

Delay-Tolerant Networking

Internet-Draft

Intended status: Informational

Expires: May 3, 2017

E. Birrane

Johns Hopkins Applied Physics Laboratory

October 30, 2016

Asynchronous Management Architecture
draft-birrane-dtn-ama-04

Abstract

This document describes the motivation, desirable properties, system model, roles/responsibilities, and component models associated with 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 human-in-the-loop operations centers with synchronous feedback in the context of administrative 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 May 3, 2017.

Copyright Notice

Copyright (c) 2016 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	7
3.2. Current Management Approaches	8
3.3. Limitations of Current Approaches	8
4. Service Definitions	9
4.1. Configuration	9
4.2. Reporting	10
4.3. Autonomous Parameterized Control	10
4.4. Administration	11
5. Desirable Properties	11
5.1. Intelligent Push of Information	12
5.2. Minimize Message Size Not Node Processing	12
5.3. Absolute Data Identification	12
5.4. Custom Data Definition	13
5.5. Autonomous Operation	13
6. Roles and Responsibilities	13
6.1. Agent Responsibilities	13
6.2. Manager Responsibilities	15
7. System Model	15
7.1. Control and Data Flows	16
7.2. Control Flow by Role	16
7.2.1. Notation	17
7.2.2. Serialized Management	17
7.2.3. Multiplexed Management	18
7.2.4. Data Fusion	20
8. Logical Data Model	21
8.1. Data Decomposition	21
8.1.1. Groups	21
8.1.2. Levels	21
8.2. Data Model	22
8.2.1. EDDs, VARs, and Reporting	23
8.2.2. Controls and Macros	24
8.2.3. Rules	24

8.2.4. Operators and Literals	25
8.3. Application Data Model	25
9. IANA Considerations	26
10. Security Considerations	26
11. Informative References	26
Author's Address	27

1. Introduction

This document presents an Asynchronous Management Architecture (AMA) providing 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, rationale, desirable properties, and roles/responsibilities associated with an asynchronous management architecture (AMA) suitable for providing network management services in a challenged networking environment. These descriptions should be of sufficient specificity that an implementing Asynchronous Management Protocol (AMP) in conformance with this architecture will operate successfully in a challenged networking environment.

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.

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 likely that a challenged network will eventually interface with an unchallenged network, this document does not address the concept of network management compatibility with synchronous approaches. An AMP in conformance with this architecture should examine compatibility with existing approaches as part of supporting nodes acting as gateways between network types.

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

- 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 AMP 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 of logical Actors in the AMA and their associated responsibilities. It provides the terminology and context for discussing how network management services interact.
- 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.

- o Logical Component Model - This section describes those logical functions that must exist in any instantiation of an AMP.

2. 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 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 Asynchronous Management Protocol (AMP) - An application-layer protocol used to manage the Data, Controls, and other items necessary for configuration, monitoring, and administration of applications and protocols on a node in a challenged network.
- o Application Data Model (ADM) - The set of predefined data definitions, reports, literals, operations, and controls given to an Actor to manage a particular application or protocol. Actors support multiple ADMs, one for each application/protocol being managed.
- o Externally Defined Data (EDD) - Information made available to an Agent by a managed device, but not computed directly by the Agent. EDD definitions form the "lingua franca" for data within the AMA and are defined by ADMs.
- o Variable (VAR) - Information that is computed by an Agent, typically as a function of EDDs and/or other Variables. A VAR is a strongly-typed value. When a VAR is specified in an ADM, its type and default value are immutable. When a VAR is defined outside of an ADM, the type and default value may be changed. If an ADM wishes to define an item whose type and value are both immutable, that is no longer considered a Variable and should be represented as a Literal.

- 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. Similar to Externally Defined Data, Controls are defined solely in ADMs and their definition is immutable.
- o Literals (LIT) - Constants, enumerations, and other immutable definitions.
- o Macros - A named, ordered collection of Controls. When a Macro is defined in an ADM, that definition is immutable. When a Macro is defined outside of an ADM, that definition may be changed.
- 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. Operators are specified in Application Data Models (ADMs) and their definition is immutable.
- o Report Entry (RPTE) - A named, typed, ordered collection of data values gathered by one or more Agents and provided to one or more Managers. Report entries populate report templates with values.
- o Report Template (RPTT) - An ordered collection of data identifiers. When defined in an ADM the report template definition is immutable. When defined outside of an ADM, the template definition may change.
- 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 Management Approaches

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.

