

Delay-Tolerant Networking
Internet-Draft
Intended status: Informational
Expires: September 14, 2017

E. Birrane
Johns Hopkins Applied Physics Laboratory
March 13, 2017

Asynchronous Management Architecture
draft-birrane-dtn-ama-05

Abstract

This document describes an asynchronous management architecture (AMA) suitable for providing application-level network management services in a challenged networking environment. Challenged networks are those that require fault protection, configuration, and performance reporting while unable to provide humans-in-the-loop with synchronous feedback or otherwise preserve transport-layer sessions. In such a context, networks must exhibit behavior that is both determinable and autonomous while maintaining compatibility with existing network management protocols and operational concepts.

Status of This Memo

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

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

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

This Internet-Draft will expire on September 14, 2017.

Copyright Notice

Copyright (c) 2017 IETF Trust and the persons identified as the document authors. All rights reserved.

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

to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction	3
1.1. Purpose	3
1.2. Scope	3
1.3. Requirements Language	4
1.4. Organization	4
2. Terminology	5
3. Motivation	6
3.1. Challenged Networks	6
3.2. Current Approaches and Their Limitations	7
4. Service Definitions	8
4.1. Configuration	9
4.2. Reporting	9
4.3. Autonomous Parameterized Procedure Calls	10
4.4. Administration	10
5. Desirable Properties	11
5.1. Intelligent Push of Information	11
5.2. Minimize Message Size Not Node Processing	12
5.3. Absolute Data Identification	12
5.4. Custom Data Definition	12
5.5. Autonomous Operation	13
6. Roles and Responsibilities	13
6.1. Agent Responsibilities	14
6.2. Manager Responsibilities	15
7. Logical Data Model	15
7.1. EDDs, VARs, and Reporting	16
7.2. Controls and Macros	17
7.3. Rules	17
7.4. Operators and Literals	18
8. System Model	18
8.1. Control and Data Flows	18
8.2. Control Flow by Role	19
8.2.1. Notation	19
8.2.2. Serialized Management	20
8.2.3. Multiplexed Management	21
8.2.4. Data Fusion	23
9. IANA Considerations	24
10. Security Considerations	24
11. Informative References	24
Author's Address	25

1. Introduction

The Asynchronous Management Architecture (AMA) provides application-layer network management services over links where delivery delays prevent timely communications between a network operator and a managed device. These delays may be caused by long signal propagations or frequent link disruptions (such as described in [RFC4838]) or by non-environmental factors such as unavailability of network operators, administrative delays, or delays caused by quality-of-service prioritizations and service-level agreements.

1.1. Purpose

This document describes the motivation, service definitions, desirable properties, roles/responsibilities, system model, and logical data model that form the AMA. These descriptions should be of sufficient specificity that implementations conformant to this architecture will operate successfully in a challenged networking environment.

This document is not a prescriptive standardization of a physical data model or protocol. Instead, it serves as informative guidance to authors of such models and protocols.

An AMA is necessary as the assumptions inherent to the architecture and design of synchronous management tools and techniques are not valid in challenged network scenarios. In these scenarios, synchronous approaches either patiently wait for periods of bi-directional connectivity or require the investment of significant time and resources to evolve a challenged network into a well-connected, low-latency network. In some cases such evolution is merely a costly way to over-resource a network. In other cases, such evolution is impossible given physical limitations imposed by signal propagation delays, power, transmission technologies, and other phenomena. Asynchronous management of asynchronous networks enables large-scale deployments, distributed technical capabilities, and reduced deployment and operations costs.

The rationale and motivation for asynchronous management is captured in [BIRrane1], [BIRrane2],[BIRrane3]. The properties and feasibility of such a system are taken from prototyping work done in accordance with [I-D.irtf-dtnrg-dtnmp].

1.2. Scope

It is assumed that any challenged network where network management would be usefully applied supports basic services (where necessary) such as naming, addressing, integrity, confidentiality,

authentication, fragmentation, and traditional network/session layer functions. Therefore, these items are outside of the scope of the AMA and not covered in this document.

While possible that a challenged network may interface with an unchallenged network, this document does not address the concept of network management compatibility with synchronous approaches.

1.3. 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].

1.4. Organization

The remainder of this document is organized into seven sections that, together, describe an AMA suitable for enterprise management of asynchronous networks: terminology, motivation, service definitions, desirable properties, roles/responsibilities, logical data model, and system model. The description of each section is as follows.

- o Terminology - This section identifies those terms critical to understanding the proper operation of the AMA. Whenever possible, these terms align in both word selection and meaning with their analogs from other management protocols.
- o Motivation - This section provides an overall motivation for this work as providing a novel and useful alternative to current network management approaches. Specifically, this section describes common network functions and how synchronous mechanisms fail to provide these functions in an asynchronous environment.
- o Service Definitions - This section defines asynchronous network management services in terms of terminology, scope, and impact.
- o Desirable Properties - This section identifies the properties to which an asynchronous management system should adhere to effectively implement service definitions in an asynchronous environment. These properties guide the subsequent definition of the system and logical models that comprise the AMA.
- o Roles and Responsibilities - This section identifies the roles in the AMA and their associated responsibilities. It provides the terminology and context for discussing how network management services interact.

- o Logical Data Model - This section describes the kinds of data that should be represented in deployment asynchronous management system.
- o System Model - This section describes data flows amongst various defined Actor roles. These flows capture how the AMA system works to provide asynchronous network management services in accordance with defined desirable properties.

2. Terminology

- o Actor - A software service running on either managed or managing devices for the purpose of implementing management protocols between such devices. Actors may implement the "Manager" role, "Agent" role, or both.
- o Agent Role (or Agent) - The role associated with a managed device, responsible for reporting performance data, enforcing administrative policies, and accepting/performing actions. Agents exchange information with Managers operating either on the same device or on a remote managing device.
- o Externally Defined Data (EDD) - Information made available to an Agent by a managed device, but not computed directly by the Agent.
- o Variable (VAR) - Information that is computed by an Agent, typically as a function of EDD values and/or other Variables.
- o Controls (CTRLs) - Operations that may be undertaken by an Actor to change the behavior, configuration, or state of an application or protocol managed by an AMP.
- o Literals (LIT) - Constants, enumerations, and other immutable definitions.
- o Macros - A named, ordered collection of Controls.
- o Manager - A role associated with a managing device responsible for configuring the behavior of, and receiving information from, Agents. Managers interact with one or more Agents located on the same device and/or on remote devices in the network.
- o Operator (OP) - The enumeration and specification of a mathematical function used to calculate computed data definitions and construct expressions to calculate state.

- o Report (RPT) - A named, typed, ordered collection of data values gathered by one or more Agents and provided to one or more Managers.
- o Rule - A unit of autonomous specification that provides a stimulus-response relationship between time or state on an Agent and the Controls to be run as a result of that time or state.

3. Motivation

Challenged networks, to include networks challenged by administrative or policy delays, cannot guarantee capabilities required to enable synchronous management techniques. These capabilities include high-rate, highly-available data, round-trip data exchange, and operators "in-the-loop". The inability of current approaches to provide network management services in a challenged network motivates the need for a new network management architecture focused on asynchronous, open-loop, autonomous control of network components.

3.1. Challenged Networks

A growing variety of link-challenged networks support packetization to increase data communications reliability without otherwise guaranteeing a simultaneous end-to-end path. Examples of such networks include Mobile Ad-Hoc Networks (MANets), Vehicular Ad-Hoc Networks (VANets), Space-Terrestrial Internetworks (STINTs), and heterogeneous networking overlays. Links in such networks are often unavailable due to attenuations, propagation delays, occultation, and other limitations imposed by energy and mass considerations. Data communications in such networks rely on store-and-forward and other queueing strategies to wait for the connectivity necessary to usefully advance a packet along its route.

Similarly, there also exist well-resourced networks that incur high message delivery delays due to non-environmental limitations. For example, networks whose operations centers are understaffed or where data volume and management requirements exceed the real-time cognitive load of operators or the associated operations console software support. Also, networks where policy restricts user access to existing bandwidth creates situations functionally similar to link disruption and delay.

Independent of the reason, when a node experiences an inability to communicate it must rely on autonomous mechanisms to ensure its safe operation and ability to usefully re-join the network at a later time. In cases of sparsely-populated networks, there may never be a practical concept of "the connected network" as most nodes may be disconnected most of the time. In such environments, defining a

network in terms of instantaneous connectivity becomes impractical or impossible.

Specifically, challenged networks exhibit the following properties that may violate assumptions built into current approaches to synchronous network management.

- o Links may be uni-directional.
- o Bi-directional links may have asymmetric data rates.
- o No end-to-end path is guaranteed to exist at any given time between any two nodes.
- o Round-trip communications between any two nodes within any given time window may be impossible.

3.2. Current Approaches and Their Limitations

Network management tools in unchallenged networks provide mechanisms for communicating locally-collected data from Agents to Managers, typically using a "pull" mechanism where data must be explicitly requested by a Manager in order to be transmitted by an Agent.

Management approaches that rely on timely data exchange, such as those that rely on negotiated sessions or other synchronized acknowledgment, do not function in challenged network environments. Familiar examples of TCP/IP based management via closed-loop, synchronous messaging do not work when network disruptions increase in frequency and severity. While no protocol delivers data in the absence of a networking link, protocols that eliminate or drastically reduce overhead and end-point coordination require smaller transmission windows and continue to function when confronted with scaling delays and disruptions in the network.

A legacy method for management in unchallenged networks today is the Simple Network Management Protocol (SNMP) [RFC3416]. SNMP utilizes a request/response model to set and retrieve data values such as host identifiers, link utilizations, error rates, and counters between application software on Agents and Managers. Data may be directly sampled or consolidated into representative statistics. Additionally, SNMP supports a model for asynchronous notification messages, called traps, based on predefined triggering events. Thus, Managers can query Agents for status information, send new configurations, and be informed when specific events have occurred. Traps and queryable data are defined in one or more Managed Information Bases (MIBs) which define the information for a particular data standard, protocol, device, or application.

While there is a large installation base for SNMP there are several aspects of the protocol that make it inappropriate for use in a challenged networking environment. SNMP relies on sessions with low round-trip latency to support its "pull" model. The SNMP trap model provides some Agent-side processing, but with very low fidelity and traps are typically "fire and forget" requiring the underlying transport to support reliable, in-order message delivery. Adaptive modifications to SNMP to support challenged networks would alter the basic function of the protocol (data models, control flows, and syntax) so as to be functionally incompatible with existing SNMP installations. Therefore, this approach is not suitable for an asynchronous network management system.

The Network Configuration Protocol (NETCONF) provides device-level configuration capabilities [RFC6241] to replace vendor-specific command line interface (CLI) configuration software. The XML-based protocol provides a remote procedure call (RPC) syntax such that any exposed functionality on an Agent can be exercised via a software application interface. NETCONF places no specific functional requirements or constraints on the capabilities of the Agent, which makes it a very flexible tool for configuring a homogeneous network of devices.

NETCONF places specific constraints on any underlying transport protocol: a long-lived, reliable, low-latency sequenced data delivery session. This is a fundamental requirement given the RPC-nature of the operating concept, and it is unsustainable in a challenged network. Aspects of the data modeling associated with NETCONF may apply to an asynchronous network management system, such that some modeling tools may be used, even if the network control plane cannot.

Just as the concept of a loosely-confederated set of nodes changes the definition of a network, it also changes the operational concept of what it means to manage a network. When a network stops being a single entity exhibiting a single behavior, "network management" becomes large-scale "node management". Individual nodes must share the burden of implementing desirable behavior without reliance on a single oracle of configuration or other coordinating function such as an operator-in-the-loop.

4. Service Definitions

This section identifies the services that must exist between Managers and Agents within an AMA. These services include configuration, reporting, parameterized control, and administration.

4.1. Configuration

Configuration services update Agent data associated with managed applications and protocols. Some configuration data might be defined in the context of an application or protocol, such that any network using that application or protocol would understand that data. Other configuration data may be defined tactically for use in a specific network deployment and not available to other networks even if they use the same applications or protocols.

New configurations received by an Agent must be validated to ensure that they do not conflict with other configurations or would otherwise prevent the Agent from effectively working with other Actors in its region. With no guarantee of round-trip data exchange, Agents cannot rely on remote Managers to correct erroneous or stale configurations from harming the flow of data through a challenged network.

Examples of configuration service behavior include the following.

- o Creating a new datum as a function of other well-known data:
 $C = A + B$.
- o Creating a new report as a unique, ordered collection of known data:
 $RPT = \{A, B, C\}$.
- o Storing pre-defined, parameterized responses to potential future conditions:
`IF (X > 3) THEN RUN CMD(PARM).`

4.2. Reporting

Reporting services populate report templates with values collected or computed by an Agent. The resultant reports are sent to one or more Managers by the Agent. The term "reporting" is used in place of the term "monitoring", as monitoring implies a timeliness and regularity that cannot be guaranteed by a challenged network. Reports sent by an Agent provide best-effort information to receiving Managers.

Since a Manager is not actively "monitoring" an Agent, the Agent must make its own determination on when to send what Reports based on its own local time and state information. Agents should produce Reports of varying fidelity and with varying frequency based on thresholds and other information set as part of configuration services.

Examples of reporting service behavior include the following.

- o Generate Report R1 every hour (time-based production).
- o Generate Report R2 when $X > 3$ (state-based production).

4.3. Autonomous Parameterized Procedure Calls

Similar to an RPC call, some mechanism MUST exist to allow a procedure to be run on an Agent to effect behavior or otherwise change the Agent's internal state. Since there is no guarantee that a Manager will be in contact with an Agent at any given time, the decisions of whether and when a procedure should be run MUST be made locally and autonomously by the Agent. Two types of automation triggers are identified in the AMA: triggers based on the general state of the Agent and triggers based on an Agent's notion of time. As such, the autonomous execution of procedures can be viewed as a stimulus-response system, where the stimulus is the positive evaluation of a state or time based predicate and the response is the function to be executed.

The autonomous nature of procedure execution by an Agent implies that the full suite of information necessary to run a procedure may not be known by a Manager in advance. To address this situation, a parameterization mechanism MUST be available so that required data can be provided at the time of execution on the Agent rather than at the time of definition/configuration by the Manager.

Autonomous, parameterized procedure calls provide a powerful mechanism for Managers to "manage" an Agent asynchronously during periods of no communication by pre-configuring responses to events that may be encountered by the Agent at a future time.

Examples of potential behavior include the following.

- o Updating local routing information based on instantaneous link analysis.
- o Managing storage on the device to enforce quotas.
- o Applying or modifying local security policy.

4.4. Administration

Administration services enforce the potentially complex mapping of configuration, reporting, and control services amongst Agents and Managers in the network. Fine-grained access control specifying which Managers may apply which services to which Agents may be necessary in networks dealing with multiple administrative entities or overlay networks crossing multiple administrative boundaries.

Whitelists, blacklists, key-based infrastructures, or other schemes may be used for this purpose.

Examples of administration service behavior include the following.

- o Agent A1 only Sends reports for Protocol P1 to Manager M1.
- o Agent A2 only accepts a configurations for Application Y from Managers M2 and M3.
- o Agent A3 accepts services from any Manager providing the proper authentication token.

Note that the administrative enforcement of access control is different from security services provided by the networking stack carrying AMP messages.

5. Desirable Properties

This section describes those design properties that are desirable when defining an architecture that must operate across challenged links in a network. These properties ensure that network management capabilities are retained even as delays and disruptions in the network scale. Ultimately, these properties are the driving design principles for the AMA.

5.1. Intelligent Push of Information

Pull management mechanisms require that a Manager send a query to an Agent and then wait for the response to that query. This practice implies a control-session between entities and increases the overall message traffic in the network. Challenged networks cannot guarantee timely roundtrip data-exchange and, in extreme cases, are comprised solely of uni-directional links. Therefore, pull mechanisms must be avoided in favor of push mechanisms.

Push mechanisms, in this context, refer to Agents making their own determinations relating to the information that should be sent to Managers. Such mechanisms do not require round-trip communications as Managers do not request each reporting instance; Managers need only request once, in advance, that information be produced in accordance with a pre-determined schedule or in response to a pre-defined state on the Agent. In this way information is "pushed" from Agents to Managers and the push is "intelligent" because it is based on some internal evaluation performed by the Agent.

5.2. Minimize Message Size Not Node Processing

Protocol designers must balance message size versus message processing time at sending and receiving nodes. Verbose representations of data simplify node processing whereas compact representations require additional activities to generate/parse the compacted message. There is no asynchronous management advantage to minimizing node processing time in a challenged network. However, there is a significant advantage to smaller message sizes in such networks. Compact messages require smaller periods of viable transmission for communication, incur less re-transmission cost, and consume less resources when persistently stored en-route in the network. AMPs should minimize PDUs whenever practical, to include packing and unpacking binary data, variable-length fields, and pre-configured data definitions.

5.3. Absolute Data Identification

Elements within the management system must be uniquely identifiable so that they can be individually manipulated. Identification schemes that are relative to system configuration make data exchange between Agents and Managers difficult as system configurations may change faster than nodes can communicate.

Consider the following common technique for approximating an associative array lookup. A manager wishing to do an associative lookup for some key K1 will (1) query a list of array keys from the agent, (2) find the key that matches K1 and infer the index of K1 from the returned key list, and (3) query the discovered index on the agent to retrieve the desired data.

Ignoring the inefficiency of two pull requests, this mechanism fails when the Agent changes its key-index mapping between the first and second query. Rather than constructing an artificial mapping from K1 to an index, an AMP must provide an absolute mechanism to lookup the value K1 without an abstraction between the Agent and Manager.

5.4. Custom Data Definition

Custom definition of new data from existing data (such as through data fusion, averaging, sampling, or other mechanisms) provides the ability to communicate desired information in as compact a form as possible. Specifically, an Agent should not be required to transmit a large data set for a Manager that only wishes to calculate a smaller, inferred data set. The Agent should calculate the smaller data set on its own and transmit that instead. Since the identification of custom data sets is likely to occur in the context

of a specific network deployment, AMPs must provide a mechanism for their definition.

5.5. Autonomous Operation

AMA network functions must be achievable using only knowledge local to the Agent. Rather than directly controlling an Agent, a Manager configures an engine of the Agent to take its own action under the appropriate conditions in accordance with the Agent's notion of local state and time.

Such an engine may be used for simple automation of pre-defined tasks or to support semi-autonomous behavior in determining when to run tasks and how to configure or parameterize tasks when they are run. Wholly autonomous operations MAY be supported where required. Generally, autonomous operations should provide the following benefits.

- o Distributed Operation - The concept of pre-configuration allows the Agent to operate without regular contact with Managers in the system. The initial configuration (and periodic update) of the system remains difficult in a challenged network, but an initial synchronization on stimuli and responses drastically reduces needs for centralized operations.
- o Deterministic Behavior - Such behavior is necessary in critical operational systems where the actions of a platform must be well understood even in the absence of an operator in the loop. Depending on the types of stimuli and responses, these systems may be considered simple automation or semi-autonomous behavior, both of which imply the ability of a frequently-out-of-contact Manager to better predict the state of an Agent than if controls were to be run by an independent, fully autonomous system.
- o Engine-Based Behavior - Several operational systems are unable to deploy "mobile code" based solutions due to network bandwidth, memory or processor loading, or security concerns. Engine-based approaches are preferred as they can be flexible without incurring a set of problematic requirements or concerns.

