management Information (SMI) Numbers" registry [IANA-SMI], a sub-registry titled "SMI Security for PKIX Module Identifier". The table is updated to include a row "id-mod-dtn-tcpclv4-2021" for identifying the module in Appendix B as in the following table.

Decimal	Description	References
MOD-TBD	id-mod-dtn-tcpclv4-2021	This specification.

Table 18

9.10. Object Identifier for PKIX Other Name Forms

IANA has created, under the "Structure of Management Information (SMI) Numbers" registry [IANA-SMI], a sub-registry titled "SMI Security for PKIX Other Name Forms". The other name forms table is updated to include a row "id-on-bundleEID" for identifying DTN Endpoint IDs as in the following table.

Decimal	Description	References
ON-TBD	id-on-bundleEID	This specification.

Table 19

The formal structure of the associated other name form is in Appendix B. The use of this OID is defined in Section 4.4.1 and Section 4.4.2.

9.11. Object Identifier for PKIX Extended Key Usage

IANA has created, under the "Structure of Management Information (SMI) Numbers" registry [IANA-SMI], a sub-registry titled "SMI Security for PKIX Extended Key Purpose". The extended key purpose table is updated to include a purpose "id-kp-bundleSecurity" for identifying DTN endpoints as in the following table.

Decimal	Description	References
KP-TBD	id-kp-bundleSecurity	This specification.

Table 20

The formal definition of this EKU is in Appendix B. The use of this OID is defined in Section 4.4.2.

10. Acknowledgments

This specification is based on comments on implementation of [RFC7242] provided from Scott Burleigh.

11. References

11.1. Normative References

[IANA-BUNDLE]

IANA, "Bundle Protocol",
<<https://www.iana.org/assignments/bundle/>>.

[IANA-PORTS]

IANA, "Service Name and Transport Protocol Port Number Registry", <<https://www.iana.org/assignments/service-names-port-numbers/>>.

- [IANA-SMI] IANA, "Structure of Management Information (SMI) Numbers", <<https://www.iana.org/assignments/smi-numbers/>>.
- [RFC0793] Postel, J., "Transmission Control Protocol", STD 7, RFC 793, DOI 10.17487/RFC0793, September 1981, <<https://www.rfc-editor.org/info/rfc793>>.
- [RFC1122] Braden, R., Ed., "Requirements for Internet Hosts - Communication Layers", STD 3, RFC 1122, DOI 10.17487/RFC1122, October 1989, <<https://www.rfc-editor.org/info/rfc1122>>.
- [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>>.
- [RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, DOI 10.17487/RFC3986, January 2005, <<https://www.rfc-editor.org/info/rfc3986>>.
- [RFC5280] Cooper, D., Santesson, S., Farrell, S., Boeyen, S., Housley, R., and W. Polk, "Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile", RFC 5280, DOI 10.17487/RFC5280, May 2008, <<https://www.rfc-editor.org/info/rfc5280>>.
- [RFC6066] Eastlake 3rd, D., "Transport Layer Security (TLS) Extensions: Extension Definitions", RFC 6066, DOI 10.17487/RFC6066, January 2011, <<https://www.rfc-editor.org/info/rfc6066>>.
- [RFC6125] Saint-Andre, P. and J. Hodges, "Representation and Verification of Domain-Based Application Service Identity within Internet Public Key Infrastructure Using X.509 (PKIX) Certificates in the Context of Transport Layer Security (TLS)", RFC 6125, DOI 10.17487/RFC6125, March 2011, <<https://www.rfc-editor.org/info/rfc6125>>.
- [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, <<https://www.rfc-editor.org/info/rfc6960>>.

- [RFC7525] Sheffer, Y., Holz, R., and P. Saint-Andre, "Recommendations for Secure Use of Transport Layer Security (TLS) and Datagram Transport Layer Security (DTLS)", BCP 195, RFC 7525, DOI 10.17487/RFC7525, May 2015, <<https://www.rfc-editor.org/info/rfc7525>>.
- [RFC8126] Cotton, M., Leiba, B., and T. Narten, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 8126, DOI 10.17487/RFC8126, June 2017, <<https://www.rfc-editor.org/info/rfc8126>>.
- [RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <<https://www.rfc-editor.org/info/rfc8174>>.
- [RFC8446] Rescorla, E., "The Transport Layer Security (TLS) Protocol Version 1.3", RFC 8446, DOI 10.17487/RFC8446, August 2018, <<https://www.rfc-editor.org/info/rfc8446>>.
- [I-D.ietf-dtn-bpbis]
Burleigh, S., Fall, K., and E. J. Birrane, "Bundle Protocol Version 7", Work in Progress, Internet-Draft, draft-ietf-dtn-bpbis-31, 25 January 2021, <<https://datatracker.ietf.org/doc/html/draft-ietf-dtn-bpbis-31>>.
- [X.680] ITU-T, "Information technology -- Abstract Syntax Notation One (ASN.1): Specification of basic notation", ITU-T Recommendation X.680, ISO/IEC 8824-1:2015, August 2015, <<https://www.itu.int/rec/T-REC-X.680-201508-I/en>>.