A near ubiquitous 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.

In challenged networks, the request/response method of data collection is neither efficient nor, at times, possible as it relies on sessions, round-trip latency, message retransmission, and ordered 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.

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. However, NETCONF does place specific constraints on any underlying transport protocol: namely, 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.

3.3. Limitations of Current Approaches

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 does 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.

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 local Agent information relating to managed applications and protocols. This information may be configured from ADMs, the specification of parameters associated with these models, and as defined by operators in the network.

New configurations received by a node must be validated to ensure that they do not conflict with other configurations at the node, or prevent the node from effectively working with other nodes 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 pre-defined Report Templates with values collected or computed by an Agent. The resultant Report Entries 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. Report Entries 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 Report Entries based on its own local time and state information. Agents should produce Report Entries 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 Entry R1 every hour (time-based production).
- o Generate Report Entry R2 when $X > 3$ (state-based production).

4.3. Autonomous Parameterized Control

Controls represent a function that can be run by an Agent to affect its behavior or otherwise change its internal state. In this context, a Control may refer to a single function or an ordered set of functions run in sequence (e.g., a macro). The set of Controls understood by an Agent define the functions available to affect the behavior of applications and protocols managed by the Agent.

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 Control 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, more specifically, triggers based on an Agent's notion of time. As such, the autonomous execution of Controls 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 Control to be executed.

The autonomous nature of Control execution by an Agent implies that the full suite of information necessary to run a Control may not be known by a Manager in advance of running the Control on an Agent. To address this situation, Controls in the AMA MUST support a

parameterization mechanism 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 control provides 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 control service 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 SNMP 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 matched 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 construting 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 the autonomy 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.

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 Data Model (ADM) Support

Agents MUST collect all data, execute all Controls, populate all Report Templates and run operations required by each ADM which the Agent claims to support. Agents MUST report

supported ADMs 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 Externally Defined Data defined in all ADMs for which they have been configured.

Autonomous Control

Agents MUST determine, without Manager intervention, whether a configured Control should be invoked. Agents MUST periodically evaluate the conditions associated with configured Controls and invoke those Controls based on local state. Agents MAY also invoke Controls 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 Variables, Report Templates, Macros and other information 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 Entry 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 Report Entry messages to a Manager, an Agent SHOULD prefer to send a single message containing multiple Report Entries.

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/ADM Mapping

Managers **MUST** understand what ADMs are supported by the various Agents with which they communicate. Managers should not attempt to request, invoke, or refer to ADM information for ADMs unsupported by an agent.

Data Collection

Managers **MUST** receive information from Agents by asynchronously configuring the production of data 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 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, such as the SNMP. 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. 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.

7.1. 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

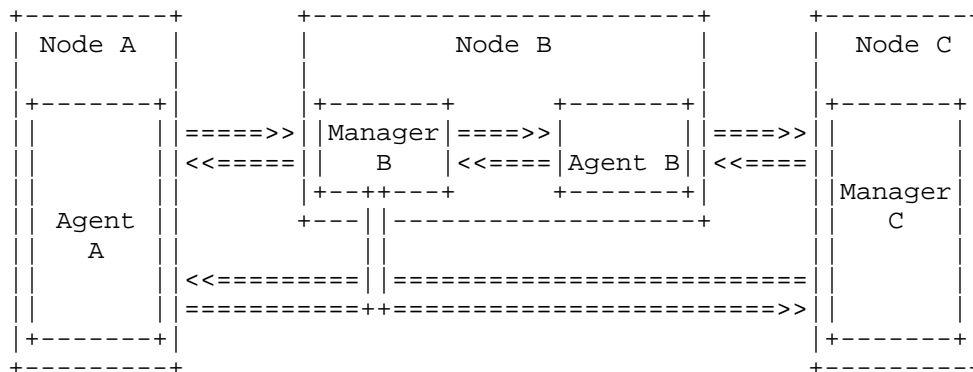


Figure 1

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.

7.2. 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.