6. Roles and Responsibilities

By definition, Agents reside on managed devices and Managers reside on managing devices. This section describes how these roles participate in the network management functions outlined in the prior section.

6.1. Agent Responsibilities

Application Support

Agents MUST collect all data, execute all procedures, populate all reports and run operations required by each application which the Agent claims to manage. Agents MUST report supported applications so that Managers in a network understands what information is understood by what Agent.

Local Data Collection

Agents MUST collect from local firmware (or other on-board mechanisms) and report all data defined for the management of applications for which they have been configured.

Autonomous Control

Agents MUST determine, without Manager intervention, whether a procedure should be invoked. Agents MAY also invoke procedures on other devices for which they act as proxy.

User Data Definition

Agents MUST provide mechanisms for operators in the network to use configuration services to create customized data definitions in the context of a specific network or network use-case. Agents MUST allow for the creation, listing, and removal of such definitions in accordance with whatever security models are deployed within the particular network.

Where applicable, Agents MUST verify the validity of these definitions when they are configured and respond in a way consistent with the logging/error-handling policies of the Agent and the network.

Autonomous Reporting

Agents MUST determine, without real-time Manager intervention, whether and when to populate and transmit a given report targeted to one or more Managers in the network.

Consolidate Messages

Agents SHOULD produce as few messages as possible when sending information. For example, rather than sending multiple messages, each with one report to a Manager, an Agent SHOULD prefer to send a single message containing multiple reports.

Regional Proxy

Agents MAY perform any of their responsibilities on behalf of other network nodes that, themselves, do not have an Agent.

In such a configuration, the Agent acts as a proxy for these other network nodes.

6.2. Manager Responsibilities

Agent Capabilities Mapping

Managers MUST understand what applications are managed by the various Agents with which they communicate. Managers should not attempt to request, invoke, or refer to application information for applications not managed by an Agent.

Data Collection

Managers MUST receive information from Agents by asynchronously configuring the production of reports and then waiting for, and collecting, responses from Agents over time. Managers MAY try to detect conditions where Agent information has not been received within operationally relevant timespans and react in accordance with network policy.

Custom Definitions

Managers should provide the ability to define custom data definitions. Any custom definitions MUST be transmitted to appropriate Agents and these definitions MUST be remembered to interpret the reporting of these custom values from Agents in the future.

Data Translation

Managers should provide some interface to other network management protocols. Managers MAY accomplish this by accumulating a repository of push-data from high-latency parts of the network from which data may be pulled by low-latency parts of the network.

Data Fusion

Managers MAY support the fusion of data from multiple Agents with the purpose of transmitting fused data results to other Managers within the network. Managers MAY receive fused reports from other Managers pursuant to appropriate security and administrative configurations.

7. Logical Data Model

The AMA logical data model captures the types of information that should be collected and exchanged to implement necessary roles and responsibilities. The data model presented in this section does not presuppose a specific mapping to a physical data model or encoding technique; it is included to provide a way to logically reason about

the types of data that should be exchanged in an asynchronously managed network.

The elements of the AMA logical data model are described as follows.

7.1. EDDs, VARs, and Reporting

There are three fundamental representations of data in the AMA: (1) data which are sampled/calculated external to the network management system, (2) data which are calculated internal to the network management system, and (3) ordered collections of data items used for reporting.

Data that is sampled/calculated external to the network management system is defined as "externally defined data" (EDD). EDD values represent the most useful information in the management system as they are provided by the applications or protocol being managed on the Agent. It is RECOMMENDED that EDD values be strongly typed to avoid issues with interpreting the data value. It is also RECOMMENDED that the timeliness/staleness of the data value be considered when using the data in the context of autonomous action on the Agent.

Data that is calculated internal to the network management system is defined as a "variable" (VAR). VARs allow the creation of new data values for use in the network management system. New value definitions are useful for storing user-defined information, storing the results of complex calculations for easier re-use, and providing a mechanism for combining information from multiple external sources. It is RECOMMENDED that VARs be strongly typed to avoid issues with interpreting the data value. In cases where a VAR definition relies on other VAR definitions, mechanisms to prevent circular references MUST be included in any actual data model or implementation.

Ordered collections of EDD values and VARs should be produced by Agents and sent to Managers as a way of communicating Agent state. Such an ordered collection is called a "report" (RPT). It is RECOMMENDED that the structure of a RPT be given in a template that can be synchronized between an Agent and a Manager so that RPTs themselves do not need to be self-describing. A RPT may include EDD values, VARs, and also other RPTs. In cases where a RPT includes another RPT, mechanisms to prevent circular references MUST be included in any actual data model or implementation.

7.2. Controls and Macros

Low-latency, high-availability approaches to network management use mechanisms such as (or similar to) remote procedure calls (RPCs) to cause some action to be performed on an Agent. The AMA requires similar capabilities, though without requiring that the Manager be in the processing loop of the Agent.

A "control" (CTRL) represents a parameterized, pre-defined procedure that can be run on an Agent. CTRLs do not have a return code as there is not the same concept of sequential execution in an asynchronous model. Parameters can be provided when running a command from a Manager, pre-configured as part of an autonomy response on the Agent, or auto-generated as needed on the Agent. The success or failure of a control MAY be inferred by reports generated for that purpose.

Often, a series of controls must be executed in concert to achieve a particular outcome. A "macro" (MACRO) represents an ordered collection of controls (or other macros). In cases where a MACRO includes another MACRO, mechanisms to prevent circular references and maximum nesting levels MUST be included in any actual data model or implementation.

7.3. Rules

The AMA data model contains EDD values and VARs that capture the state of applications on an Agent. The model also contains controls and macros to perform actions on an Agent. A mechanism is needed to relate these two capabilities; to perform an action on the Agent in response to the state of the Agent.

One way of mapping Agent state to Agent actions is via a stimulus-response system. A "rule" represents a stimulus-response pairing in the following form.

IF predicate THEN response

The predicate is a logical expression that evaluates to true if the rule stimulus is present and evaluates to false otherwise. The response may be any control or macro known to the Agent. An example of a time-based predicate is to perform some activity every 24 hours (e.g., $((\text{CUR_TIME} - \text{START_TIME}) \% 24\text{Hrs}) == 0$). An example of a state-based predicate is to perform some activity if a given EDD value exceeds a pre-defined threshold such as a measured temperature exceeds 80 degrees centigrade (e.g., $(\text{TEMP} > 80.0)$)

Rules should be allowed to construct their stimuli from the full set of EDD values and VARs available to the network management system. Similarly, macro responses should be allowed to include controls from

all applications known by the Agent. This enables an expressive capability to have multiple applications monitored and managed by the Agent.

7.4. Operators and Literals

Actions such as computing a VAR value or describing a rule predicate require calculating mathematical expressions. An element of an expression will be one of four types of data: an EDD value, a VAR value, a mathematical operations, and literal values.

An "operator" (OP) represents a mathematical operation in an expression. OPs should support multiple operands based on the operation supported. A common set of OPs SHOULD be defined for any Agent and systems MAY choose to allow individual applications to define new OPs to assist in the generation of new VAR values and predicates for managing that application. OPs may be simple binary operations such as "A + B" or more complex functions such as sin(A) or avg(A,B,C,D).

A "literal" (LIT) represents a constant value, such as simple numbers (e.g., 4), well-known mathematical numbers (e.g., PI, E), or other useful data such as Epoch times. LITs should be strongly typed to avoid any misinterpretation of their data value.

8. System Model

This section describes the notional data flows and control flows that illustrate how Managers and Agents within an AMA cooperate to perform network management services.

8.1. Control and Data Flows

The AMA identifies three significant data flows: control flows from Managers to Agents, reports flows from Agents to Managers, and fusion reports from Managers to other Managers. These data flows are illustrated in Figure 1.

AMA Control and Data Flows

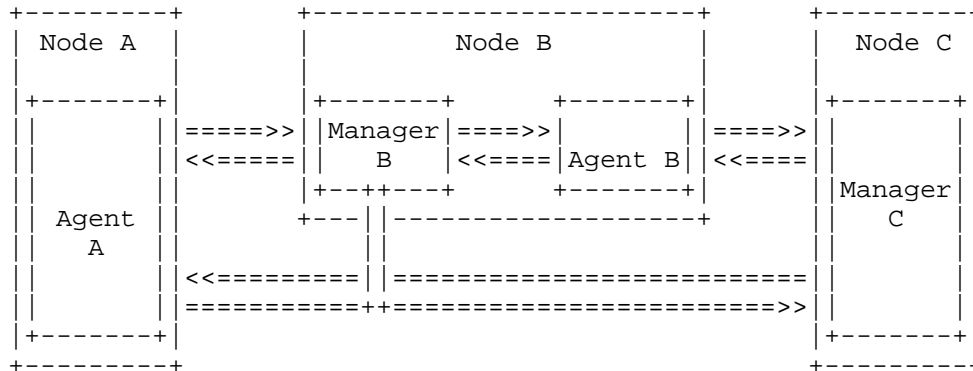


Figure 1

In this data flow, the Agent on node A receives Controls from Managers on nodes B and C, and replies with Report Entries back to these Managers. Similarly, the Agent on node B interacts with the local Manager on node B and the remote Manager on node C. Finally, the Manager on node B may fuse Report Entries received from Agents at nodes A and B and send these fused Report Entries back to the Manager on node C.

From this figure it is clear that there exist many-to-many relationships amongst Managers, amongst Agents, and between Agents and Managers. Note that Agents and Managers are roles, not necessarily differing software applications. Node A may represent a single software application fulfilling only the Agent role, whereas node B may have a single software application fulfilling both the Agent and Manager roles. The specifics of how these roles are realized is an implementation matter.

8.2. Control Flow by Role

This section describes three common configurations of Agents and Managers and the flow of messages between them. These configurations involve local and remote management and data fusion.

8.2.1. Notation

The notation outlined in Table 1 describes the types of control messages exchanged between Agents and Managers.

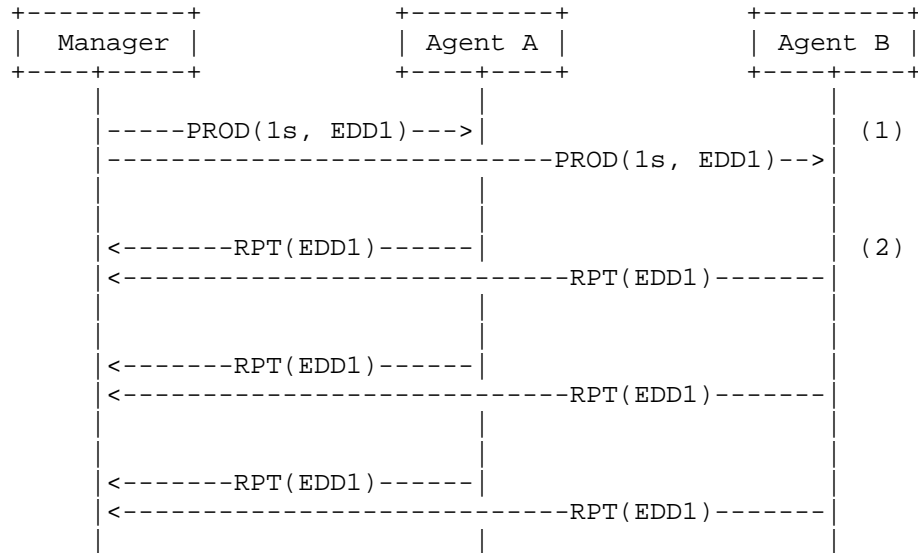
Term	Definition	Example
EDD#	EDD definition.	EDD1
V#	Custom data definition.	V1 = EDD1 + V0.
DEF([ACL], ID,EXPR)	Define id from expression. Allow managers in access control list (ACL) to request this id.	DEF([*], V1, EDD1 + EDD2)
PROD(P,ID)	Produce ID according to predicate P. P may be a time period (1s) or an expression (EDD1 > 10).	PROD(1s, EDD1)
RPT(ID)	A report identified by ID.	RPT(EDD1)

Table 1: Terminology

8.2.2. Serialized Management

This is a nominal configuration of network management where a Manager interacts with a set of Agents. The control flows for this are outlined in Figure 2.

Serialized Management Control Flow



In a simple network, a Manager interacts with multiple Agents.

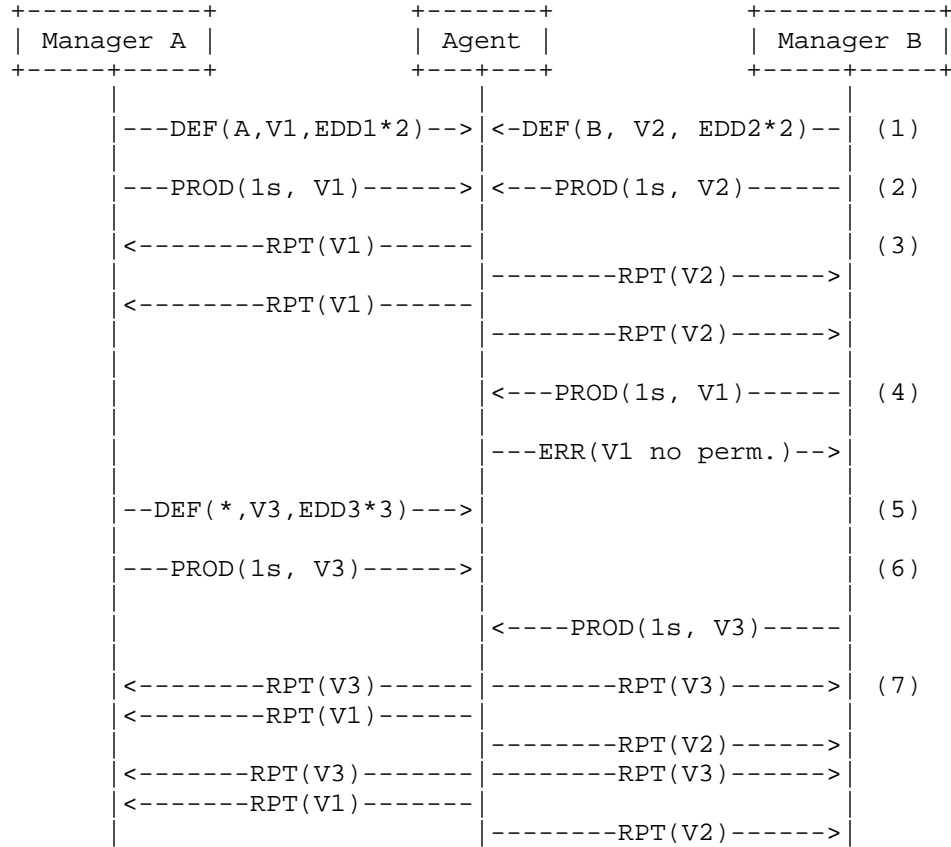
Figure 2

In this figure, the Manager configures Agents A and B to produce EDD1 every second in (1). At some point in the future, upon receiving and configuring this message, Agents A and B then build a Report Entry containing EDD1 and send those reports back to the Manager in (2).

8.2.3. Multiplexed Management

Networks spanning multiple administrative domains may require multiple Managers (for example, one per domain). When a Manager defines custom Reports/Variables to an Agent, that definition may be tagged with an access control list (ACL) to limit what other Managers will be privy to this information. Managers in such networks should synchronize with those other Managers granted access to their custom data definitions. When Agents generate messages, they **MUST** only send messages to Managers according to these ACLs, if present. The control flows in this scenario are outlined in Figure 3.

Multiplexed Management Control Flow



Complex networks require multiple Managers interfacing with Agents.

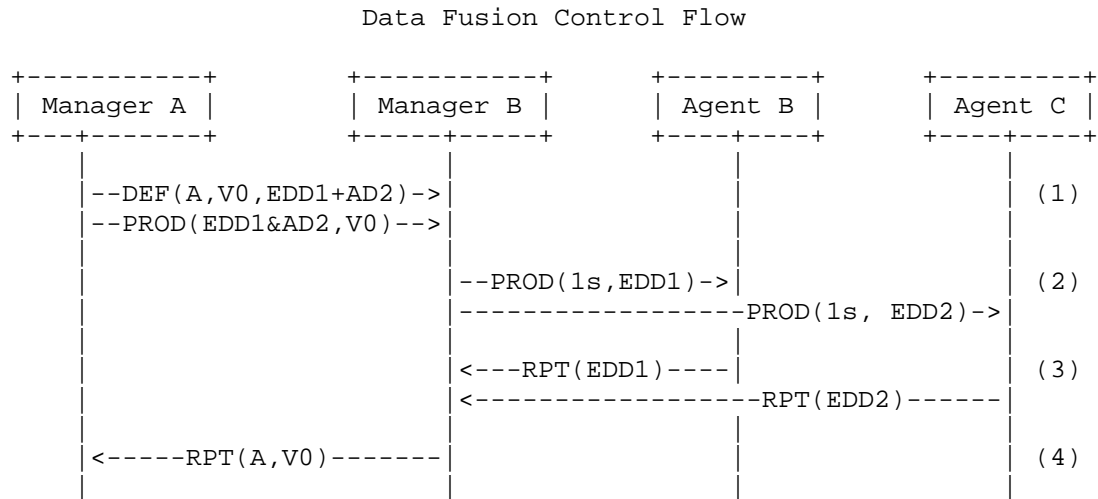
Figure 3

In more complex networks, any Manager may choose to define custom Reports and Variables, and Agents may need to accept such definitions from multiple Managers. Variable definitions may include an ACL that describes who may query and otherwise understand these definitions. In (1), Manager A defines V1 only for A while Manager B defines V2 only for B. Managers may, then, request the production of Report Entries containing these definitions, as shown in (2). Agents produce different data for different Managers in accordance with configured production rules, as shown in (3). If a Manager requests the production of a custom definition for which the Manager has no permissions, a response consistent with the configured logging policy on the Agent should be implemented, as shown in (4). Alternatively,

as shown in (5), a Manager may define custom data with no restrictions allowing all other Managers to request and use this definition. This allows all Managers to request the production of Report Entries containing this definition, shown in (6) and have all Managers receive this and other data going forward, as shown in (7).

8.2.4. Data Fusion

In some networks, Agents do not individually transmit their data to a Manager, preferring instead to fuse reporting data with local nodes prior to transmission. This approach reduces the number and size of messages in the network and reduces overall transmission energy expenditure. The AMA supports fusion of NM reports by co-locating Agents and Managers on nodes and offloading fusion activities to the Manager. This process is illustrated in Figure 4.



Data fusion occurs amongst Managers in the network.

Figure 4

In this example, Manager A requires the production of a Variable V0, from node B, as shown in (1). The Manager role understands what data is available from what agents in the subnetwork local to B, understanding that EDD1 is available locally and EDD2 is available remotely. Production messages are produced in (2) and data collected in (3). This allows the Manager at node B to fuse the collected Report Entries into V0 and return it in (4). While a trivial example, the mechanism of associating fusion with the Manager function rather than the Agent function scales with fusion complexity, though it is important to reiterate that Agent and

Manager designations are roles, not individual software components. There may be a single software application running on node B implementing both Manager B and Agent B roles.

9. IANA Considerations

At this time, this protocol has no fields registered by IANA.

10. Security Considerations

Security within an AMA MUST exist in two layers: transport layer security and access control.

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

Finer grain application security is done via ACLs which are defined via configuration messages and implementation specific.

11. Informative References

[BIRrane1]

Birrane, E. and R. Cole, "Management of Disruption-Tolerant Networks: A Systems Engineering Approach", 2010.