11.2. Informative References

- [AEAD-LIMITS]
Luykx, A. and K. Paterson, "Limits on Authenticated Encryption Use in TLS", August 2017, <<http://www.isg.rhul.ac.uk/~kp/TLS-AEbounds.pdf>>.
- [RFC2595] Newman, C., "Using TLS with IMAP, POP3 and ACAP", RFC 2595, DOI 10.17487/RFC2595, June 1999, <<https://www.rfc-editor.org/info/rfc2595>>.
- [RFC3552] Rescorla, E. and B. Korver, "Guidelines for Writing RFC Text on Security Considerations", BCP 72, RFC 3552, DOI 10.17487/RFC3552, July 2003, <<https://www.rfc-editor.org/info/rfc3552>>.

- [RFC4511] Serfersheim, J., Ed., "Lightweight Directory Access Protocol (LDAP): The Protocol", RFC 4511, DOI 10.17487/RFC4511, June 2006, <<https://www.rfc-editor.org/info/rfc4511>>.
- [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, <<https://www.rfc-editor.org/info/rfc4838>>.
- [RFC5489] Badra, M. and I. Hajjeh, "ECDHE_PSK Cipher Suites for Transport Layer Security (TLS)", RFC 5489, DOI 10.17487/RFC5489, March 2009, <<https://www.rfc-editor.org/info/rfc5489>>.
- [RFC5912] Hoffman, P. and J. Schaad, "New ASN.1 Modules for the Public Key Infrastructure Using X.509 (PKIX)", RFC 5912, DOI 10.17487/RFC5912, June 2010, <<https://www.rfc-editor.org/info/rfc5912>>.
- [RFC6698] Hoffman, P. and J. Schlyter, "The DNS-Based Authentication of Named Entities (DANE) Transport Layer Security (TLS) Protocol: TLSA", RFC 6698, DOI 10.17487/RFC6698, August 2012, <<https://www.rfc-editor.org/info/rfc6698>>.
- [RFC7122] Kruse, H., Jero, S., and S. Ostermann, "Datagram Convergence Layers for the Delay- and Disruption-Tolerant Networking (DTN) Bundle Protocol and Licklider Transmission Protocol (LTP)", RFC 7122, DOI 10.17487/RFC7122, March 2014, <<https://www.rfc-editor.org/info/rfc7122>>.
- [RFC7242] Demmer, M., Ott, J., and S. Perreault, "Delay-Tolerant Networking TCP Convergence-Layer Protocol", RFC 7242, DOI 10.17487/RFC7242, June 2014, <<https://www.rfc-editor.org/info/rfc7242>>.
- [RFC7250] Wouters, P., Ed., Tschofenig, H., Ed., Gilmore, J., Weiler, S., and T. Kivinen, "Using Raw Public Keys in Transport Layer Security (TLS) and Datagram Transport Layer Security (DTLS)", RFC 7250, DOI 10.17487/RFC7250, June 2014, <<https://www.rfc-editor.org/info/rfc7250>>.
- [RFC7435] Dukhovni, V., "Opportunistic Security: Some Protection Most of the Time", RFC 7435, DOI 10.17487/RFC7435, December 2014, <<https://www.rfc-editor.org/info/rfc7435>>.

- [RFC7457] Sheffer, Y., Holz, R., and P. Saint-Andre, "Summarizing Known Attacks on Transport Layer Security (TLS) and Datagram TLS (DTLS)", RFC 7457, DOI 10.17487/RFC7457, February 2015, <<https://www.rfc-editor.org/info/rfc7457>>.
- [RFC7942] Sheffer, Y. and A. Farrel, "Improving Awareness of Running Code: The Implementation Status Section", BCP 205, RFC 7942, DOI 10.17487/RFC7942, July 2016, <<https://www.rfc-editor.org/info/rfc7942>>.
- [RFC8555] Barnes, R., Hoffman-Andrews, J., McCarney, D., and J. Kasten, "Automatic Certificate Management Environment (ACME)", RFC 8555, DOI 10.17487/RFC8555, March 2019, <<https://www.rfc-editor.org/info/rfc8555>>.
- [I-D.ietf-dtn-bpsec]
III, E. J. B. and K. McKeever, "Bundle Protocol Security Specification", Work in Progress, Internet-Draft, draft-ietf-dtn-bpsec-27, 16 February 2021, <<https://datatracker.ietf.org/doc/html/draft-ietf-dtn-bpsec-27>>.
- [I-D.ietf-dtn-bibect]
Burleigh, S., "Bundle-in-Bundle Encapsulation", Work in Progress, Internet-Draft, draft-ietf-dtn-bibect-03, 18 February 2020, <<https://datatracker.ietf.org/doc/html/draft-ietf-dtn-bibect-03>>.
- [github-dtn-demo-agent]
Sipos, B., "TCPCL Example Implementation", <<https://github.com/BSipos-RKF/dtn-demo-agent/>>.
- [github-dtn-wireshark]
Sipos, B., "TCPCL Wireshark Dissector", <<https://github.com/BSipos-RKF/dtn-wireshark/>>.