7.2.1. Notation

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

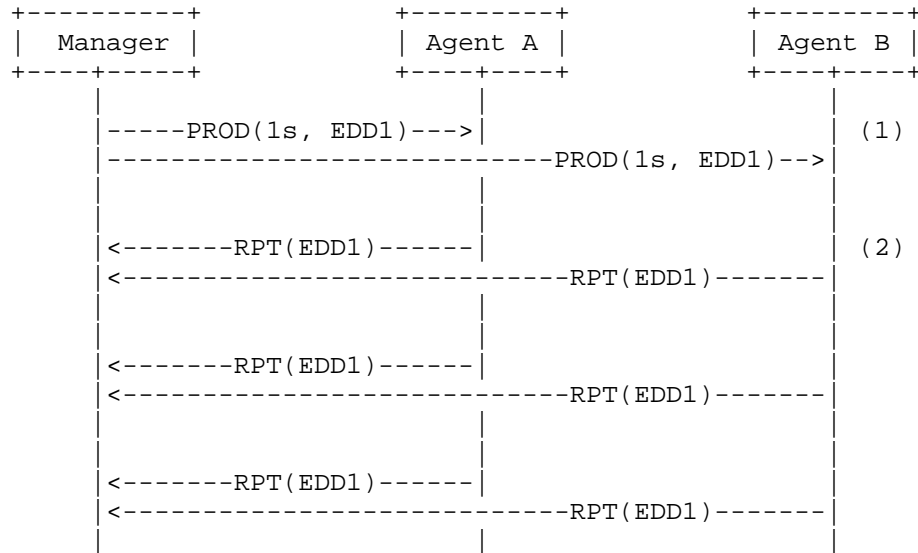
Term	Definition	Example
EDD#	EDD definition, from ADM.	EDD1
V#	Custom data definition.	V1 = EDD1 + V0.
DEF([ACL], ID,EXPR)	Define id from expression. Allow managers in access control list (ACL) to request this id.	DEF([*], V1, EDD1 + EDD2)
PROD(P,ID)	Produce ID according to predicate P. P may be a time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

Table 1: Terminology

7.2.2. 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.

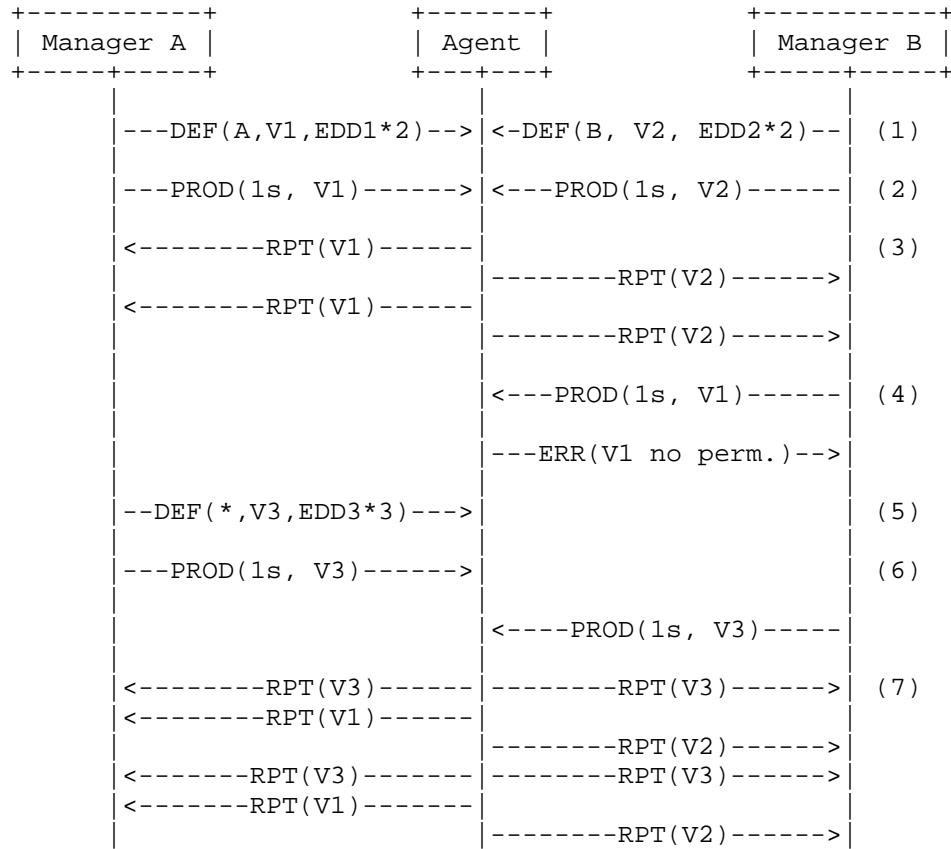
Figure 2

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).

7.2.3. Multiplexed Management

Networks spanning multiple administrative domains may require multiple Managers (for example, one per domain). When a Manager defines custom Report Templates/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.