[BIRrane2]

Birrane, E., Burleigh, S., and V. Cerf, "Defining Tolerance: Impacts of Delay and Disruption when Managing Challenged Networks", 2011.

[BIRrane3]

Birrane, E. and H. Kruse, "Delay-Tolerant Network Management: The Definition and Exchange of Infrastructure Information in High Delay Environments", 2011.

[I-D.irtf-dtnrg-dtnmp]

Birrane, E. and V. Ramachandran, "Delay Tolerant Network Management Protocol", draft-irtf-dtnrg-dtnmp-01 (work in progress), December 2014.

[RFC2119]

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

- [RFC3416] Presuhn, R., Ed., "Version 2 of the Protocol Operations for the Simple Network Management Protocol (SNMP)", STD 62, RFC 3416, DOI 10.17487/RFC3416, December 2002, <<http://www.rfc-editor.org/info/rfc3416>>.
- [RFC4838] Cerf, V., Burleigh, S., Hooke, A., Torgerson, L., Durst, R., Scott, K., Fall, K., and H. Weiss, "Delay-Tolerant Networking Architecture", RFC 4838, April 2007.
- [RFC6241] Enns, R., Ed., Bjorklund, M., Ed., Schoenwaelder, J., Ed., and A. Bierman, Ed., "Network Configuration Protocol (NETCONF)", RFC 6241, DOI 10.17487/RFC6241, June 2011, <<http://www.rfc-editor.org/info/rfc6241>>.

Author's Address

Edward J. Birrane
Johns Hopkins Applied Physics Laboratory

Email: Edward.Birrane@jhuapl.edu

Delay-Tolerant Networking Working Group
Internet Draft
Intended status: Standards Track
Expires: December 2017

S. Burleigh
JPL, Calif. Inst. Of Technology
K. Fall
Nefeli Networks, Inc.
E. Birrane
APL, Johns Hopkins University
June 22, 2017

Bundle Protocol
draft-ietf-dtn-bpbis-07.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 December 24, 2017.

Copyright Notice

Copyright (c) 2017 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 Internet Draft presents a specification for Bundle Protocol, adapted from the experimental Bundle Protocol specification developed by the Delay-Tolerant Networking Research group of the Internet Research Task Force and documented in RFC 5050.

Table of Contents

1. Introduction.....	3
2. Conventions used in this document.....	5
3. Service Description.....	5
3.1. Definitions.....	5
3.2. Discussion of BP concepts.....	9
3.3. Services Offered by Bundle Protocol Agents.....	11
4. Bundle Format.....	12
4.1. BP Fundamental Data Structures.....	12
4.1.1. CRC Type.....	12
4.1.2. CRC.....	13
4.1.3. Bundle Processing Control Flags.....	13
4.1.4. Block Processing Control Flags.....	14
4.1.5. Identifiers.....	15
4.1.5.1. Endpoint ID.....	15
4.1.5.2. Node ID.....	16
4.1.6. DTN Time.....	17
4.1.7. Creation Timestamp.....	17
4.1.8. Block-type-specific Data.....	17
4.2. Bundle Representation.....	17
4.2.1. Bundle.....	17
4.2.2. Primary Bundle Block.....	18
4.2.3. Canonical Bundle Block Format.....	19
4.3. Extension Blocks.....	20
4.3.1. Previous Node.....	21
4.3.2. Bundle Age.....	21
4.3.3. Hop Count.....	22
5. Bundle Processing.....	22
5.1. Generation of Administrative Records.....	23
5.2. Bundle Transmission.....	23
5.3. Bundle Dispatching.....	24
5.4. Bundle Forwarding.....	24
5.4.1. Forwarding Contraindicated.....	26
5.4.2. Forwarding Failed.....	26
5.5. Bundle Expiration.....	26

5.6. Bundle Reception.....	27
5.7. Local Bundle Delivery.....	27
5.8. Bundle Fragmentation.....	28
5.9. Application Data Unit Reassembly.....	30
5.10. Bundle Deletion.....	30
5.11. Discarding a Bundle.....	30
5.12. Canceling a Transmission.....	30
6. Administrative Record Processing.....	31
6.1. Administrative Records.....	31
6.1.1. Bundle Status Reports.....	32
6.2. Generation of Administrative Records.....	34
7. Services Required of the Convergence Layer.....	34
7.1. The Convergence Layer.....	34
7.2. Summary of Convergence Layer Services.....	35
8. Implementation Status.....	35
9. Security Considerations.....	36
10. IANA Considerations.....	38
11. References.....	38
11.1. Normative References.....	38
11.2. Informative References.....	38
12. Acknowledgments.....	39
13. Significant Changes from RFC 5050.....	39
Appendix A. For More Information.....	40
Appendix B. CDDL expression.....	41

1. Introduction

Since the publication of the Bundle Protocol Specification (Experimental RFC 5050) in 2007, the Delay-Tolerant Networking Bundle Protocol has been implemented in multiple programming languages and deployed to a wide variety of computing platforms. This implementation and deployment experience has identified opportunities for making the protocol simpler, more capable, and easier to use. The present document, standardizing the Bundle Protocol (BP), is adapted from RFC 5050 in that context.

This document describes version 7 of BP.

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

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

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

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

The interface between the bundle protocol and a specific underlying protocol is termed a "convergence layer adapter".

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

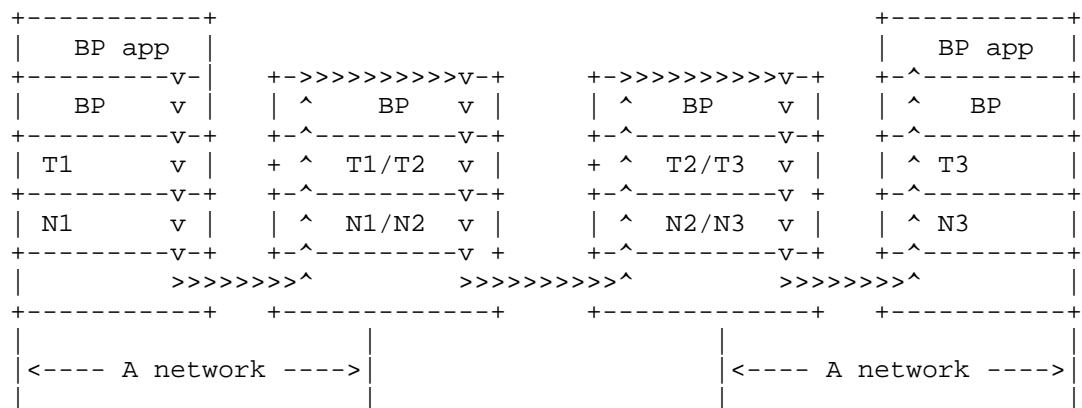


Figure 1: The Bundle Protocol in the Protocol Stack Model

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

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

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

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

2. Conventions used in this document

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

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

3. Service Description

3.1. Definitions

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

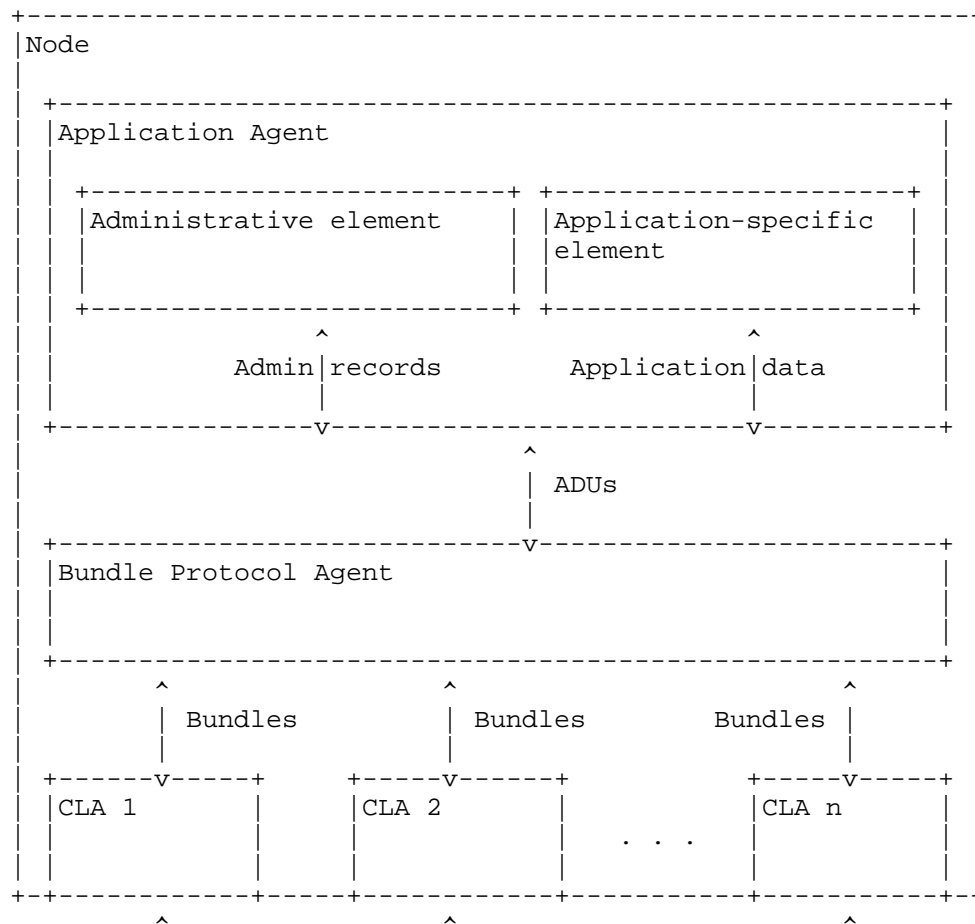
Block - A bundle protocol block is one of the protocol data structures that together constitute a well-formed bundle.

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

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

Fragment - A fragment is a bundle whose payload block contains a partial payload.

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



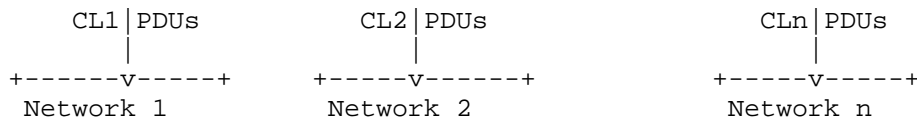


Figure 2: Components of a BP Node

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

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

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

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

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

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

Bundle endpoint - A bundle endpoint (or simply "endpoint") is a set of zero or more bundle nodes that all identify themselves for BP purposes by some common identifier, called a "bundle endpoint ID" (or, in this document, simply "endpoint ID"; endpoint IDs are described in detail in Section 4.4.1 below).

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

Registration - A registration is the state machine characterizing a given node's membership in a given endpoint. Any single registration has an associated delivery failure action as defined below and must at any time be in one of two states: Active or Passive.

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

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

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

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

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

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

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

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

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

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

3.2. Discussion of BP concepts

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

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

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

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

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

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

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

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

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

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

Singletons are the most familiar sort of endpoint, but in general the endpoint notion is meant to be broader. For example, the nodes in a sensor network might constitute a set of bundle nodes that identify themselves by a single common endpoint ID and thus form a single bundle endpoint. *Note* too that a given bundle node might identify itself by multiple endpoint IDs and thus be a member of multiple bundle endpoints.

The destination of every bundle is an endpoint, which may or may not be singleton. The source of every bundle is a node, identified by the endpoint ID for some singleton endpoint that contains that node.

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

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

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

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

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

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

3.3. Services Offered by Bundle Protocol Agents

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

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

4. Bundle Format

The format of bundles SHALL conform to the Concise Binary Object Representation (CBOR [RFC7049]).

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

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

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

4.1. BP Fundamental Data Structures

4.1.1. CRC Type

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

- . 0 indicates "no CRC is present."
- . 1 indicates "a CRC-16 (a.k.a., CRC-16-ANSI) is present."
- . 2 indicates "a standard IEEE 802.3 CRC-32 is present."

CRC type SHALL be represented as a CBOR unsigned integer.

4.1.2. CRC

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

When not omitted, the CRC SHALL be represented as sequence of two bytes (if CRC type is 1) or as a sequence of four bytes (if CRC type is 2).

4.1.3. Bundle Processing Control Flags

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

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

- . The bundle is a fragment. (Boolean)
- . The bundle's payload is an administrative record. (Boolean)
- . The bundle must not be fragmented. (Boolean)
- . The bundle's destination endpoint is a singleton. (Boolean)
- . Acknowledgment by the user application is requested. (Boolean)
- . Status time is requested in all status reports. (Boolean)
- . The bundle contains a "manifest" extension block. (Boolean)
- . Flags requesting types of status reports (all Boolean):
 - o Request reporting of bundle reception.
 - o Request reporting of bundle forwarding.
 - o Request reporting of bundle delivery.
 - o Request reporting of bundle deletion.

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

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

The bundle processing control flags SHALL be represented as a CBOR unsigned integer item containing a bit field of 16 bits indicating the control flag values as follows:

- . Bit 0 (the high-order bit, 0x8000): reserved.
- . Bit 1 (0x4000): reserved.
- . Bit 2 (0x2000): reserved.
- . Bit 3(0x1000): bundle deletion status reports are requested.
- . Bit 4(0x0800): bundle delivery status reports are requested.
- . Bit 5(0x0400): bundle forwarding status reports are requested.
- . Bit 6(0x0200): reserved.
- . Bit 7(0x0100): bundle reception status reports are requested.
- . Bit 8(0x0080): bundle contains a Manifest block.
- . Bit 9(0x0040): status time is requested in all status reports.
- . Bit 10(0x0020): user application acknowledgement is requested.
- . Bit 11(0x0010): destination is a singleton endpoint.
- . Bit 12(0x0008): reserved.
- . Bit 13(0x0004): bundle must not be fragmented.
- . Bit 14(0x0002): payload is an administrative record.
- . Bit 15 (the low-order bit, 0x0001: bundle is a fragment.

4.1.4. Block Processing Control Flags

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

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

- . This block must be replicated in every fragment. (Boolean)
- . Transmission of a status report is requested if this block can't be processed. (Boolean)
- . Block must be removed from the bundle if it can't be processed. (Boolean)

- . Bundle must be deleted if this block can't be processed.
(Boolean)

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

The block processing control flags SHALL be represented as a CBOR unsigned integer item containing a bit field of 8 bits indicating the control flag values as follows:

- . Bit 0 (the high-order bit, 0x80): reserved.
- . Bit 1 (0x40): reserved.
- . Bit 2(0x20): reserved.
- . Bit 3(0x10): reserved.
- . Bit 4(0x08): bundle must be deleted if block can't be processed.
- . Bit 5(0x04): transmission of a status report is requested if block can't be processed.
- . Bit 6(0x02): block must be removed from bundle if it can't be processed.
- . Bit 7(the low-order bit, 0x01): block must be replicated in every fragment.

4.1.5. Identifiers

4.1.5.1. Endpoint ID

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

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

The scheme identified by the < scheme name > in an endpoint ID is a set of syntactic and semantic rules that fully explain how to parse and interpret the SSP. The set of allowable schemes is effectively unlimited. Any scheme conforming to [URI REG] may be used in a bundle protocol endpoint ID.

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

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

Each BP endpoint ID (EID) SHALL be represented as a CBOR array comprising a 2-tuple.

The first item of the array SHALL be the code number identifying the endpoint's URI scheme [URI], as defined in the registry of URI scheme code numbers for Bundle Protocol maintained by IANA as described in Section 10. [URIREG]. Each URI scheme code number SHALL be represented as a CBOR unsigned integer.

The second item of the array SHALL be the applicable CBOR representation of the scheme-specific part (SSP) of the EID, defined as follows:

- . If the EID's URI scheme is "dtn" then the SSP SHALL be represented as a CBOR text string unless the EID's SSP is "none", in which case the SSP SHALL be represented as a CBOR unsigned integer with the value zero.
- . If the EID's URI scheme is "ipn" then the SSP SHALL be represented as a CBOR array comprising a 2-tuple. The first item of this array SHALL be the EID's node number represented as a CBOR unsigned integer. The second item of this array SHALL be the EID's service number represented as a CBOR unsigned integer.
- . Definitions of the CBOR representations of the SSPs of EIDs encoded in other URI schemes are included in the specifications defining those schemes.

4.1.5.2. Node ID

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

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

- . Every node MUST be a member of at least one singleton endpoint.
- . The EID of any singleton endpoint of which a node is a member MAY be used to identify that node. A "node ID" is an EID that is used in this way.
- . A node's membership in a given singleton endpoint MUST be sustained at least until the nominal operation of the Bundle

Protocol no longer depends on the identification of that node using that endpoint's ID.

4.1.6. DTN Time

A DTN time is an unsigned integer indicating a count of seconds since the start of the year 2000 on the Coordinated Universal Time (UTC) scale [UTC]. Each DTN time SHALL be represented as a CBOR unsigned integer item.

4.1.7. Creation Timestamp

Each creation timestamp SHALL be represented as a CBOR array item comprising a 2-tuple.

The first item of the array SHALL be a DTN time.

The second item of the array SHALL be the creation timestamp's sequence number, represented as a CBOR unsigned integer.

4.1.8. Block-type-specific Data

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

4.2. Bundle Representation

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

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

4.2.1. Bundle

Each bundle SHALL be represented as a CBOR indefinite-length array. The first item of this array SHALL be the CBOR representation of a Primary Block. Every other item of the array except the last SHALL be the CBOR representation of a Canonical Block. The last item of the array SHALL be a CBOR "break" stop code.

4.2.2. Primary Bundle Block

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

Each primary block SHALL be represented as a CBOR array; the number of elements in the array SHALL be 8 (if the bundle is not a fragment and CRC type is zero) or 9 (if the bundle is not a fragment and CRC type is non-zero) or 10 (if the bundle is a fragment and CRC type is zero) or 11 (if the bundle is a fragment and CRC-type is non-zero).

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

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

Bundle Processing Control Flags: The Bundle Processing Control Flags are discussed in Section 4.1.3. above.

CRC Type: CRC Type codes are discussed in Section 4.1.1. above.

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

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

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

Creation Timestamp: The creation timestamp is a pair of unsigned integers that, together with the source node ID and (if the bundle is a fragment) the fragment offset and payload length, serve to identify the bundle. The first of these integers is the bundle's creation time, while the second is the bundle's creation timestamp sequence number. Bundle creation time shall be the time - expressed in seconds since the start of the year 2000, on the Coordinated Universal Time (UTC) scale [UTC] - at which the transmission request was received that resulted in the creation of the bundle. Sequence count shall be the latest value (as of the time at which that

transmission request was received) of a monotonically increasing positive integer counter managed by the source node's bundle protocol agent that may be reset to zero whenever the current time advances by one second. For nodes that lack accurate clocks, it is recommended that bundle creation time be set to zero and that the counter used as the source of the bundle sequence count never be reset to zero. Note that, in general, the creation of two distinct bundles with the same source node ID and bundle creation timestamp may result in unexpected network behavior and/or suboptimal performance. The combination of source node ID and bundle creation timestamp serves to identify a single transmission request, enabling it to be acknowledged by the receiving application (provided the source node ID is not the null endpoint ID).

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

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

CRC: If and only if the value of the CRC type field of this Primary block is non-zero, a CRC **SHALL** be present in the primary block. The length and nature of the CRC **SHALL** be as indicated by the CRC type. The CRC **SHALL** be computed over the concatenation of all bytes (including CBOR "break" characters) of the primary block including the CRC field itself, which for this purpose **SHALL** be temporarily populated with the value zero.