Appendix A. Significant changes from RFC7242

The areas in which changes from [RFC7242] have been made to existing headers and messages are:

- * Split Contact Header into pre-TLS protocol negotiation and SESS_INIT parameter negotiation. The Contact Header is now fixed-length.
- * Changed Contact Header content to limit number of negotiated options.

- * Added session option to negotiate maximum segment size (per each direction).
- * Renamed "Endpoint ID" to "Node ID" to conform with BPv7 terminology.
- * Added session extension capability.
- * Added transfer extension capability. Moved transfer total length into an extension item.
- * Defined new IANA registries for message / type / reason codes to allow renaming some codes for clarity.
- * Segments of all new IANA registries are reserved for private/experimental use.
- * Expanded Message Header to octet-aligned fields instead of bit-packing.
- * Added a bundle transfer identification number to all bundle-related messages (XFER_SEGMENT, XFER_ACK, XFER_REFUSE).
- * Use flags in XFER_ACK to mirror flags from XFER_SEGMENT.
- * Removed all uses of SDNV fields and replaced with fixed-bit-length (network byte order) fields.
- * Renamed SHUTDOWN to SESS_TERM to deconflict term "shutdown" related to TCP connections.
- * Removed the notion of a re-connection delay parameter.

The areas in which extensions from [RFC7242] have been made as new messages and codes are:

- * Added contact negotiation failure SESS_TERM reason code.
- * Added MSG_REJECT message to indicate an unknown or unhandled message was received.
- * Added TLS connection security mechanism.
- * Added "Not Acceptable", "Extension Failure", and "Session Terminating" XFER_REFUSE reason codes.
- * Added "Resource Exhaustion" SESS_TERM reason code.

Appendix B. ASN.1 Module

The following ASN.1 module formally specifies the BundleEID structure, its Other Name form, and the bundleSecurity Extended Key Usage in the syntax of [X.680]. This specification uses the ASN.1 definitions from [RFC5912] with the 2002 ASN.1 notation used in that document.


```
<CODE BEGINS>
DTN-TCPCLv4-2021
{ iso(1) identified-organization(3) dod(6)
  internet(1) security(5) mechanisms(5) pkix(7) id-mod(0)
  id-mod-dtn-tcpclv4-2021(MOD-TBD) }

DEFINITIONS IMPLICIT TAGS ::=
BEGIN

IMPORTS
  OTHER-NAME
  FROM PKIX1Implicit-2009 -- [RFC5912]
    { iso(1) identified-organization(3) dod(6) internet(1)
      security(5) mechanisms(5) pkix(7) id-mod(0)
      id-mod-pkix1-implicit-02(59) }

  id-pkix
  FROM PKIX1Explicit-2009 -- [RFC5912]
    { iso(1) identified-organization(3) dod(6) internet(1)
      security(5) mechanisms(5) pkix(7) id-mod(0)
      id-mod-pkix1-explicit-02(51) } ;

id-kp OBJECT IDENTIFIER ::= { id-pkix 3 }

id-on OBJECT IDENTIFIER ::= { id-pkix 8 }

DTNOtherNames OTHER-NAME ::= { on-bundleEID, ... }

-- The otherName definition for Bundle EID
on-bundleEID OTHER-NAME ::= {
  BundleEID IDENTIFIED BY { id-on-bundleEID }
}

id-on-bundleEID OBJECT IDENTIFIER ::= { id-on ON-TBD }

-- Same encoding as GeneralName of uniformResourceIdentifier
BundleEID ::= IA5String

-- The Extended Key Usage key for bundle security
id-kp-bundleSecurity OBJECT IDENTIFIER ::= { id-kp KP-TBD }

END
<CODE ENDS>
```

Appendix C. Example of the BundleEID Other Name Form

EDITOR NOTE: The encoded hex part "0b" and OID segment "11" are to be replaced by ON-TBD allocated value. It was necessary to choose some OID value, so I chose the first not-allocated code point.

This non-normative example demonstrates an otherName with a name form of BundleEID to encode the Node ID "dtn://example/".

The hexadecimal form of the DER encoding of the otherName is:

a01c06082b0601050507080ba010160e64746e3a2f2f6578616d706c652f

And the text decoding in Figure 28 is an output of Peter Gutmann's "dumpasn1" program.

```
0 28: [0] {
2  8:  OBJECT IDENTIFIER '1 3 6 1 5 5 7 8 11'
12 16:  [0] {
14 14:    IA5String 'dtn://example/'
      :    }
      :  }
```

Figure 28: Visualized decoding of the on-bundleEID

Authors' Addresses

Brian Sipos
RKF Engineering Solutions, LLC
7500 Old Georgetown Road
Suite 1275
Bethesda, MD 20814-6198
United States of America

Email: brian.sipos+ietf@gmail.com

Michael Demmer
University of California, Berkeley
Computer Science Division
445 Soda Hall
Berkeley, CA 94720-1776
United States of America

Email: demmer@cs.berkeley.edu

Joerg Ott
Aalto University
Department of Communications and Networking
PO Box 13000
FI-02015 Aalto
Finland

Email: ott@in.tum.de

Simon Perreault
Quebec QC
Canada

Email: simon@per.reau.lt