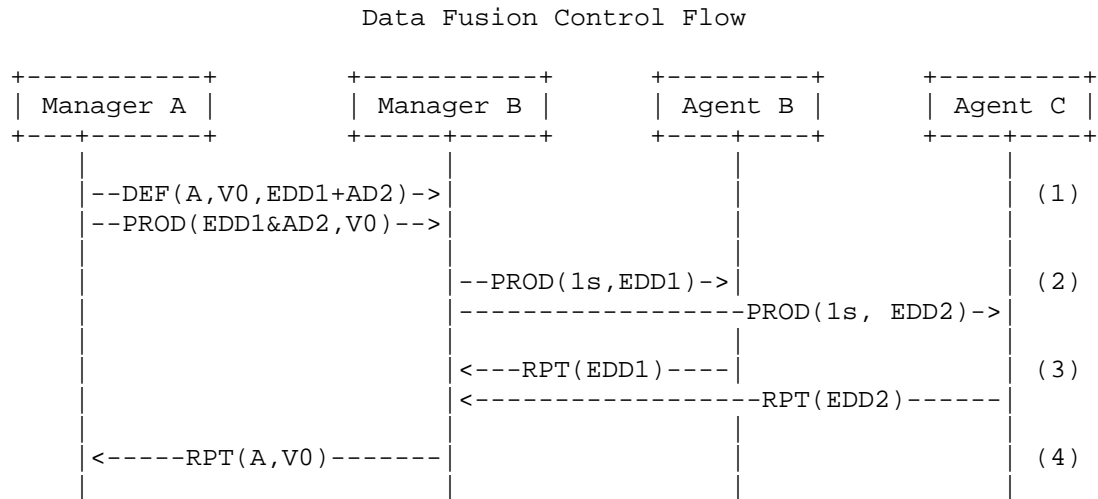
Figure 3

In more complex networks, any Manager may choose to define custom Report Templates 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).

7.2.4. 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.

8. Logical Data Model

This section identifies the different kinds of information present in an asynchronously-managed network and describes how this information should be communicated in the context of an ADM.

8.1. Data Decomposition

8.1.1. Groups

The AMA supports four basic groups of information: Data, Actions, Literals, and Operators:

Data Data values consist of information collected by an Agent and reported to Managers. This includes definitions from an ADM, derived data values as configured from Managers, and Report Entries which are collections of data elements.

Actions Actions are invoked on Agents and Managers to change behavior in response to some external event (such as local state changes or time). Actions include application-specific functions specified as part of an ADM and macros which are collections of these controls.

Literals Literals are constant numerical values that may be used in the evaluation of expressions and predicates.

Operators Operators are those mathematical functions that operate on series of Data and Literals, such as addition, subtraction, multiplication, and division.

8.1.2. Levels

The AMA defines three levels that describe the origins and multiplicity of data groups within the system. These classifications are atomic, computed, and collection.

Atomic

The Atomic level contains items computed or defined externally to the AMA and, thus, cannot be changed or otherwise decomposed by Actors within the AMA. These items are described in the context of an ADM and implemented in the context of firmware or software running on an Agent. The

identification of Atomic items MUST be globally unique and should be managed by a registration authority.

Computed

The Computed level contains items whose definition/value are specified/computed within the scope of an Actor in the AMA. Items at the computed level may be formally specified in an ADM (and therefore have definitions that are not subject to change) or may be defined dynamically on Agents by Managers and therefore have definitions that are subject to change in accordance with configuration services. In either case the definition of a Computed level item may reference other Computed level items and other Atomic level items if such inclusion does not result in a circular reference. When defined in the context of an ADM, a Computed level item MUST be globally unique and should be managed by a registration authority.

Collection

The Collection level contains items representing groups of other items, including other Collection level items. When a Collection level item definition references another Collection level item, circular references MUST be prevented. When defined in the context of an ADM, a Collection level item MUST be globally unique and should be managed by a registration authority.

8.2. Data Model

Each component of the AMA data model can be identified as a combination of group and level, as illustrated in Table 2. In this table, group/level combinations that are unsupported are listed as N/A. In this context, N/A indicates that the AMA does not require support for groups of data at a particular level for compliance.