4.2.3. Canonical Bundle Block Format

Every block other than the primary block (which blocks are termed "canonical" blocks) **SHALL** be represented as a CBOR array; the number of elements in the array **SHALL** be 6 (if CRC type is zero) or 7 (otherwise).

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

- . Block type code, an unsigned integer. Bundle block type code 1 indicates that the block is a bundle payload block. Block type codes 2 through 9 are explicitly reserved as noted later in this specification. Block type codes 192 through 255 are not reserved and are available for private and/or experimental use. All other block type code values are reserved for future use.
- . Block number, an unsigned integer. The block number uniquely identifies the block within the bundle, enabling blocks (notably bundle security protocol blocks) to explicitly reference other blocks in the same bundle. Block numbers need not be in continuous sequence, and blocks need not appear in block number sequence in the bundle. The block number of the payload block is always zero.
- . Block processing control flags as discussed in Section 4.1.4 above.
- . CRC type as discussed in Section 4.1.1 above.
- . If and only if the value of the CRC type field of this block is non-zero, a CRC. If present, the length and nature of the CRC SHALL be as indicated by the CRC type and the CRC SHALL be computed over the concatenation of all bytes of the block (including CBOR "break" characters) including the CRC field itself, which for this purpose SHALL be temporarily populated with the value zero.
- . Block data length, an unsigned integer. The block data length field SHALL contain the aggregate length of all remaining fields of the block, i.e., the block-type-specific data fields. Block data length SHALL be represented as a CBOR unsigned integer item.
- . Block-type-specific data fields, whose nature and order are type-specific and whose aggregate length in octets is the value of the block data length field. For the Payload Block in particular (block type 1), there SHALL be exactly one block-type-specific data field, termed the "payload", which SHALL be an application data unit, or some contiguous extent thereof, represented as a CBOR byte string.

4.3. Extension Blocks

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

The values of the block processing control flags indicate the action to be taken by the bundle protocol agent when this is the case.

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

The following extension blocks are defined in other DTN protocol specification documents as noted:

- . Block Integrity Block (block type 2) and Block Confidentiality Block (block type 3) are defined in the Bundle Security Protocol specification (work in progress).
- . Manifest Block (block type 4) is defined in the Manifest Extension Block specification (work in progress). The manifest block identifies the blocks that were present in the bundle at the time it was created. The bundle MUST contain one (1) occurrence of this type of block if the value of the "manifest" flag in the bundle processing control flags is 1; otherwise the bundle MUST NOT contain any Manifest block.
- . The Flow Label Block (block type 6) is defined in the Flow Label Extension Block specification (work in progress). The flow label block is intended to govern transmission of the bundle by convergence-layer adapters.

The following extension blocks are defined in the current document.

4.3.1. Previous Node

The Previous Node block, block type 7, identifies the node that forwarded this bundle to the local node (i.e., to the node at which the bundle currently resides); its block-type-specific data is the node ID of that forwarder node which SHALL take the form of a node ID represented as described in Section 4.1.5.2. above. If the local node is the source of the bundle, then the bundle MUST NOT contain any previous node block. Otherwise the bundle SHOULD contain one (1) occurrence of this type of block.

4.3.2. Bundle Age

The Bundle Age block, block type 8, contains the number of seconds that have elapsed between the time the bundle was created and time at which it was most recently forwarded. It is intended for use by nodes lacking access to an accurate clock, to aid in determining the time at which a bundle's lifetime expires. The block-type-specific data of this block is an unsigned integer containing the age of the bundle in seconds, which SHALL be represented as a CBOR unsigned

integer item. (The age of the bundle is the sum of all known intervals of the bundle's residence at forwarding nodes, up to the time at which the bundle was most recently forwarded, plus the summation of signal propagation time over all episodes of transmission between forwarding nodes. Determination of these values is an implementation matter.) If the bundle's creation time is zero, then the bundle MUST contain exactly one (1) occurrence of this type of block; otherwise, the bundle MAY contain at most one (1) occurrence of this type of block. A bundle MUST NOT contain multiple occurrences of the bundle age block, as this could result in processing anomalies.

4.3.3. Hop Count

The Hop Count block, block type 9, contains two unsigned integers, hop limit and hop count. A "hop" is here defined as an occasion on which a bundle was forwarded from one node to another node. The hop limit value SHOULD NOT be changed at any time after creation of the Hop Count block; the hop count value SHOULD initially be zero and SHOULD be increased by 1 on each hop.

The hop count block is mainly intended as a safety mechanism, a means of identifying bundles for removal from the network that can never be delivered due to a persistent forwarding error. When a bundle's hop count exceeds its hop limit, the bundle SHOULD be deleted for the reason "hop limit exceeded", following the bundle deletion procedure defined in Section 5.14. Procedures for determining the appropriate hop limit for a block are beyond the scope of this specification. The block-type-specific data in a hop count block SHALL be represented as a CBOR array comprising a 2-tuple. The first item of this array SHALL be the bundle's hop limit, represented as a CBOR unsigned integer. The second item of this array SHALL be the bundle's hop count, represented as a CBOR unsigned integer. A bundle MAY contain at most one (1) occurrence of this type of block.

5. Bundle Processing

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

5.1. Generation of Administrative Records

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

Under some circumstances, the requesting of status reports could result in an unacceptable increase in the bundle traffic in the network. For this reason, the generation of status reports **MUST** be disabled by default and enabled only when the risk of excessive network traffic is deemed acceptable.

When the generation of status reports is enabled, the decision on whether or not to generate a requested status report is left to the discretion of the bundle protocol agent. Mechanisms that could assist in making such decisions, such as pre-placed agreements authorizing the generation of status reports under specified circumstances, are beyond the scope of this specification.

Notes on administrative record terminology:

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

5.2. Bundle Transmission

The steps in processing a bundle transmission request are:

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

EID of a singleton endpoint whose only member is the node of which the BPA is a component.

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

5.3. Bundle Dispatching

The steps in dispatching a bundle are:

Step 1: If the bundle's destination endpoint is an endpoint of which the node is a member, the bundle delivery procedure defined in Section 5.7 MUST be followed.

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

5.4. Bundle Forwarding

The steps in forwarding a bundle are:

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

Step 2: The bundle protocol agent MUST determine whether or not forwarding is contraindicated for any of the reasons listed in Figure 4. In particular:

- . The bundle protocol agent MAY choose either to forward the bundle directly to its destination node(s) (if possible) or to forward the bundle to some other node(s) for further forwarding. The manner in which this decision is made may depend on the scheme name in the destination endpoint ID and/or on other state but in any case is beyond the scope of this document. If the BPA elects to forward the bundle to some other node(s) for further forwarding but finds it impossible to select any node(s) to forward the bundle to, then forwarding is contraindicated.
- . Provided the bundle protocol agent succeeded in selecting the node(s) to forward the bundle to, the bundle protocol agent MUST select the convergence layer adapter(s) whose services will enable the node to send the bundle to those nodes. The manner in which specific appropriate convergence layer adapters are selected is beyond the scope of this document. If the agent finds it impossible to select any appropriate convergence layer adapter(s) to use in forwarding this bundle, then forwarding is contraindicated.

Step 3: If forwarding of the bundle is determined to be contraindicated for any of the reasons listed in Figure 4, then the Forwarding Contraindicated procedure defined in Section 5.4.1 MUST be followed; the remaining steps of Section 5 are skipped at this time.

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

Note that:

- . If the bundle contains a flow label extension block (to be defined in a future document) then that flow label value MAY identify procedures for determining the order in which convergence layer adapters must send bundles, e.g., considering bundle source when determining the order in which bundles are sent. The definition of such procedures is beyond the scope of this specification.
- . If the bundle has a bundle age block, as defined in 4.3.2. above, then at the last possible moment before the CLA initiates conveyance of the bundle node via the CL protocol the bundle age value MUST be increased by the difference between the current time and the time at which the bundle was received (or, if the local node is the source of the bundle, created).

Step 5: When all selected convergence layer adapters have informed the bundle protocol agent that they have concluded their data sending procedures with regard to this bundle:

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

5.4.1. Forwarding Contraindicated

The steps in responding to contraindication of forwarding are:

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

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

Otherwise, when -- at some future time - the forwarding of this bundle ceases to be contraindicated, processing proceeds from Step 5 of Section 5.4.

5.4.2. Forwarding Failed

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

Step 1: The bundle protocol agent MAY forward the bundle back to the node that sent it, as identified by the Previous Node block, if present.

Step 2: If the bundle's destination endpoint is an endpoint of which the node is a member, then the bundle's "Forward pending" retention constraint MUST be removed. Otherwise, the bundle MUST be deleted: the bundle deletion procedure defined in Section 5.14 MUST be followed, citing the reason for which forwarding was determined to be contraindicated.

5.5. Bundle Expiration

A bundle expires when the bundle's age exceeds its lifetime as specified in the primary bundle block. Bundle age MAY be determined by subtracting the bundle's creation timestamp time from the current time if (a) that timestamp time is not zero and (b) the local node's clock is known to be accurate (as discussed in section 4.5.1 above); otherwise bundle age MUST be obtained from the Bundle Age extension block. Bundle expiration MAY occur at any point in the processing of a bundle. When a bundle expires, the bundle protocol agent MUST delete the bundle for the reason "lifetime expired": the bundle deletion procedure defined in Section 5.14 MUST be followed.

5.6. Bundle Reception

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

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

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

Step 3: For each block in the bundle that is an extension block that the bundle protocol agent cannot process:

- . If the block processing flags in that block indicate that a status report is requested in this event, and status reporting is enabled, then a bundle reception status report with reason code "Block unintelligible" SHOULD be generated, destined for the bundle's report-to endpoint ID.
- . If the block processing flags in that block indicate that the bundle must be deleted in this event, then the bundle protocol agent MUST delete the bundle for the reason "Block unintelligible"; the bundle deletion procedure defined in Section 5.14 MUST be followed and all remaining steps of the bundle reception procedure MUST be skipped.
- . If the block processing flags in that block do NOT indicate that the bundle must be deleted in this event but do indicate that the block must be discarded, then the bundle protocol agent MUST remove this block from the bundle.
- . If the block processing flags in that block indicate neither that the bundle must be deleted nor that that the block must be discarded, then processing continues with the next extension block that the bundle protocol agent cannot process, if any; otherwise, processing proceeds from step 4.

Step 4: Processing proceeds from Step 1 of Section 5.3.

5.7. Local Bundle Delivery

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

Step 1: If the received bundle is a fragment, the application data unit reassembly procedure described in Section 5.9 MUST be followed.

If this procedure results in reassembly of the entire original application data unit, processing of this bundle (whose fragmentary payload has been replaced by the reassembled application data unit) proceeds from Step 2; otherwise, the retention constraint "Reassembly pending" MUST be added to the bundle and all remaining steps of this procedure MUST be skipped.

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

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

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

5.8. Bundle Fragmentation

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

The size of a bundle can be reduced by "fragmenting" the bundle. To fragment a bundle whose payload is of size M is to replace it with two "fragments" -- new bundles with the same source node ID and creation timestamp as the original bundle -- whose payloads are the first N and the last $(M - N)$ bytes of the original bundle's payload, where $0 < N < M$. Note that fragments may themselves be fragmented, so fragmentation may in effect replace the original bundle with more than two fragments. (However, there is only one 'level' of fragmentation, as in IP fragmentation.)

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

Fragmentation SHALL be constrained as follows:

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

5.9. Application Data Unit Reassembly

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

- . This byte array -- the reassembled application data unit -- MUST replace the payload of this fragment.
- . The "Reassembly pending" retention constraint MUST be removed from every other fragment whose payload is a subset of the reassembled application data unit.

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

5.10. Bundle Deletion

The steps in deleting a bundle are:

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

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

5.11. Discarding a Bundle

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

5.12. Canceling a Transmission

When requested to cancel a specified transmission, where the bundle created upon initiation of the indicated transmission has not yet been discarded, the bundle protocol agent MUST delete that bundle for the reason "transmission cancelled". For this purpose, the procedure defined in Section 5.14 MUST be followed.

6. Administrative Record Processing

6.1. Administrative Records

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

Every administrative record consists of:

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

Valid administrative record type codes are defined as follows:

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

Figure 3: Administrative Record Type Codes

Each BP administrative record SHALL be represented as a CBOR array comprising a 2-tuple.

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

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

6.1.1.1. Bundle Status Reports

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

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

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

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

Each item of the bundle status information array SHALL be a bundle status item represented as a CBOR array; the number of elements in each such array SHALL be either 2 (if the value of the first item of this bundle status item is 1 AND the "Report status time" flag was set to 1 in the bundle processing flags of the bundle whose status is being reported) or 1 (otherwise). The first item of the bundle status item array SHALL be a status indicator, a Boolean value indicating whether or not the corresponding bundle status is asserted, represented as a CBOR Boolean value. The second item of the bundle status item array, if present, SHALL indicate the time (as reported by the local system clock, an implementation matter) at which the indicated status was asserted for this bundle, represented as a DTN time as described in Section 4.1.6. above.

The second item of the bundle status report array SHALL be the bundle status report reason code explaining the value of the status indicator, represented as a CBOR unsigned integer. Valid status report reason codes are defined in Figure 4 below but the list of status report reason codes provided here is neither exhaustive nor exclusive; supplementary DTN protocol specifications (including, but not restricted to, the Bundle Security Protocol [BPSEC]) may define additional reason codes.

+-----+-----+-----+-----+-----+-----+		
Value		Meaning
+=====+=====+=====+=====+=====+=====+		
0		No additional information.
+-----+-----+-----+-----+-----+-----+		
1		Lifetime expired.
+-----+-----+-----+-----+-----+-----+		
2		Forwarded over unidirectional link.
+-----+-----+-----+-----+-----+-----+		
3		Transmission canceled.
+-----+-----+-----+-----+-----+-----+		
4		Depleted storage.
+-----+-----+-----+-----+-----+-----+		
5		Destination endpoint ID unintelligible.
+-----+-----+-----+-----+-----+-----+		
6		No known route to destination from here.
+-----+-----+-----+-----+-----+-----+		
7		No timely contact with next node on route.
+-----+-----+-----+-----+-----+-----+		
8		Block unintelligible.
+-----+-----+-----+-----+-----+-----+		
9		Hop limit exceeded.
+-----+-----+-----+-----+-----+-----+		
(other)		Reserved for future use.

-----+

Figure 4: Status Report Reason Codes

The third item of the bundle status report array SHALL be the source node ID identifying the source of the bundle whose status is being reported, represented as described in Section 4.1.5.2. above.

The fourth item of the bundle status report array SHALL be the creation timestamp of the bundle whose status is being reported, represented as described in Section 4.1.7. above.

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

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

6.2. Generation of Administrative Records

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

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

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

7. Services Required of the Convergence Layer

7.1. The Convergence Layer

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

defined minimal set of services to the bundle protocol agent. This convergence layer service specification enumerates those services.

7.2. Summary of Convergence Layer Services

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

- . sending a bundle to a bundle node that is reachable via the convergence layer protocol;
- . delivering to the bundle protocol agent a bundle that was sent by a bundle node via the convergence layer protocol.

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

8. Implementation Status

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

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

According to RFC 7942, "this will allow reviewers and working groups to assign due consideration to documents that have the benefit of running code, which may serve as evidence of valuable experimentation and feedback that have made the implemented

protocols more mature. It is up to the individual working groups to use this information as they see fit".

At the time of this writing, the only known implementation of the current document is microPCN (<https://upcn.eu/>). According to the developers:

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

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

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

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

9. Security Considerations

The bundle protocol security architecture and the available security services are specified in an accompanying document, the Bundle Security Protocol specification [BPSEC].

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

Because the security mechanisms are extension blocks that are themselves inserted into the bundle, the integrity and confidentiality of bundle blocks are protected while the bundle is at rest, awaiting transmission at the next forwarding opportunity, as well as in transit.

Additionally, convergence-layer protocols that ensure authenticity of communication between adjacent nodes in BP network topology SHOULD be used where available, to minimize the ability of unauthenticated nodes to introduce inauthentic traffic into the network.

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

Note that the generation of bundle status reports is disabled by default because malicious initiation of bundle status reporting could result in the transmission of extremely large numbers of bundle, effecting a denial of service attack.

The bpsec extensions accommodate an open-ended range of ciphersuites; different ciphersuites may be utilized to protect different blocks. One possible variation is to sign and/or encrypt blocks in symmetric keys securely formed by Diffie-Hellman procedures (such as ECDH) using the public and private keys of the sending and receiving nodes. For this purpose, the key distribution problem reduces to the problem of trustworthy delay-tolerant distribution of public keys, a current research topic.

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

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

Inclusion of the Bundle Security Protocol in any Bundle Protocol implementation is RECOMMENDED. Use of the Bundle Security Protocol in Bundle Protocol operations is OPTIONAL, subject to the following guidelines:

- . Every block (that is not a bpsec extension block) of every bundle SHOULD be authenticated by a BIB citing the ID of the node that inserted that block. (Note that a single BIB may authenticate multiple "target" blocks.) BIB authentication MAY be omitted on (and only on) any initial end-to-end path segments on which it would impose unacceptable overhead, provided that satisfactory authentication is ensured at the

convergence layer and that BIB authentication is asserted on the first path segment on which the resulting overhead is acceptable and on all subsequent path segments.

- . If any segment of the end-to-end path of a bundle will traverse the Internet or any other potentially insecure communication environment, then the payload block SHOULD be encrypted by a BCB on this path segment and all subsequent segments of the end-to-end path.

10. IANA Considerations

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

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

11. References

11.1. Normative References

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

[RFC7049] Borman, C. and P. Hoffman, "Concise Binary Object Representation (CBOR)", RFC 7049, October 2013.

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

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

11.2. Informative References

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

[BPSEC] Birrane, E., "Bundle Security Protocol Specification", Work In Progress, October 2015.

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

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

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

12. Acknowledgments

This work is freely adapted from RFC 5050, which was an effort of the Delay Tolerant Networking Research Group. The following DTNRG participants contributed significant technical material and/or inputs to that document: Dr. Vinton Cerf of Google, Scott Burleigh, Adrian Hooke, and Leigh Torgerson of the Jet Propulsion Laboratory, Michael Demmer of the University of California at Berkeley, Robert Durst, Keith Scott, and Susan Symington of The MITRE Corporation, Kevin Fall of Carnegie Mellon University, Stephen Farrell of Trinity College Dublin, Peter Lovell of SPARTA, Inc., Manikantan Ramadas of Ohio University, and Howard Weiss of SPARTA, Inc.

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

13. Significant Changes from RFC 5050

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

- . Clarify the difference between transmission and forwarding.
- . Migrate custody transfer to the bundle-in-bundle encapsulation specification.
- . Introduce the concept of "node ID" as functionally distinct from endpoint ID, while having the same syntax.
- . Restructure primary block, making it immutable. Add optional CRC.
- . Add optional CRCs to non-primary blocks.
- . Add block ID number to canonical block format (to support streamlined BSP).
- . Add bundle age extension block, defined in this specification.
- . Add previous node extension block, defined in this specification.
- . Add flow label extension block, *not* defined in this specification.
- . Add manifest extension block, *not* defined in this specification.
- . Add hop count extension block, defined in this specification.
- . Migrate Quality of Service markings to a new QoS extension block, *not* defined in this specification.

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) 2017 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). That CDDL expression is 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.

```
start = bundle

dtn-time = uint

creation-timestamp = [dtn-time, sequence: uint]

eid-generic = [uri-code, SSP: any]

uri-code = uint

eid = eid-choice .within eid-generic

eid-choice /= [dtn-code, SSP: (text / 0)]

dtn-code = 1 ; TBD

eid-choice /= [ipn-code, SSP: [nodenum: uint, servicenum: uint]]

ipn-code = 2 ; TBD

bundle-control-flags = uint .bits bundleflagbits

bundleflagbits = &(
    reserved: 15
    reserved: 14
    reserved: 13
    bundle-deletion-status-reports-are-requested: 12
    bundle-delivery-status-reports-are-requested: 11
    bundle-forwarding-status-reports-are-requested: 10
    reserved: 9
```

```
    bundle-reception-status-reports-are-requested: 8
    bundle-contains-a-Manifest-block: 7
    status-time-is-requested-in-all-status-reports: 6
    user-application-acknowledgement-is-requested: 5
    destination-is-a-singleton-endpoint: 4
    reserved: 3
    bundle-must-not-be-fragmented: 2
    payload-is-an-administrative-record: 1
    bundle-is-a-fragment: 0
)

crc = bytes

block-control-flags = uint .bits blockflagbits

blockflagbits = &(amp;
    reserved: 7
    reserved: 6
    reserved: 5
    reserved: 4
    bundle-must-be-deleted-if-block-cannot-be-processed: 3
    status-report-must-be-transmitted-if-block-cannot-be-processed: 2
    block-must-be-removed-from-bundle-if-it-cannot-be-processed: 1
    block-must-be-replicated-in-every-fragment: 0
)

bundle = [primary-block, *extension-block, payload-block]

primary-block = [
```

```
    version: 7,  
    bundle-control-flags,  
    crc-type: uint,  
    destination: eid,  
    source-node: eid,  
    report-to: eid,  
    creation-timestamp,  
    lifetime: uint,  
    ? fragment-offset: uint,  
    ? total-application-data-length: uint,  
    ? crc,  
]  
  
canonical-block-generic = [  
    block-type-code: uint,  
    canonical-block-common,  
    content: any  
]  
  
canonical-block-common = (  
    block-number: uint,  
    block-control-flags,  
    crc-type: uint,  
    block-data-length: uint,  
    ? crc,  
)
```

```
canonical-block = canonical-block-choice .within canonical-block-  
generic  
  
canonical-block-choice /= payload-block  
  
payload-block = [1, canonical-block-common, adu-extent: payload]  
  
payload = bytes / bytes .cbor admin-record  
  
canonical-block-choice /= extension-block  
  
extension-block = extension-block-choice .within canonical-block  
  
extension-block-choice /= previous-node-block  
  
previous-node-block = [7, canonical-block-common, eid]  
  
extension-block-choice /= bundle-age-block  
  
bundle-age-block = [8, canonical-block-common, bundle-age: uint]  
  
extension-block-choice /= hop-count-block  
  
hop-count-block = [9, canonical-block-common,  
    [hop-limit: uint,  
    hop-count: uint]]  
  
admin-record-generic = [record-type: uint, any]  
  
admin-record = admin-record-choice .within admin-record-generic  
  
admin-record-choice /= bundle-status-report  
  
bundle-status-report = [1, [bundle-status-information,  
    bundle-status-reason: uint,  
    admin-common]]  
  
admin-common = (  
    source-node: eid,
```

```
        creation-timestamp,
        ? fragment-offset: uint,
        ? payload-length: uint)

bundle-status-information = [
    reporting-node-received-bundle: bundle-status-item,
    reporting-node-forwarded-the-bundle: bundle-status-item,
    reporting-node-delivered-the-bundle: bundle-status-item,
    reporting-node-deleted-the-bundle: bundle-status-item,
]

bundle-status-item = [
    asserted: bool,
    ? time-of-assertion: dtn-time]
```

Authors' Addresses

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

Kevin Fall
Nefeli Networks, Inc.
2150 Shattuck Ave.
Berkeley, CA 94704
US
Email: kfall@kfall.com

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

Delay-Tolerant Networking
Internet-Draft
Intended status: Standards Track
Expires: January 2, 2018

E. Birrane
K. McKeever
JHU/APL
July 1, 2017

Bundle Protocol Security Specification
draft-ietf-dtn-bpsec-05

Abstract

This document defines a security protocol providing end to end 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 <http://datatracker.ietf.org/drafts/current/>.

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

This Internet-Draft will expire on January 2, 2018.

Copyright Notice

Copyright (c) 2017 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction	3
1.1. Supported Security Services	3
1.2. Specification Scope	4
1.3. Related Documents	5
1.4. Terminology	5
2. Design Decisions	6
2.1. Block-Level Granularity	6
2.2. Multiple Security Sources	7
2.3. Mixed Security Policy	7
2.4. User-Selected Cipher Suites	8
2.5. Deterministic Processing	8
3. Security Blocks	8
3.1. Block Definitions	8
3.2. Uniqueness	9
3.3. Target Multiplicity	9
3.4. Target Identification	10
3.5. Block Representation	10
3.6. Abstract Security Block	11
3.7. Block Integrity Block	14
3.8. Block Confidentiality Block	15
3.9. Block Interactions	16
3.10. Cipher Suite Parameter and Result Identification	17
3.11. BSP Block Example	18
4. Canonical Forms	19
5. Security Processing	20
5.1. Bundles Received from Other Nodes	20
5.1.1. Receiving BCB Blocks	20
5.1.2. Receiving BIB Blocks	21
5.2. Bundle Fragmentation and Reassembly	22
6. Key Management	22
7. Security Policy Considerations	23
8. Security Considerations	24
8.1. Attacker Capabilities and Objectives	24
8.2. Attacker Behaviors and BPSec Mitigations	25
8.2.1. Eavesdropping Attacks	25
8.2.2. Modification Attacks	26
8.2.3. Topology Attacks	27
8.2.4. Message Injection	27
9. Cipher Suite Authorship Considerations	28
10. Defining Other Security Blocks	29
11. IANA Considerations	30
11.1. Bundle Block Types	30
12. References	30
12.1. Normative References	30
12.2. Informative References	31
Appendix A. Acknowledgements	31

Authors' Addresses	31
------------------------------	----

1. Introduction

This document defines security features for the Bundle Protocol (BP) [BPBIS] and is intended for use in Delay Tolerant Networks (DTNs) to provide end-to-end security services.

The Bundle Protocol specification [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.

The BP might be deployed such that portions of 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 end-to-end integrity and confidentiality services for BP bundles.

Integrity services ensure that protected data within a bundle are not changed from the time they are provided to the network to the time they are delivered at their destination. Data changes may be caused by processing errors, environmental conditions, or intentional manipulation.

Confidentiality services ensure that protected data is unintelligible to nodes in the DTN, except for authorized nodes possessing special information. Confidentiality, in this context, applies to the contents of protected data and does not extend to hiding the fact that protected data exist in the bundle.

NOTE: Hop-by-hop authentication is NOT a supported security service in this specification, for three 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. Networks in which BPsec may be deployed may have a mixture of security-aware and not-security-aware nodes. Hop-by-hop authentication cannot be deployed in a network if adjacent nodes in the network have different security capabilities.
3. Hop-by-hop authentication is a special case of data integrity and can be achieved with the integrity mechanisms defined in this specification. Therefore, a separate authentication service is not necessary.

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 applies only to those nodes that implement it, known as "security-aware" nodes. There might be other nodes in the DTN that do not implement BPsec. While all nodes in a BP overlay can exchange bundles, BPsec security operations can only happen at BPsec security-aware nodes.

This specification does not address individual cipher suite implementations. Different networking conditions and operational considerations require varying strengths of security mechanism such that mandating a cipher suite in this specification may result in too much security for some networks and too little security in others. It is expected that separate documents will be standardized to define cipher suites compatible with BPsec, to include operational cipher suites and interoperability cipher suites.

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.

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 [BPBIS] defines the format and processing of bundles, defines the extension block format used to represent BPsec security blocks, and defines the canonicalization algorithms used by this specification.

The Bundle Security Protocol [RFC6257] and Streamlined Bundle Security Protocol [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 [RFC2119].

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

- o Bundle Source - the node which originates a bundle. The Node ID of the BPA originating the bundle.
- o Forwarder - any node that transmits a bundle in the DTN. The Node ID of the Bundle Protocol Agent (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 Destination. 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 Block - a BPSec extension block in a bundle.
- o Security Operation - the application of a security service to a security target, notated as OP(security service, security target). For example, OP(confidentiality, payload). Every security operation in a bundle MUST be unique, meaning that a security service can only be applied to a security target once in a bundle. A security operation is implemented by a security block.
- o Security Service - the security features supported by this specification: integrity and confidentiality.
- o Security Source - a bundle node that adds a security block to a bundle. 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.

2. Design Decisions

The application of security services in a DTN is a complex endeavor that must consider physical properties of the network, policies at each node, and various application security requirements. 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 may 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 MAY have multiple security blocks and these blocks MAY have different security sources.

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

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 destination node might interpret policy as it related to security blocks as a function of the security source for that block.

2.3. Mixed Security Policy

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

Some waypoints may not be security aware and will not be able to process security blocks. Therefore, security blocks MUST have their processing flags set such that the block will be treated appropriately by non-security-aware waypoints

Some waypoints will have security policies that require evaluating security services even if they are not the bundle destination or the final intended destination of the service. For example, a waypoint may choose to verify an integrity service even though the waypoint is not the bundle destination and the integrity service will be needed by other node 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 may 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 may understand security blocks but refuse to process them unless they are the bundle destination.

2.4. User-Selected Cipher Suites

The security services defined in this specification rely on a variety of cipher suites providing integrity signatures, cipher-text, and other information necessary to populate security blocks. Users MAY select different cipher suites to implement security services. For example, some users might prefer a SHA2 hash function for integrity whereas other users may prefer a SHA3 hash function instead. The security services defined in this specification MUST provide a mechanism for identifying what cipher suite has been used to populate 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 security target(s). The integrity information in the BIB MAY be verified by any node in between the BIB security source and the bundle destination. Security-aware waypoints may add or remove BIBs from bundles in accordance with their security policy.

The BCB indicates that the security target(s) have been encrypted at the BCB security source in order to protect its content while in transit. The BCB may be decrypted by security-aware nodes in

the network, up to and including the bundle destination, as a matter of security policy.

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 as a security block, this limits what security blocks may be added to a bundle: if adding a security block to a bundle would cause some other security block to no longer represent a unique security operation then the new block MUST NOT be added.

If multiple security blocks representing the same security operation were allowed in a bundle at the same time, there would exist ambiguity regarding block processing order and the property of deterministic processing blocks would be lost.

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

- o Signing the payload twice: The two operations `OP(integrity, payload)` and `OP(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(integrity, payload)` and `OP(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(integrity, extension_block_1)` and `OP(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(integrity, payload)` and `OP(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.

3.3. Target Multiplicity

Under special circumstances, 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 may be represented by a single security block if and only if 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 cipher suite parameters and key information 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/removed 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. When the security block is processed all security operations represented by the security block MUST be applied/evaluated at that time.

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 [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 [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 and CRC Field (if present)

- o Block Data Length
- o Block Type Specific Data Fields

Security-specific information for a security block is captured in the "Block Type Specific Data Fields".

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.

Security Targets:

This field identifies the block(s) targetted 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.

Cipher Suite Id:

This field identifies the cipher suite 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.

Cipher Suite Flags:

This field identifies which optional fields are present in the security block. This field SHALL be represented as a CBOR unsigned integer containing a bit field of 5 bits indicating the presence or absence of other security block fields, as follows.

Bit 1 (the most-significant bit, 0x10): reserved.

Bit 2 (0x08): reserved.

Bit 3 (0x04): reserved.

Bit 4 (0x02): Security Source Present Flag.

Bit 5 (the least-significant bit, 0x01): Cipher Suite Parameters Present Flag.

In this field, 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 (Optional Field):

This field identifies the Endpoint that inserted the security block in the bundle. If the security source field is not present then the source **MAY** be inferred from other information, such as the bundle source or the previous hop, as defined by security policy. This field **SHALL** be represented by a CBOR array in accordance with [BPBIS] rules for representing Endpoint Identifiers (EIDs).

Cipher Suite Parameters (Optional Field):

This field captures one or more cipher suite parameters that should be provided to security-aware nodes 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 cipher suite parameter. A single cipher suite 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 cipher suite 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 cipher suite parameters array is illustrated in Figure 1.

+-----+-----+		+-----+-----+		+-----+-----+	
	Parameter 1		Parameter 2		... Parameter N
+-----+-----+		+-----+-----+		+-----+-----+	
	Id		Value		Id Value
+-----+-----+		+-----+-----+		+-----+-----+	

Figure 1: Cipher Suite 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.

Target 1				...	Target N			
Result 1		Result M		...	Result 1		Result K	
Id	Value	Id	Value	...	Id	Value	Id	Value

Figure 2: Security Results

3.7. Block Integrity Block

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

- o The Block Type Code value is as specified in Section 11.1.
- o The Block Type Specific Data Fields follow the structure of the ASB.
- o 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).
- o The Cipher Suite Id MUST be documented as an end-to-end authentication-cipher suite or as an end-to-end error-detection-cipher suite.
- o An EID-reference to the security source MAY be present. If this field is not present, then the security source of the block SHOULD be inferred according to security policy and MAY default to the bundle source. The security source may also be specified as part of key information described in Section 3.10.

Notes:

- o It is RECOMMENDED that cipher suite designers 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(integrity, target) is allowed only once in a bundle per target, it is RECOMMENDED that users wishing to support multiple integrity signatures for the same target define a multi-signature cipher suite.
- o For some cipher suites, (e.g., those using asymmetric keying to produce signatures or those using symmetric keying with a group key), the security information MAY be checked at any hop on the

way to the destination that has access to the required keying information, in accordance with Section 3.9.

- o The use of a generally available key is RECOMMENDED if custodial transfer is employed and all nodes SHOULD verify the bundle before accepting custody.

3.8. 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, except that this block MUST have the "replicate in every fragment" flag set if the target of the BCB 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.

The Block Type Specific Data Fields follow the structure of the ASB.

A security target listed in the Security Targets field MAY reference the payload block, a non-security extension block, or a BIB block. A BCB MUST NOT include another BCB as a security target. A BCB MUST NOT target the primary block.

The Cipher Suite Id MUST be documented as a confidentiality cipher suite.

Any additional bytes generated from applying the cipher suite to a security target (such as additional authenticated text) MAY be placed in an appropriate security result (e.g., an Integrity Check Value) in accordance with cipher suite and security policy.

An EID-reference to the security source MAY be present. If this field is not present, then the security source of the block SHOULD be inferred according to security policy and MAY default to the bundle source. The security source may also be specified as part of key information described in Section 3.10.

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 Fields contains cipher-text, not plain-text. Other block fields remain unmodified, with the exception of the Block Data Length field,

which may be changed if the BCB is allowed to change the length of the block (see below).

Fragmentation, reassembly, and custody transfer are adversely affected by a change in size of the payload block due to ambiguity about what byte range of the block is actually in any particular fragment. Therefore, when the security target of a BCB is the bundle payload, the BCB MUST NOT alter the size of the payload block body data. This "in-place" encryption allows fragmentation, reassembly, and custody transfer to operate without knowledge of whether or not encryption has occurred.

If a BCB cannot alter the size of the security target (e.g., the security target is the payload block or block length modifications are disallowed by policy) then differences in the size of the cipher-text and plain-text MUST be handled in the following way. If the cipher-text is shorter in length than the plain-text, padding must be used in accordance with the cipher suite policy. If the cipher-text is larger than the plain-text, overflow bytes MUST be placed in overflow parameters in the Security Result field.

Notes:

- o It is RECOMMENDED that cipher suite designers 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 The BCB block processing control flags MAY 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.
- o A BCB MAY include information as part of additional authenticated data to address parts of the target block that are not converted to cipher-text.

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 confidentiality is being applied to a target that already has integrity applied to it, then an undesirable condition occurs where a security aware waypoint would be unable to check the integrity result of a block because the block contents have been encrypted after the

integrity signature was generated. To address this concern, the following processing rules MUST be followed.

- o If confidentiality is to be applied to a target, it MUST also be applied to any integrity operation already defined for that target. This means that if a BCB is added to encrypt a block, another BCB MUST also be added to encrypt a BIB also targeting that block.
- o An integrity operation MUST NOT be applied to a security target if a BCB in the bundle shares the same security target. This prevents ambiguity in the order of evaluation when receiving a BIB and a BCB for a given security target.
- o An integrity value MUST NOT be evaluated if the BIB providing the integrity value is the security target of an existing BCB block in the bundle. In such a case, the BIB data contains cipher-text as it has been encrypted.
- o An integrity value MUST NOT be evaluated if the security target of the BIB is also the security target of a BCB in the bundle. In such a case, the security target data contains cipher-text as it has been encrypted.
- o As mentioned in Section 3.7, a BIB MUST NOT have a BCB as its security target. BCBs may embed integrity results as part of security results.

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.

3.10. Cipher Suite Parameter and Result Identification

Cipher suite parameters and security results each represent multiple distinct pieces of information in a security block. Each piece of information is assigned an identifier and a CBOR encoding. Identifiers MUST be unique for a given cipher suite but do not need to be unique across all cipher suites. Therefore, parameter ids and security result ids are specified in the context of a cipher suite definition.

Individual BPsec cipher suites SHOULD use existing registries of identifiers and CBOR encodings, such as those defined in [COSE],

whenever possible. Cipher suites MAY define their own identifiers and CBOR encodings when necessary.

A cipher suite MAY include multiple instances of the same identifier for a parameter or result in a security block. Parameters and results are represented using CBOR, and any identification of a new parameter or result MUST include how the value will be represented using the CBOR specification. Ids themselves are always represented as a CBOR unsigned integer.

3.11. BSP Block Example

An example of BPSec blocks applied to a bundle is illustrated in Figure 3. In this figure 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.

Block in Bundle	ID
Primary Block	B1
BIB OP(integrity, target=B1)	B2
BCB OP(confidentiality, target=B4)	B3
Extension Block	B4
BIB OP(integrity, target=B6)	B5
Extension Block	B6
BCB OP(confidentiality, targets=B8,B9)	B7
BIB (encrypted by B7) OP(integrity, target=B9)	B8
Payload Block	B9

Figure 3: Sample Use of BPSec Blocks

In this example a bundle has four non-security-related blocks: the primary block (B1), two extension blocks (B4,B6), and a payload block

(B9). The following security applications are applied to this bundle.

- o An integrity signature applied to the canonicalized primary block. This is accomplished by a single BIB (B2).
- o Confidentiality for the first extension block (B4). This is accomplished by a BCB block (B3).
- o Integrity for the second extension block (B6). This is accomplished by a BIB block (B5). NOTE: If the extension block B6 contains a representation of the serialized bundle (such as a hash over all blocks in the bundle at the time of its last transmission) then the BIB block is also providing an authentication service.
- o An integrity signature on the payload (B10). This is accomplished by a BIB block (B8).
- o Confidentiality for the payload block and its integrity signature. This is accomplished by a BCB block, B7, encrypting B8 and B9. In this case, the security source, key parameters, and service are identical, so a single security block MAY be used for this purpose, rather than requiring two BCBs one to encrypt B8 and one to encrypt B9.

4. Canonical Forms

Security services require consistency and determinism in how information is presented to cipher suites at the security source and at a receiving node. 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 generating a comparison signature. Canonicalization algorithms are used to construct a stable, end-to-end bit representation of a target block.

Canonical forms are not transmitted, they are used to generate input to a cipher suite for security processing at a security-aware node.