	Data	Action	Literals	Operator
Atomic	Externally Defined Data	Control	Literal	Operator
Computed	Variable	Rule	N/A	N/A
Collection	Report Entry	Macro	N/A	N/A

Table 2

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

8.2.1. EDDs, VARs, and Reporting

Fundamental to any performance reporting function is the ability to measure the state of the Agent. Measurement may be accomplished through direct sampling of hardware, query against in-situ data stores, or other mechanisms that provide the initial quantification of state.

EDDs serve as the "lingua franca" of the management system: the unit of information that cannot be otherwise created. As such, this information serves as the basis for any user-defined (Variable) values in the system.

AMPs MAY consider the concept of the confidence of the EDD as a function of time. For example, to understand at which point a measurement should be considered stale and need to be re-measured before acting on the associated data.

While EDDs provide the full, raw set of information available to Managers and Agents there is a performance optimization to pre-computed re-used combinations of these values. Computing new values as a function of measured values simplifies operator specifications and prevents Agent implementations from continuously re-calculating the same value each time it is used in a given time period.

For example, consider a sensor node which wishes to report a temperature averaged over the past 10 measurements. An Agent may either transmit all 10 measurements to a Manager, or calculate locally the average measurement and transmit the "fused" data. Clearly, the decision to reduce data volume is highly coupled to the nature of the science and the resources of the network. For this reason, the ability to define custom computations per deployment is necessary.

Periodically, or in accordance with local state changes, Agents must collect a series of measured values and computed values and communicate them back to Managers. This ordered collection of value information is noted in this architecture as a Report Entry which populates either a pre-defined or ad-hoc Report Template. In support of hierarchical definitions, Report Entries may, themselves, contain other Report Entries. It would be incumbent on an AMP implementation to guard against circular reference in Report Template definitions.

8.2.2. Controls and Macros

Just as traditional network management approaches provide well-known identifiers for values, the AMA provides well-known identifiers for Actions. Whereas several low-latency, high-availability approaches in networks can use approaches such as remote procedure calls (RPCs), challenged networks cannot provide a similar function - Managers cannot be in the processing loop of an Agent when the Agent is not in communication with the Manager.

Controls in a system are the combination of a well-known operation that can be taken by an Agent as well as any parameters that are necessary for the proper execution of that function. For specific applications or protocols, a control specification (as a series of opcodes) can be published such that any implementing AMP accepts these opcodes and understands that sending the opcodes to an Agent supporting the application or protocol will properly execute the associated function. Parameters to such functions are provided in real-time by either Managers requesting that a control be run, pre-configured, or auto-populated by the Agent in-situ.

Often, a series of controls must be executed in concert to achieve a particular function, especially when controls represent more primitive operations for a particular application/protocol. In such scenarios, an ordered collection of controls can be specified as a Macro. In support of the hierarchical build-up of functionality, Macros may, themselves, contain other Macros, though it would be incumbent on an AMP implementation to guard against excessive recursion or other resource-intensive nesting.

8.2.3. Rules

Stimulus-response autonomy systems provide a way to pre-configure responses to anticipated events. Such a mapping from responses to events is advantageous in a challenged network for a variety of reasons, as listed below.

- o Distributed Operation - The concept of pre-configuration allows the Agent to operate without regular contact with Managers in the system. Configuration opportunities will be sporadic in any challenged network making bootstrapping of the system difficult, but this is a fundamental problem in any network scenario and any autonomy approach.
- o Deterministic Behavior - Where the mapping of stimulus to response is stable, the behavior of the Agent to a variety of in-situ state also remains stable. This stable 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.

- 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. The benefit of engine-based approaches is that the configuration inputs to the engine can be flexible without incurring a set of problematic requirements or concerns.

The logical unit of stimulus-response autonomy proposed in the AMA is a Rule of the form:

IF stimulus THEN response

Where the set of such rules, when evaluated in some prioritized sequence, provides the full set of autonomous behavior for an Agent. Stimulus in such a system would either be a function of relative time, absolute time, or some mathematical expression comprising one or more values (measurement values or computed values).

Notably, in such a system, stimuli and responses from multiple applications and protocols may be combined to provide an expressive capability.

8.2.4. Operators and Literals

Computing values or evaluating expressions requires applying mathematical operations to data known to the management system.

Operators in the AMA represent enumerated mathematical operations applied to primitive and computed values in the AMA for the purpose of creating new values. Operations may be simple binary operations such as "A + B" or more complex functions such as sin(A) or avg(A,B,C,D).