The canonicalization of the primary block is as specified in [BPBIS].

All non-primary blocks share the same block structure and are canonicalized as specified in [BPBIS] with the following exception.

- o If the service being applied is a confidentiality service, then the Block Type Code, Block Number, Block Processing Control Flags, CRC Type and CRC Field (if present), and Block Data Length fields

MUST NOT be included in the canonicalization. Confidentiality services are used solely to convert the Block Type Specific Data Fields from plain-text to cipher-text.

- o Reserved flags MUST NOT be included in any canonicalization as it is not known if those flags will change in transit.

These canonicalization algorithms assume that Endpoint IDs do not change from the time at which a security source adds a security block to a bundle and the time at which a node processes that security block.

Cipher suites 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, cipher suite algorithms take precedence over this specification.

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 security-aware node. The processing order is as follows.

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

5.1.1. Receiving BCB Blocks

If a received bundle contains a BCB, the receiving node MUST determine whether it has the responsibility of decrypting the BCB security target and removing the BCB prior to delivering data to an application at the node or forwarding 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 MAY decrypt the BCB if directed to do so as a matter of security policy.

If the security policy of a security-aware node specifies that a bundle 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. This MAY involve removing the security target from the bundle. If

the removed security target is the payload block, the bundle MAY be discarded.

If an encrypted payload block cannot be decrypted (i.e., the decryption key cannot be deduced or decryption fails), 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 [BPBIS]) MAY be generated to reflect bundle or block deletion.

When a BCB is decrypted, the recovered plain-text MUST replace the cipher-text in the security target Block Type Specific Data Fields. If the Block Data Length field was modified at the time of encryption it MUST be updated to reflect the decrypted block length.

If a BCB contains multiple security targets, all security targets MUST be processed when the BCB is processed. Errors and other processing steps SHALL be made as if each security target had been represented by an individual BCB with a single security target.

5.1.2. Receiving BIB Blocks

If a received bundle contains a BIB, the receiving node MUST determine whether it has the final responsibility of verifying the BIB security target and removing it prior to delivering data to an application at the node or forwarding the bundle. If a BIB check fails, the security target has failed to authenticate and the security target SHALL be processed according to the security policy. A bundle status report indicating the failure MAY be generated. Otherwise, if the BIB verifies, the security target is ready to be processed for delivery.

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

If the security policy of a security-aware node specifies that a bundle should 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. This MAY involve removing the security target from the bundle. If the removed security target is the payload or primary block, the bundle

MAY be discarded. This action may occur at any node that has the ability to verify an integrity signature, not just the bundle destination.

If a receiving node does not have the final responsibility of verifying the BIB it MAY still attempt to verify the BIB to prevent the needless forwarding of corrupt data. If the check fails, the node SHALL process the security target in accordance to local security policy. It is RECOMMENDED that if a payload integrity check fails at a waypoint that it is processed in the same way as if the check fails at the destination. If the check passes, the node MUST NOT remove the BIB prior to forwarding.

If a BIB contains multiple security targets, all security targets MUST be processed if the BIB is processed by the Node. Errors and other processing steps SHALL be made as if each security target had been represented by an individual BIB with a single security target.

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 [BPBIS] MUST be followed. As defined there and summarized here for completeness, only the payload block may 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.

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 more than one security operation, in violation of BPsec, then the BPA must determine how to handle this bundle. The bundle may be discarded, the block affected by the security operation may be discarded, or one security operation may be favored over another.
- o BPAs in the network MUST understand what security operations they should apply to bundles. This decision may be based on the source of the bundle, the destination of the bundle, or some other information related to the bundle.
- o If 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 only be applied to the blocks that absolutely need them. If a BPA were to apply security operations such as integrity or confidentiality to every block in the bundle, regardless of need, there could be downstream errors processing blocks whose contents must be inspected or changed at every hop along the path.
- 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, an integrity signature may be generated and added to the BCB for the security target as additional information in the security result field.
 2. The encrypted block may be replicated as a new block and integrity signed.
 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.

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.

It should be noted that BPSEC addresses only the security of data traveling over the DTN, not the underlying DTN itself. Additionally, BPsec addresses neither the fitness of externally-defined cryptographic methods nor the security of their implementation. It is the responsibility of the BPsec implementer that appropriate algorithms and methods are chosen. Furthermore, the BPsec protocol 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.

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 man-in-the-middle (MITM) attackers.

8.1. Attacker Capabilities and Objectives

BPsec was designed to protect against MITM threats which may have access to a bundle during transit from its source, Alice, to its destination, Bob. A MITM node, Mallory, 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. For the purposes of this section, any MITM node is assumed to effectively be security-aware even if it does not implement the BPsec protocol.

There are three classes of MITM nodes which are differentiated based on their access to cryptographic material:

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

If Mallory is operating as a privileged node, this is tantamount to compromise; BPsec does not provide mechanisms to detect or remove Mallory 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 $K_M == K_A == K_B$.

A special case of the legitimate node is when Mallory is either Alice or Bob (i.e., $K_M == K_A$ or $K_M == K_B$). In this case, Mallory is able to impersonate traffic as either Alice or Bob, 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 Mallory 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 Mallory 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. If a bundle does persist on the network for years and the cipher suite used for a BCB provides inadequate protection, Mallory may be able to recover the protected data before that bundle reaches its intended destination.

8.2.2. Modification Attacks

As a node participating in the DTN between Alice and Bob, Mallory 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 [BPBIS]. Mallory 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 Mallory.

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 Mallory 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, Mallory 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 Mallory removes blocks from a bundle. If Mallory removes a BCB block, 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 Mallory removes both a BCB (or BIB) and its security target there is no evidence left in the bundle of the security operation. Similarly, if Mallory 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 Mallory 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 Mallory instead of Alice. In order to provide verifiable integrity checks, both a BIB and BCB should be used. 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. In this configuration, since Mallory is only a legitimate node and does not have access to Alice's key K_A , Mallory is unable to decrypt the BCB and replace the BIB.

8.2.3. Topology Attacks

If Mallory is in a MITM 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, Mallory 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

Mallory 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 Mallory's objectives may vary, but may be targeting either the bundle protocol or application-layer protocols conveyed by the bundle protocol.

BPSec relies on cipher suite capabilities to prevent replay or forged message attacks. A BCB used with appropriate cryptographic mechanisms (e.g., a counter-based cipher mode) 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.

9. Cipher Suite Authorship Considerations

Cipher suite 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. Cipher suite developers should consider these conditions to better describe the conditions when those suites will operate or exhibit vulnerability, and selection of these suites for implementation should be made with consideration to the reality. There are key differences that may limit the opportunity 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 new cipher suites for use with BPSec, the following information SHOULD be considered for inclusion in these specifications.

- o Cipher Suite Parameters. Cipher suites MUST define their parameter ids, the data types of those parameters, and their CBOR encoding.
- o Security Results. Cipher suites MUST define their security result ids, the data types of those results, and their CBOR encoding.
- o New Canonicalizations. Cipher suites MAY define new canonicalization algorithms as necessary.

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 MAY co-exist within a bundle and MAY 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 MAY 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 MAY 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 standards defining new security blocks and the identification of such blocks shall not, alone, require maintenance of this specification.

11. IANA Considerations

A registry of cipher suite identifiers will be required.

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
TBD	Block Integrity Block	This document
TBD	Block Confidentiality Block	This document

Table 1

12. References

12.1. Normative References

- [BPBIS] Burleigh, S., Fall, K., and E. Birrane, "Bundle Protocol", draft-ietf-dtn-bpbis-06 (work in progress), July 2016.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.
- [RFC3552] Rescorla, E. and B. Korver, "Guidelines for Writing RFC Text on Security Considerations", BCP 72, RFC 3552, DOI 10.17487/RFC3552, July 2003, <<http://www.rfc-editor.org/info/rfc3552>>.

- [RFC6255] Blanchet, M., "Delay-Tolerant Networking Bundle Protocol IANA Registries", RFC 6255, May 2011.

12.2. Informative References

- [COSE] Schaad, J., "CBOR Object Signing and Encryption (COSE)", draft-ietf-cose-msg-24 (work in progress), November 2016.
- [RFC4838] Cerf, V., Burleigh, S., Hooke, A., Torgerson, L., Durst, R., Scott, K., Fall, K., and H. Weiss, "Delay-Tolerant Networking Architecture", RFC 4838, April 2007.
- [RFC6257] Symington, S., Farrell, S., Weiss, H., and P. Lovell, "Bundle Security Protocol Specification", RFC 6257, May 2011.
- [SBSP] Birrane, E., "Streamlined Bundle Security Protocol", draft-birrane-dtn-sbsp-01 (work in progress), October 2015.

Appendix A. Acknowledgements

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

Authors' Addresses

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

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

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
Obsoletes: 7242 (if approved)
Intended status: Standards Track
Expires: November 23, 2017

B. Sipos
RKF Engineering
M. Demmer
UC Berkeley
J. Ott
Aalto University
S. Perreault
May 22, 2017

Delay-Tolerant Networking TCP Convergence Layer Protocol Version 4
draft-ietf-dtn-tcpclv4-02

Abstract

This document describes a revised protocol for the TCP-based convergence layer (TCPCL) for Delay-Tolerant Networking (DTN). The protocol revision is based on implementation issues in the original TCPCL Version 3 and updates to the Bundle Protocol contents, encodings, and convergence layer requirements in Bundle Protocol Version 7. Several new IANA registries are defined for TCPCLv4 which define some behaviors inherited from TCPCLv3 but with updated encodings and/or semantics.

Status of This Memo

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

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

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

This Internet-Draft will expire on November 23, 2017.

Copyright Notice

Copyright (c) 2017 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents

(<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction	3
2. Requirements Language	4
2.1. Definitions Specific to the TCPCL Protocol	4
3. General Protocol Description	5
3.1. Bidirectional Use of TCPCL Sessions	6
3.2. Example Message Exchange	7
4. Session Establishment	8
4.1. Contact Header	10
4.1.1. Header Extension Items	11
4.2. Validation and Parameter Negotiation	13
5. Established Session Operation	14
5.1. Message Type Codes	14
5.2. Upkeep and Status Messages	15
5.2.1. Session Upkeep (KEEPALIVE)	15
5.2.2. Message Rejection (MSG_REJECT)	16
5.3. Session Security	17
5.3.1. TLS Handshake Result	17
5.3.2. Example TLS Initiation	18
5.4. Bundle Transfer	18
5.4.1. Bundle Transfer ID	19
5.4.2. Transfer initialization (XFER_INIT)	19
5.4.3. Data Transmission (XFER_SEGMENT)	20
5.4.4. Data Acknowledgments (XFER_ACK)	22
5.4.5. Transfer Refusal (XFER_REFUSE)	23
6. Session Termination	24
6.1. Shutdown Message (SHUTDOWN)	25
6.2. Idle Session Shutdown	27
7. Security Considerations	27
8. IANA Considerations	29
8.1. Port Number	29
8.2. Protocol Versions	29
8.3. Header Extension Types	30
8.4. Message Types	31
8.5. XFER_REFUSE Reason Codes	31
8.6. SHUTDOWN Reason Codes	32
8.7. MSG_REJECT Reason Codes	33
9. Acknowledgments	33
10. References	33

10.1. Normative References	33
10.2. Informative References	34
Appendix A. Significant changes from RFC7242	35
Authors' Addresses	35

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 revised Bundle Protocol (BP) [I-D.ietf-dtn-bpbis], an application-layer protocol that is used to construct a store-and-forward overlay network. As described in the Bundle Protocol specification [I-D.ietf-dtn-bpbis], it 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 TCPCL.

The locations of the TCPCL and the BP in the Internet model protocol stack 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.

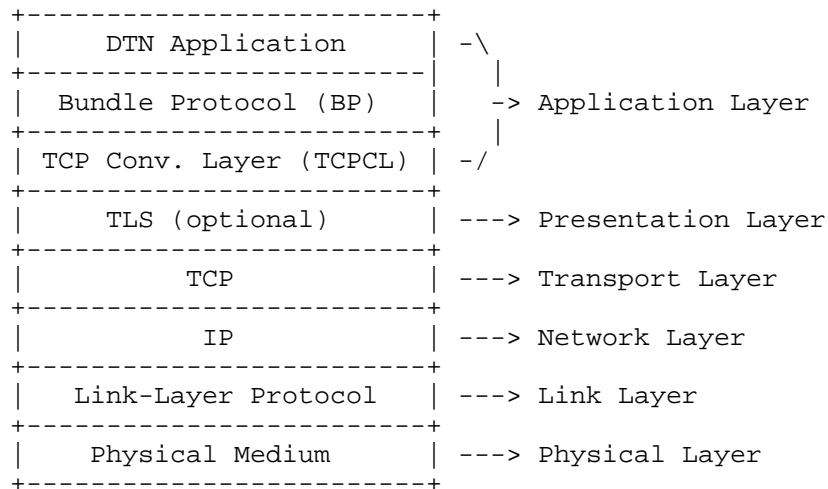


Figure 1: The Locations of the Bundle Protocol and the TCP Convergence-Layer Protocol above the Internet Protocol Stack

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

- o The format of protocol data units of the Bundle Protocol, as those are defined elsewhere in [RFC5050] and [I-D.ietf-dtn-bpbis]. This includes the concept of bundle fragmentation or bundle encapsulation. The TCPCL transfers bundles as opaque data blocks.
- o Mechanisms for locating or identifying other bundle nodes within an internet.

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].

2.1. Definitions Specific to the TCPCL Protocol

This section contains definitions that are interpreted to be specific to the operation of the TCPCL protocol, as described below.

TCP Connection: A TCP connection refers to a transport connection using TCP as the transport protocol.

TCPCL Session: A TCPCL session (as opposed to a TCP connection) is a TCPCL communication relationship between two bundle nodes. The lifetime of a TCPCL session is bound to the lifetime of an underlying TCP connection. Therefore, a TCPCL session is initiated after a bundle node establishes a TCP connection to for the purposes of bundle communication. A TCPCL session is terminated when the TCP connection ends, due either to one or both nodes actively terminating the TCP connection or due to network errors causing a failure of the TCP connection. For the remainder of this document, the term "session" without the prefix "TCPCL" refer to a TCPCL session.

Session parameters: The session parameters 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 node and thereby to the TCPCL is implementation dependent. However, the mechanism by which two bundle nodes exchange and negotiate the values to be used for a given session is described in Section 4.2.

Transfer Transfer refers to the procedures and mechanisms (described below) for conveyance of an individual bundle from one node to another. Each transfer within TCPCLv4 is identified by a Transfer ID number which is unique only to a single direction within a single Session.

3. General Protocol Description

The service of this protocol is the transmission of DTN bundles over TCP. This document specifies the encapsulation of bundles, procedures for TCP setup and teardown, and a set of messages and node requirements. The general operation of the protocol is as follows.

First, one node establishes a TCPCL session to the other by initiating a TCP connection. After setup of the TCP connection is complete, an initial contact header is exchanged in both directions to set parameters of the TCPCL session and exchange a singleton endpoint identifier for each node (not the singleton Endpoint Identifier (EID) of any application running on the node) to denote the bundle-layer identity of each DTN node. This is used to assist in routing and forwarding messages, e.g., to prevent loops.

Once the TCPCL session is established and configured in this way, bundles can be transferred in either direction. Each transfer is performed in one or more logical segments of data. Each logical data segment consists of a XFER_SEGMENT message header and flags, a count of the length of the segment, and finally the octet range of the bundle data. The choice of the length to use for segments is an implementation matter (but must be within the Segment MRU size of

Section 4.1). The first segment for a bundle MUST set the 'START' flag, and the last one MUST set the 'end' flag in the XFER_SEGMENT message flags.

If multiple bundles are transmitted on a single TCPCL connection, they MUST be transmitted consecutively. Interleaving data segments from different bundles is not allowed. Bundle interleaving can be accomplished by fragmentation at the BP layer or by establishing multiple TCPCL sessions.

A feature of this protocol is for the receiving node to send acknowledgments as bundle data segments arrive (XFER_ACK). The rationale behind these acknowledgments is to enable the sender node 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. For each data segment that is received, the receiving node sends an XFER_ACK message containing the cumulative length of the bundle that has been received. The sending node MAY transmit multiple XFER_SEGMENT messages without necessarily waiting for the corresponding XFER_ACK responses. This enables pipelining of messages on a channel. In addition, there is no explicit flow control on the TCPCL layer.

Another feature is that a receiver MAY 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 current bundle, after completing transmission of a partially sent data segment. 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 liveness information.

Finally, before sessions close, a SHUTDOWN message is sent to the session peer. After sending a SHUTDOWN message, the sender of this message MAY send further acknowledgments (XFER_ACK or XFER_REFUSE) but no further data messages (XFER_SEGMENT). A SHUTDOWN message MAY also be used to refuse a session setup by a peer.

3.1. Bidirectional Use of TCPCL Sessions

There are specific messages for sending and receiving operations (in addition to session setup/teardown). TCPCL is symmetric, i.e., 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.

3.2. Example Message Exchange

The following figure visually 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 Node A to Node B.

Note that the sending node MAY transmit multiple XFER_SEGMENT messages without necessarily waiting for the corresponding XFER_ACK responses. This enables pipelining of messages on a channel. 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.

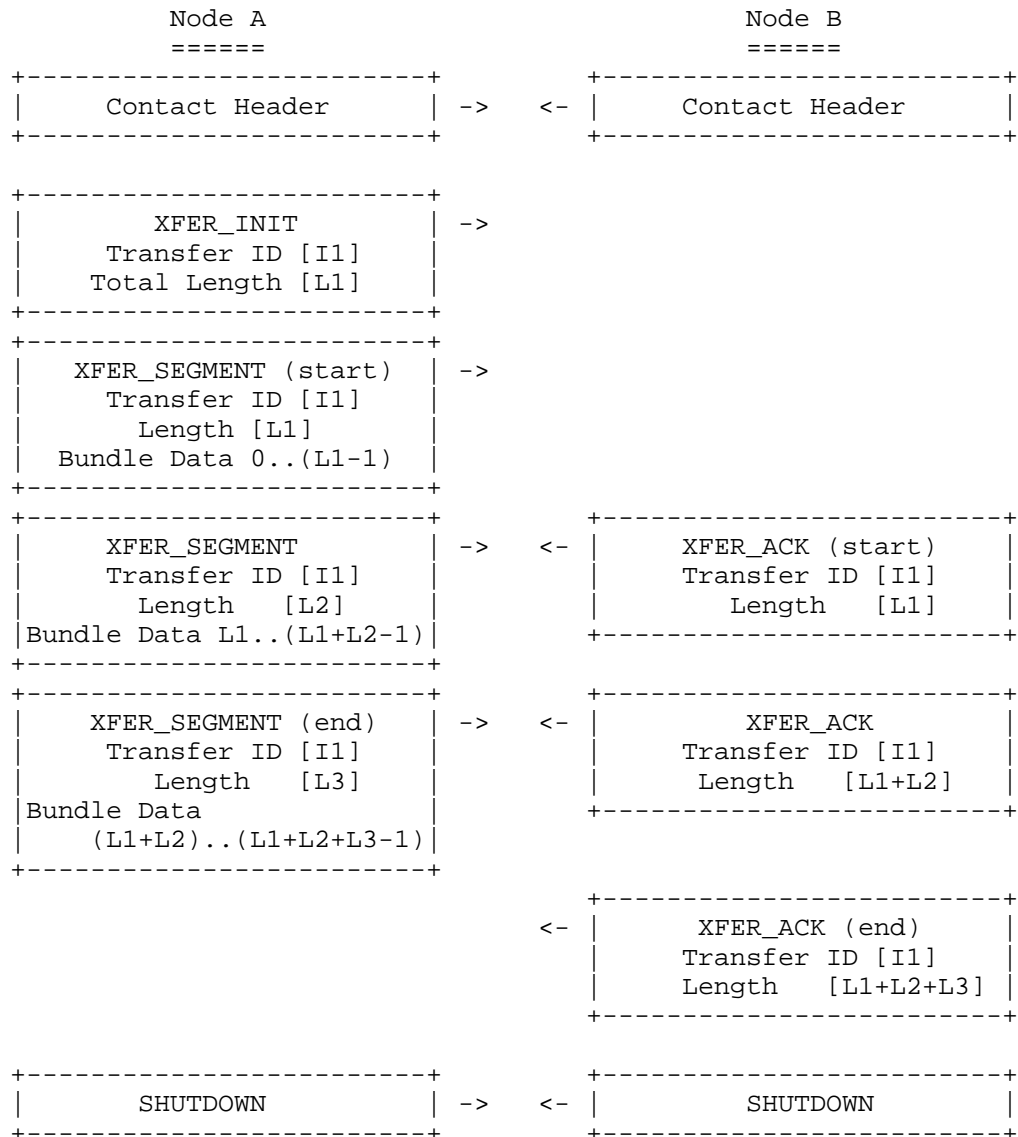


Figure 2: A SL1e Visual Example of the Flow of Protocol Messages on a Single TCP Session between Two Nodes (A and B)

4. Session Establishment

For bundle transmissions to occur using the TCPCL, a TCPCL session MUST first be established between communicating nodes. It is up to the implementation to decide how and when session setup is triggered.

For example, some sessions MAY be opened proactively and maintained for as long as is possible given the network conditions, while other sessions MAY be opened only when there is a bundle that is queued for transmission and the routing algorithm selects a certain next-hop node.

To establish a TCPCL session, a node MUST first establish a TCP connection with the intended peer node, typically by using the services provided by the operating system. Destination port number 4556 has been assigned by IANA as the well-known 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 well-known TCPCL port) 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 node is unable to establish a TCP connection for any reason, then it is an implementation matter to determine how to handle the connection failure. A node 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 node MUST retry the connection setup only after some delay (a 1-second minimum is RECOMMENDED), and it SHOULD use a (binary) exponential backoff mechanism to increase this delay in case of repeated failures. In case a SHUTDOWN message specifying a reconnection delay is received, that delay is used as the initial delay. The default initial delay SHOULD be at least 1 second but SHOULD be configurable since it will be application and network type dependent.

The node MAY declare failure after one or more connection attempts and MAY attempt to find an alternate route for bundle data. Such decisions are up to the higher layer (i.e., the BP).

Once a TCP connection is established, each node MUST immediately transmit a contact header over the TCP connection. The format of the contact header is described in Section 4.1.

Upon receipt of the contact header, both nodes perform the validation and negotiation procedures defined in Section 4.2

After receiving the contact header from the other node, either node MAY also refuse the session by sending a SHUTDOWN message. If session setup is refused, a reason MUST be included in the SHUTDOWN message.

4.1. Contact Header

Once a TCP connection is established, both parties exchange a contact header. This section describes the format of the contact header and the meaning of its fields.

The format for the Contact Header is as follows:

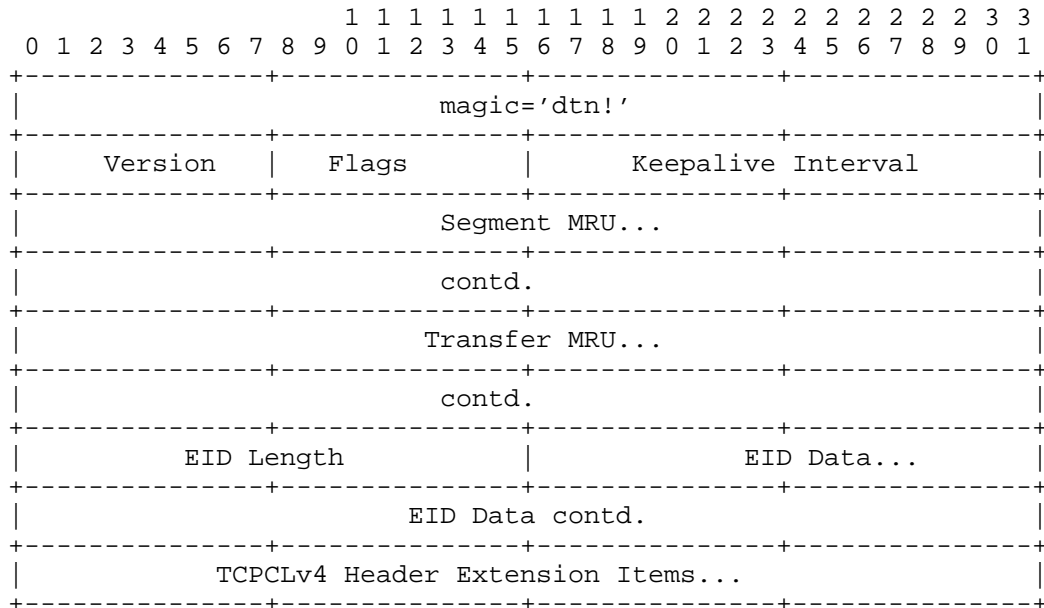


Figure 3: Contact Header Format

See Section 4.2 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 protocol).

Flags: A one-octet field of single-bit flags, interpreted according to the descriptions in Table 1.

Keepalive Interval: A 16-bit unsigned integer indicating the longest allowable interval in seconds between KEEPALIVE messages received in this session.

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 endpoints 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 data payload no longer than the peer's Transfer MRU. This value can be used to perform proactive bundle fragmentation. The two endpoints of a single session MAY have different Transfer MRUs, and no relation between the two is required.

EID Length and EID Data: Together these fields represent a variable-length text string. The EID Length is a 16-bit unsigned integer indicating the number of octets of EID Data to follow. A zero EID Length SHALL be used to indicate the lack of EID rather than a truly empty EID. This case allows an endpoint to avoid exposing EID information on an untrusted network. A non-zero-length EID Data SHALL contain the UTF-8 encoded EID of some singleton endpoint in which the sending node is a member, in the canonical format of <scheme name>:<scheme-specific part>. This EID encoding is consistent with [I-D.ietf-dtn-bpbis].

Header Extension Values: The remaining items of the contact header represent protocol extension data not defined by this specification. The encoding of each Header Extension Item is identical form as described in Section 4.1.1.

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

Table 1: Contact Header Flags

4.1.1. Header Extension Items

Each of the Header Extension items SHALL be encoded in an identical Type-Length-Value (TLV) container form as indicated in Figure 4. The fields of the header extension item are:

Flags: A one-octet field containing generic bit flags about the item, which are listed in Table 2. If a TCPCL endpoint receives an extension item with an unknown Item Type and the CRITICAL flag set, the endpoint SHALL close the TCPCL session with SHUTDOWN reason code of "Contact Failure". If the CRITICAL flag is not set, an endpoint 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. Each type This specification does not define any extension types directly, but does allocate an IANA registry for such codes (see Section 8.3).

Item Length: A 32-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 extensions use of available Item Value data. Extension specification SHOULD avoid the use of large data exchanges within the TCPCLv4 contact header as no bundle transfers can begin until the a full contact exchange and negotiation has been completed.

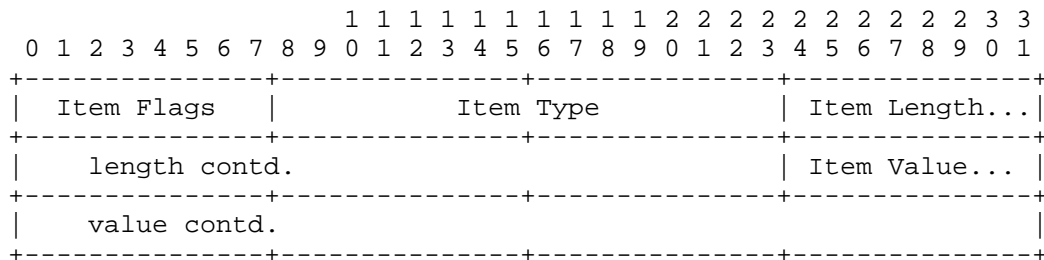


Figure 4: Header Extention Item Format

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

Table 2: Header Extension Item Flags

4.2. Validation and Parameter Negotiation

Upon reception of the contact header, each node 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 node MAY elect to hold an invalid connection open and idle for some time before closing it.

If a node receives a contact header containing a version that is greater than the current version of the protocol that the node implements, then the node SHALL shutdown the session with a reason code of "Version mismatch". If a node receives a contact header with a version that is lower than the version of the protocol that the node implements, the node MAY either terminate the session (with a reason code of "Version mismatch"). Otherwise, the node MAY adapt its operation to conform to the older version of the protocol. This decision is an implementation matter. When establishing the TCPCL session, a node SHOULD send the contact header for the latest version of TCPCL that it can use.

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

Session Keepalive: Negotiation of the Session Keepalive parameter is performed by taking the minimum of this two contact headers' Keepalive Interval. If the negotiated Session Keepalive is zero (i.e. one or both contact headers contains a zero Keepalive Interval), then the keepalive feature (described in Section 5.2.1) is disabled. There is no logical minimum value for the keepalive interval, but when used for many sessions on an open, shared network a short interval could lead to excessive traffic. For shared network use, endpoints SHOULD choose a keepalive interval no shorter than 30 seconds. There is no logical maximum value for the keepalive interval, but an idle TCP connection is liable for closure by the host operating system if the keepalive time is longer than tens-of-minutes. Endpoints SHOULD choose a keepalive interval no longer than 10 minutes (600 seconds).

Enable TLS: Negotiation of the Enable TLS parameter is performed by taking the logical AND of the two contact headers' CAN_TLS flags. If the negotiated Enable TLS value is true then TLS negotiation feature (described in Section 5.3) begins immediately following the contact header exchange.

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

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. Message Type Codes

After the initial exchange of a contact header, all messages transmitted over the session are identified by a one-octet header with the following structure:

```

  0 1 2 3 4 5 6 7
+-----+
| Message Type |
+-----+
```

Figure 5: Format of the Message Header

The message header fields are as follows:

Message Type: Indicates the type of the message as per Table 3 below.

The message types defined in this specification are listed in Table 3. Encoded values are listed in Section 8.4.

Type	Description
XFER_INIT	Contains the length (in octets) of the next transfer, as described in Section 5.4.2.
XFER_SEGMENT	Indicates the transmission of a segment of bundle data, as described in Section 5.4.3.
XFER_ACK	Acknowledges reception of a data segment, as described in Section 5.4.4.
XFER_REFUSE	Indicates that the transmission of the current bundle SHALL be stopped, as described in Section 5.4.5.
KEEPALIVE	Used to keep TCPCL session active, as described in Section 5.2.1.
SHUTDOWN	Indicates that one of the nodes participating in the session wishes to cleanly terminate the session, as described in Section 6.
MSG_REJECT	Contains a TCPCL message rejection, as described in Section 5.2.2.

Table 3: TCPCL Message Types

5.2. Upkeep and Status Messages

5.2.1. 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 4.1, one of the parameters in the contact header is the Keepalive Interval. Both sides populate this field with their requested intervals (in seconds) between KEEPALIVE messages.

The format of a KEEPALIVE message is a one-octet message type code of KEEPALIVE (as described in Table 3) with no additional data. Both sides SHOULD 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 for at least twice the Keepalive Interval, then either party MAY terminate the session by transmitting a one-octet SHUTDOWN message (as described in Section 6.1, with reason code "Idle Timeout") and by closing the session.

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 MAY experience noticeable latency.

5.2.2. Message Rejection (MSG_REJECT)

If a TCPCL endpoint receives a message which is unknown to it (possibly due to an unhandled protocol mismatch) or is inappropriate for the current session state (e.g. a KEEPALIVE message received after contact header negotiation has disabled that feature), there is a protocol-level message to signal this condition in the form of a MSG_REJECT reply.

The format of a MSG_REJECT message follows:

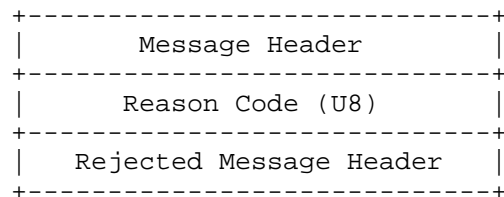


Figure 6: Format of MSG_REJECT Messages

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 endpoint.
Message Unsupported	0x02	A message was received but the TCPCL endpoint 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

5.3. Session Security

This version of the TCPCL supports establishing a session-level Transport Layer Security (TLS) session within an existing TCPCL session. Negotiation of whether or not to initiate TLS within TCPCL session is part of the contact header as described in Section 4.2.

When TLS is used within the TCPCL it affects the entire session. By convention, this protocol uses the endpoint which initiated the underlying TCP connection as the "client" role of the TLS handshake request. Once a TLS session is established within TCPCL, there is no mechanism provided to end the TLS session and downgrade the session. If a non-TLS session is desired after a TLS session is started then the entire TCPCL session MUST be shutdown first.

After negotiating an Enable TLS parameter of true, and before any other TCPCL messages are sent within the session, the session endpoints SHALL begin a TLS handshake in accordance with [RFC5246]. The parameters within each TLS negotiation are implementation dependent but any TCPCL endpoint SHOULD follow all recommended best practices of [RFC7525].

5.3.1. TLS Handshake Result

If a TLS handshake cannot negotiate a TLS session, both endpoints of the TCPCL session SHALL cause a TCPCL shutdown with reason "TLS negotiation failed".

After a TLS session is successfully established, both TCPCL endpoints SHALL re-exchange TCPCL Contact Header messages. Any information

cached from the prior Contact Header exchange SHALL be discarded. This re-exchange avoids man-in-the-middle attack in identical fashion to [RFC2595].

5.3.2. Example TLS Initiation

A summary of a typical CAN_TLS usage is shown in the sequence in Figure 7 below.

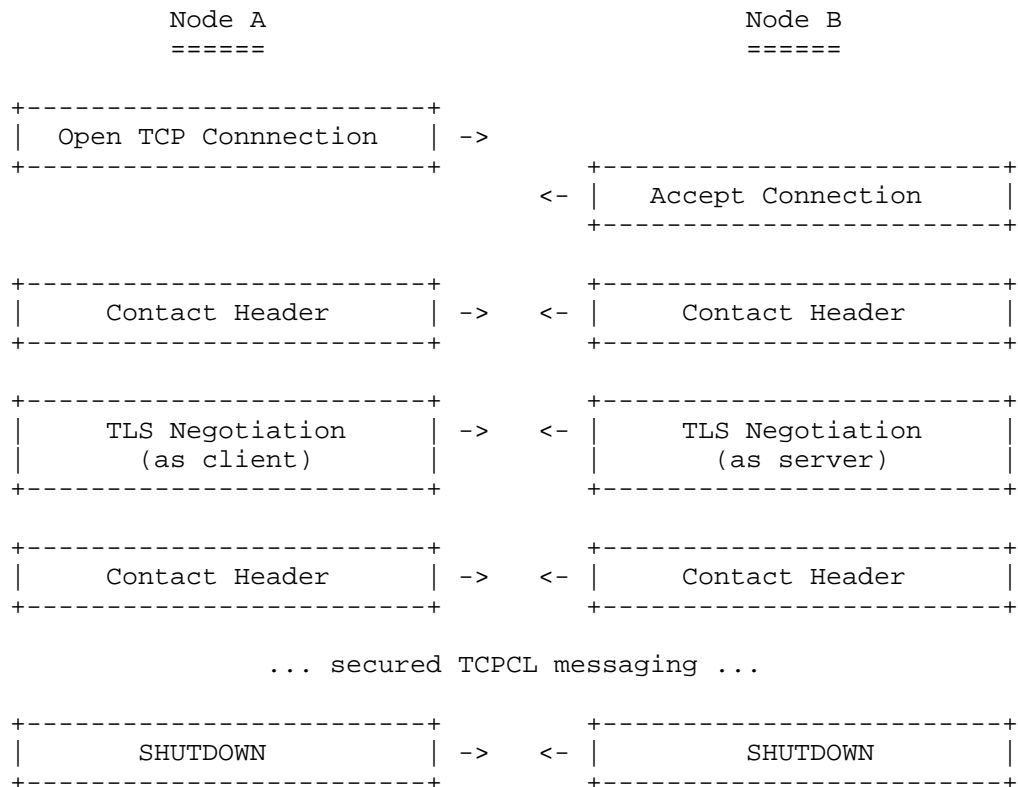


Figure 7: A simple visual example of TCPCL TLS Establishment between two nodes

5.4. Bundle Transfer

All of the message in this section are directly associated with tranfering a bundle between TCPCL endpoints.

A single TCPCL transfer results in a bundle (handled by the convergence layer as opaque data) being exchanged from one endpoint 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 4.1).

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.

5.4.1. Bundle Transfer ID

Each of the bundle transfer messages contains a Transfer ID number which is used to correlate messages originating from sender and receiver of a 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 XFER_INIT message and some number of XFER_SEGMENT and XFER_ACK messages; all are correlated by the same Transfer ID.

Transfer IDs from each endpoint SHALL be unique within a single TCPCL session. The initial Transfer ID from each endpoint SHALL have value zero. Subsequent Transfer ID values SHALL be incremented from the prior Transfer ID value by one. Upon exhaustion of the entire 64-bit Transfer ID space, the sending endpoint SHALL terminate the session with SHUTDOWN reason code "Resource Exhaustion".

For bidirectional bundle transfers, a TCPCL endpoint SHOULD NOT rely on any relation between Transfer IDs originating from each side of the TCPCL session.

5.4.2. Transfer initialization (XFER_INIT)

The XFER_INIT message contains the total length, in octets, of the bundle data in the associated transfer. The total length is formatted as a 64-bit unsigned integer.

The purpose of the XFER_INIT message is to allow nodes to preemptively refuse bundles that would exceed their resources or to prepare storage on the receiving node for the upcoming bundle data. See Section 5.4.5 for details on when refusal based on XFER_INIT content is acceptable.

The Total Bundle Length field within a XFER_INIT message 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 XFER_INIT message, the receiver SHOULD treat the transmitted data as invalid.

The format of the XFER_INIT message is as follows:

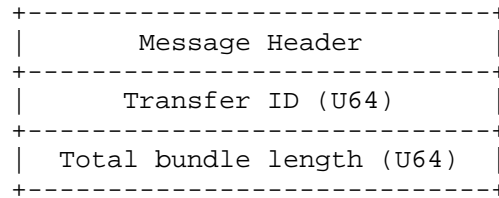


Figure 8: Format of XFER_INIT Messages

The fields of the XFER_INIT message are:

Transfer ID: A 64-bit unsigned integer identifying the transfer about to begin.

Total bundle length: A 64-bit unsigned integer indicating the size of the data-to-be-transferred.

XFER_INIT messages SHALL be sent immediately before transmission of any XFER_SEGMENT messages. XFER_INIT messages MUST NOT be sent unless the next XFER_SEGMENT message has the 'START' bit set to "1" (i.e., just before the start of a new transfer).

A receiver MAY send a BUNDLE_REFUSE message as soon as it receives a XFER_INIT message without waiting for the next XFER_SEGMENT message. The sender MUST be prepared for this and MUST associate the refusal with the correct bundle via the Transfer ID fields.

Upon reception of a XFER_INIT message not immediately before the start of a starting XFER_SEGMENT the receiver SHALL send a MSG_REJECT message with a Reason Code of "Message Unexpected".

5.4.3. Data Transmission (XFER_SEGMENT)

Each bundle is transmitted in one or more data segments. The format of a XFER_SEGMENT message follows in Figure 9.

Message Header
Message Flags (U8)
Transfer ID (U64)
Data length (U64)
Data contents (octet string)

Figure 9: 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.

Transfer ID: A 64-bit unsigned integer identifying the transfer being made.

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 optional values in the two low-order bits, denoted 'START' and 'END' in Table 5. The 'START' bit MUST be set to one if it precedes the transmission of the first segment of a transfer. The 'END' bit MUST be set to one 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' bits MUST be set to one.

Once a transfer of a bundle has commenced, the node **MUST** only send segments containing sequential portions of that bundle until it sends a segment with the 'END' bit set. No interleaving of multiple transfers from the same endpoint is possible within a single TCPCL session. Simultaneous transfers between two endpoints **MAY** be achieved using multiple TCPCL sessions.

5.4.4. 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, a Bundle Protocol agent needs an additional mechanism to determine whether the receiving agent has successfully received the segment. To this end, the TCPCL protocol provides feedback messaging whereby a receiving node transmits acknowledgments of reception of data segments.

The format of an XFER_ACK message follows in Figure 10.

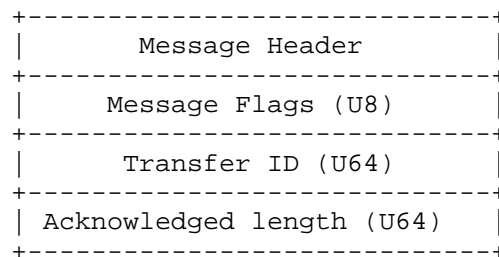


Figure 10: Format of XFER_ACK Messages

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.

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 endpoint SHALL send an XFER_ACK message in response to each received XFER_SEGMENT message. The flags portion of the XFER_ACK header SHALL be set to match the corresponding DATA_SEGMENT message being acknowledged. 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.

For example, suppose the sending node transmits four segments of bundle data with lengths 100, 200, 500, and 1000, respectively. After receiving the first segment, the node sends an acknowledgment of length 100. After the second segment is received, the node sends an acknowledgment of length 300. The third and fourth acknowledgments are of length 800 and 1800, respectively.

5.4.5. Transfer Refusal (XFER_REFUSE)

As bundles can be large, the TCPCL supports an optional mechanism by which a receiving node MAY indicate to the sender that it does not want to receive the corresponding bundle.

To do so, upon receiving a XFER_INIT or XFER_SEGMENT message, the node MAY transmit a XFER_REFUSE message. As data segments and acknowledgments MAY 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 endpoint (which is supposed to represent a firm limitation of what the endpoint 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.

The format of the XFER_REFUSE message is as follows:

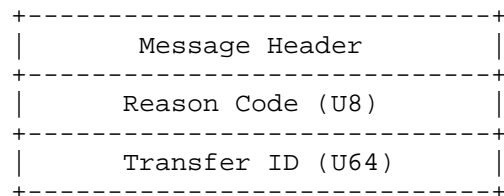


Figure 11: Format of XFER_REFUSE Messages

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	Semantics
Unknown	Reason for refusal is unknown or not specified.
Completed	The receiver now has the complete bundle. The sender MAY now consider the bundle as completely received.
No Resources	The receiver's resources are exhausted. The sender SHOULD apply reactive bundle fragmentation before retrying.
Retransmit	The receiver has encountered a problem that requires the bundle to be retransmitted in its entirety.

Table 6: XFER_REFUSE Reason Codes

The receiver MUST, for each transfer preceding the one to be refused, have either acknowledged all XFER_SEGMENTs 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 a node will not receive another XFER_SEGMENT for the same bundle after transmitting a XFER_REFUSE message since messages MAY cross on the wire; if this happens, subsequent segments of the bundle SHOULD also be refused with a XFER_REFUSE message.

Note: If a bundle transmission is aborted in this way, the receiver MAY 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' bit set to '1', indicating the start of a new transfer, and with a distinct Transfer ID value.

6. Session Termination

This section describes the procedures for ending a TCPCL session.

6.1. Shutdown Message (SHUTDOWN)

To cleanly shut down a session, a SHUTDOWN message MUST be transmitted by either node at any point following complete transmission of any other message. A receiving node SHOULD acknowledge all received data segments before sending a SHUTDOWN message to end the session. A transmitting node SHALL treat a SHUTDOWN message received mid-transfer (i.e. before the final acknowledgement) as a failure of the transfer.

After transmitting a SHUTDOWN message, an endpoint MAY immediately close the associated TCP connection. Once the SHUTDOWN message is sent, any further received data on the TCP connection SHOULD be ignored. Any delay between request to terminate the TCP connection and actual closing of the connection (a "half-closed" state) MAY be ignored by the TCPCL endpoint.

The format of the SHUTDOWN message is as follows:

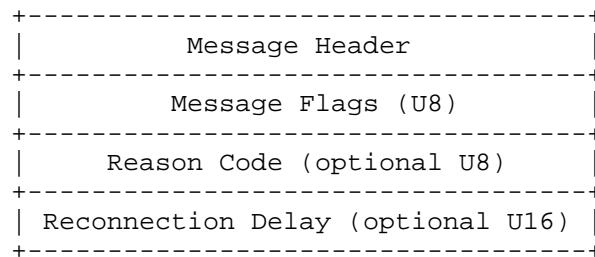


Figure 12: Format of SHUTDOWN Messages

The fields of the SHUTDOWN message are:

Message Flags: A one-octet field of single-bit flags, interpreted according to the descriptions in Table 7.

Reason Code: A one-octet refusal reason code interpreted according to the descriptions in Table 8. The Reason Code is present or absent as indicated by one of the flags.

Reconnection Delay: A 16-bit unsigned integer indicating the desired delay until further TCPCL sessions to the sending endpoint. The Reconnection Delay is present or absent as indicated by one of the flags.

Name	Code	Description
D	0x01	If bit is set, indicates that a Reconnection Delay field is present.
R	0x02	If bit is set, indicates that a Reason Code field is present.
Reserved	others	

Table 7: SHUTDOWN Flags

It is possible for a node to convey additional information regarding the reason for session termination. To do so, the node **MUST** set the 'R' bit in the message flags and transmit a one-octet reason code immediately following the message header. The specified values of the reason code are:

Name	Description
Idle timeout	The session is being closed due to idleness.
Version mismatch	The node cannot conform to the specified TCPCL protocol version.
Busy	The node is too busy to handle the current session.
Contact Failure	The node cannot interpret or negotiate contact header option.
TLS failure	The node failed to negotiate TLS session and cannot continue the session.
Resource Exhaustion	The node has run into some resource limit and cannot continue the session.

Table 8: SHUTDOWN Reason Codes

It is also possible to convey a requested reconnection delay to indicate how long the other node **MUST** wait before attempting session re-establishment. To do so, the node sets the 'D' bit in the message flags and then transmits an 16-bit unsigned integer specifying the requested delay, in seconds, following the message header (and

optionally, the SHUTDOWN reason code). The value 0 SHALL be interpreted as an infinite delay, i.e., that the connecting node MUST NOT re-establish the session. In contrast, if the node does not wish to request a delay, it SHOULD omit the reconnection delay field (and set the 'D' bit to zero).

A session shutdown MAY occur immediately after TCP connection establishment or reception of a contact header (and prior to any further data exchange). This MAY, for example, be used to notify that the node is currently not able or willing to communicate. However, a node MUST always send the contact header to its peer before sending a SHUTDOWN message.

If either node terminates a session prematurely in this manner, it SHOULD send a SHUTDOWN message and MUST indicate a reason code unless the incoming connection did not include the magic string. If the magic string was not present, a node SHOULD close the TCP connection without sending a SHUTDOWN message. If a node does not want its peer to reopen a connection immediately, it SHOULD set the 'D' bit in the flags and include a reconnection delay to indicate when the peer is allowed to attempt another session setup.

If a session is to be terminated before another protocol message has completed being sent, then the node MUST NOT transmit the SHUTDOWN message but still SHOULD close the TCP connection. This means that a SHUTDOWN cannot be used to preempt any other TCPCL messaging in-progress (particularly important when large segment sizes are being transmitted).

6.2. Idle Session Shutdown

The protocol includes a provision for clean shutdown of idle sessions. Determining the length of time to wait before closing idle sessions, if they are to be closed at all, is an implementation and configuration matter.

If there is a configured time to close idle links and if no bundle data (other than KEEPALIVE messages) has been received for at least that amount of time, then either node MAY terminate the session by transmitting a SHUTDOWN message indicating the reason code of 'Idle timeout' (as described in Table 8). After receiving a SHUTDOWN message in response, both sides MAY close the TCP connection.

7. Security Considerations

One security consideration for this protocol relates to the fact that nodes present their endpoint identifier as part of the contact header exchange. It would be possible for a node to fake this value and

present the identity of a singleton endpoint in which the node is not a member, essentially masquerading as another DTN node. If this identifier is used outside of a TLS-secured session or without further verification as a means to determine which bundles are transmitted over the session, then the node that has falsified its identity would be able to obtain bundles that it otherwise would not have. Therefore, a node SHALL NOT use the EID value of an unsecured contact header to derive a peer node's identity unless it can corroborate it via other means. When TCPCL session security is mandatory, an endpoint SHALL transmit initial unsecured contact header values indicated in Table 9 in order. These values avoid unnecessarily leaking endpoint parameters and will be ignored when secure contact header re-exchange occurs.

Parameter	Value
Flags	The USE_TLS flag is set.
Keepalive Interval	Zero, indicating no keepalive.
Segment MRU	Zero, indicating all segments are refused.
Transfer MRU	Zero, indicating all transfers are refused.
EID	Empty, indicating lack of EID.

Table 9: Recommended Unsecured Contact Header

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 mechanisms defined in [RFC6257] and [I-D.ietf-dtn-bpsec] are to be used instead.

Even when using TLS to secure the TCPCL session, the actual ciphersuite negotiated between the TLS peers MAY be insecure. TLS can be used to perform authentication without data confidentiality, for example. It is up to security policies within each TCPCL node to ensure that the negotiated TLS ciphersuite meets transport security requirements. This is identical behavior to STARTTLS use in [RFC2595].

Another consideration for this protocol relates to denial-of-service attacks. A node MAY send a large amount of data over a TCPCL session, requiring the receiving node to handle the data, attempt to stop the flood of data by sending a XFER_REFUSE message, or forcibly terminate the session. This burden could cause denial of service on

other, well-behaving sessions. There is also nothing to prevent a malicious node from continually establishing sessions and repeatedly trying to send copious amounts of bundle data. A listening node MAY take countermeasures such as ignoring TCP SYN messages, closing TCP connections as soon as they are established, waiting before sending the contact header, sending a SHUTDOWN message quickly or with a delay, etc.

8. IANA Considerations

In this section, registration procedures are as defined in [RFC5226].

Some of the registries below are created new for TCPCLv4 but share code values with TCPCLv3. This was done to disambiguate the use of these values between TCPCLv3 and TCPCLv4 while preserving the semantics of some values.

8.1. Port Number

Port number 4556 has been previously assigned as the default port for the TCP convergence layer in [RFC7242]. This assignment is unchanged by protocol version 4. Each TCPCL endpoint identifies its TCPCL protocol version in its initial contact (see Section 8.2), so there is no ambiguity about what protocol is being used.

Parameter	Value
Service Name:	dtm-bundle
Transport Protocol(s):	TCP
Assignee:	Simon Perreault <simon@per.reau.lt>
Contact:	Simon Perreault <simon@per.reau.lt>
Description:	DTN Bundle TCP CL Protocol
Reference:	[RFC7242]
Port Number:	4556

8.2. Protocol Versions

IANA has created, under the "Bundle Protocol" registry, a sub-registry titled "Bundle Protocol TCP Convergence-Layer Version

Numbers" and initialized it with the following table. The registration procedure is RFC Required.

Value	Description	Reference
0	Reserved	[RFC7242]
1	Reserved	[RFC7242]
2	Reserved	[RFC7242]
3	TCPCL	[RFC7242]
4	TCPCLbis	This specification.
5-255	Unassigned	

8.3. Header Extension Types

EDITOR NOTE: sub-registry to-be-created upon publication of this specification.

IANA will create, under the "Bundle Protocol" registry, a sub-registry titled "Bundle Protocol TCP Convergence-Layer Version 4 Header Extension Types" and initialized it with the contents of Table 10. The registration procedure is RFC Required within the lower range 0x0001--0x3fff. Values in the range 0x8000--0xffff are reserved for use on private networks for functions not published to the IANA.

Code	Message Type
0x0000	Reserved
0x0001--0x3fff	Unassigned
0x8000--0xffff	Private/Experimental Use

Table 10: Header Extension Type Codes

8.4. Message Types

EDITOR NOTE: sub-registry to-be-created upon publication of this specification.

IANA will create, under the "Bundle Protocol" registry, a sub-registry titled "Bundle Protocol TCP Convergence-Layer Version 4 Message Types" and initialized it with the contents of Table 11. The registration procedure is RFC Required.

Code	Message Type
0x00	Reserved
0x01	XFER_SEGMENT
0x02	XFER_ACK
0x03	XFER_REFUSE
0x04	KEEPALIVE
0x05	SHUTDOWN
0x06	XFER_INIT
0x07	MSG_REJECT
0x08--0xf	Unassigned

Table 11: Message Type Codes

8.5. XFER_REFUSE Reason Codes

EDITOR NOTE: sub-registry to-be-created upon publication of this specification.

IANA will create, under the "Bundle Protocol" registry, a sub-registry titled "Bundle Protocol TCP Convergence-Layer Version 4 XFER_REFUSE Reason Codes" and initialized it with the contents of Table 12. The registration procedure is RFC Required.

Code	Refusal Reason
0x0	Unknown
0x1	Completed
0x2	No Resources
0x3	Retransmit
0x4--0x7	Unassigned
0x8--0xf	Reserved for future usage

Table 12: XFER_REFUSE Reason Codes

8.6. SHUTDOWN Reason Codes

EDITOR NOTE: sub-registry to-be-created upon publication of this specification.

IANA will create, under the "Bundle Protocol" registry, a sub-registry titled "Bundle Protocol TCP Convergence-Layer Version 4 SHUTDOWN Reason Codes" and initialized it with the contents of Table 13. The registration procedure is RFC Required.

Code	Shutdown Reason
0x00	Idle timeout
0x01	Version mismatch
0x02	Busy
0x03	Contact Failure
0x04	TLS failure
0x05--0xFF	Unassigned

Table 13: SHUTDOWN Reason Codes

8.7. MSG_REJECT Reason Codes

EDITOR NOTE: sub-registry to-be-created upon publication of this specification.

IANA will create, under the "Bundle Protocol" registry, a sub-registry titled "Bundle Protocol TCP Convergence-Layer Version 4 MSG_REJECT Reason Codes" and initialized it with the contents of Table 14. The registration procedure is RFC Required.

Code	Rejection Reason
0x00	reserved
0x01	Message Type Unknown
0x02	Message Unsupported
0x03	Message Unexpected
0x04-0xFF	Unassigned

Table 14: REJECT Reason Codes

9. Acknowledgments

This memo is based on comments on implementation of [RFC7242] provided from Scott Burleigh.

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, <<http://www.rfc-editor.org/info/rfc2119>>.
- [RFC5050] Scott, K. and S. Burleigh, "Bundle Protocol Specification", RFC 5050, DOI 10.17487/RFC5050, November 2007, <<http://www.rfc-editor.org/info/rfc5050>>.
- [RFC5226] Narten, T. and H. Alvestrand, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 5226, DOI 10.17487/RFC5226, May 2008, <<http://www.rfc-editor.org/info/rfc5226>>.

- [RFC5246] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5246, DOI 10.17487/RFC5246, August 2008, <<http://www.rfc-editor.org/info/rfc5246>>.
- [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, <<http://www.rfc-editor.org/info/rfc7525>>.
- [I-D.ietf-dtn-bpbis]
Burleigh, S., Fall, K., and E. Birrane, "Bundle Protocol", draft-ietf-dtn-bpbis-06 (work in progress), October 2016.
- [refs.IANA-BP]
IANA, "Bundle Protocol registry", May 2016.

10.2. Informative References

- [RFC2595] Newman, C., "Using TLS with IMAP, POP3 and ACAP", RFC 2595, DOI 10.17487/RFC2595, June 1999, <<http://www.rfc-editor.org/info/rfc2595>>.
- [RFC4838] Cerf, V., Burleigh, S., Hooke, A., Torgerson, L., Durst, R., Scott, K., Fall, K., and H. Weiss, "Delay-Tolerant Networking Architecture", RFC 4838, DOI 10.17487/RFC4838, April 2007, <<http://www.rfc-editor.org/info/rfc4838>>.
- [RFC6257] Symington, S., Farrell, S., Weiss, H., and P. Lovell, "Bundle Security Protocol Specification", RFC 6257, DOI 10.17487/RFC6257, May 2011, <<http://www.rfc-editor.org/info/rfc6257>>.
- [RFC7242] Demmer, M., Ott, J., and S. Perreault, "Delay-Tolerant Networking TCP Convergence-Layer Protocol", RFC 7242, DOI 10.17487/RFC7242, June 2014, <<http://www.rfc-editor.org/info/rfc7242>>.
- [I-D.ietf-dtn-bpsec]
Birrane, E. and K. McKeever, "Bundle Protocol Security Specification", draft-ietf-dtn-bpsec-04 (work in progress), March 2017.

Appendix A. Significant changes from RFC7242

The areas in which changes from [RFC7242] have been made to existing headers and messages are:

- o Changed contact header content to limit number of negotiated options.
- o Added contact option to negotiate maximum segment size (per each direction).
- o Added contact header extension capability.
- o Defined new IANA registries for message / type / reason codes to allow renaming some codes for clarity.
- o Expanded Message Header to octet-aligned fields instead of bit-packing.
- o Added a bundle transfer identification number to all bundle-related messages (XFER_INIT, XFER_SEGMENT, XFER_ACK, XFER_REFUSE).
- o Use flags in XFER_ACK to mirror flags from XFER_SEGMENT.
- o Removed all uses of SDNV fields and replaced with fixed-bit-length fields.

The areas in which extensions from [RFC7242] have been made as new messages and codes are:

- o Added contact negotiation failure SHUTDOWN reason code.
- o Added MSG_REJECT message to indicate an unknown or unhandled message was received.
- o Added TLS session security mechanism.
- o Added TLS failure SHUTDOWN reason code.

Authors' Addresses

Brian Sipos
RKF Engineering Solutions, LLC
7500 Old Georgetown Road
Suite 1275
Bethesda, MD 20814-6198
US

Email: BSipos@rkf-eng.com

Michael Demmer
University of California, Berkeley
Computer Science Division
445 Soda Hall
Berkeley, CA 94720-1776
US

Email: demmer@cs.berkeley.edu

Joerg Ott
Aalto University
Department of Communications and Networking
PO Box 13000
Aalto 02015
Finland

Email: jo@netlab.tkk.fi

Simon Perreault
Quebec, QC
Canada

Email: simon@per.reau.lt