Literals represent pre-configured constants in the AMA, such as well-known mathematical numbers (e.g., PI, E), or other useful data such as Epoch times. Literals also represent asserted Primitive Values used in expressions. For example, considering the expression (A = B + 10), A would be a Variable, B would be either a Variable or EDD, + would be an Operator, and 10 would be a Literal.

8.3. Application Data Model

Application data models (ADMs) specify the data associated with a particular application/protocol. The purpose of the ADM is to provide a published interface for the management of an application or protocol independent of the nuances of its software implementation. In this respect, the ADM is conceptually similar to the Managed

Information Base (MIB) used by SNMP, but contains additional information relating to command opcodes and more expressive syntax for automated behavior.

An ADM MUST define all well-known items necessary to manage the specific application or protocol. This includes the definitions of EDDs, Variables, Report Templates, Controls, Macros, Rules, Literals, and Operators.

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
Internet-Draft
Intended status: Experimental
Expires: December 28, 2016

E. Birrane
Johns Hopkins Applied Physics Laboratory
J. Mayer
INSYEN AG
June 26, 2016

Asynchronous Management Protocol
draft-birrane-dtn-amp-03

Abstract

This document describes an Asynchronous Management Protocol (AMP) in conformance with the Asynchronous Management Architecture (AMA). The AMP provides monitoring and configuration services between managing devices (Managers) and managed devices (Agents), some of which may operate on the far side of high-delay or high-disruption links. The AMP reduces the number of transmitted bytes, operates without sessions or (concurrent) two-way links, and functions autonomously when there is no timely contact with a network operator. The AMP accomplishes this without requiring mobile code.

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 December 28, 2016.

Copyright Notice

Copyright (c) 2016 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.	Overview	3
1.2.	Technical Notes	4
1.3.	Scope	5
1.3.1.	Protocol Scope	5
1.3.2.	Specification Scope	5
1.4.	Requirements Language	6
2.	Terminology	6
3.	Data Model	6
3.1.	Primitive Types	6
3.1.1.	Standard Numeric Types	7
3.1.2.	Self-Delimiting Numeric Value (SDNV)	7
3.1.3.	Timestamp (TS)	8
3.2.	Compound Types	8
3.2.1.	Binary Large Object (BLOB)	8
3.2.2.	Data Collection (DC)	9
3.2.3.	Typed Data Collection (TDC)	9
3.2.4.	Table (TBL)	11
3.3.	Managed Identifiers (MIDs)	12
3.4.	Nicknames	16
3.5.	Parameters	17
3.5.1.	Optional Parameters	17
3.5.2.	Parameter Evaluation	17
3.6.	Special Types	19
3.6.1.	MID Collections (MC)	19
3.6.2.	Expressions (EXPR)	19
3.6.3.	Predicate (PRED)	19
4.	AMP Structures	20
4.1.	AMA Overview	20
4.2.	Externally Defined Data (EDD)	21
4.3.	Variables (VAR)	22
4.4.	Report Template (RPTT), Report Entry (RPTE)	23
4.5.	Control	26
4.6.	Time-Based Rule (TRL)	27
4.7.	State-Based Rule (SRL)	28
4.8.	Macro	30
4.9.	Literal	31
4.10.	Operator	32
5.	Data Type IDs and Enumerations	33
5.1.	Numeric Promotions	35

5.2. Numeric Conversions	36
6. Application Data Model Template	36
6.1. Overview	36
6.2. Template	36
6.2.1. ADM Metadata	36
6.2.2. ADM Information Capture	37
6.3. The Agent ADM	38
7. Functional Specification	38
7.1. Message Group Format	39
7.2. Message Format	39
7.3. Register Agent (0x00)	41
7.4. Data Report (0x12)	41
7.5. Perform Control (0x1A)	42
8. IANA Considerations	42
9. Security Considerations	42
10. References	43
10.1. Informative References	43
10.2. Normative References	43
Appendix A. Acknowledgements	43
Authors' Addresses	43

1. Introduction

This document specifies an Asynchronous Management Protocol (AMP) that provides application-layer network management service conformant to the Asynchronous Management Architecture [AMA].

1.1. Overview

Network management protocols define the messages that implement management functions amongst managed and managing devices in a network. These functions include the definition, production, and reporting of performance data, the application of administrative policy, and the configuration of behavior based on time and state measurements.

Networks whose communication links are frequently challenged by physical or administrative effects cannot guarantee the low-latency, duplex data communications necessary to support sessions and other synchronous communication. For such networks, a new protocol is required which provides familiar network management services in the absence of sessions and operator-in-the-loop control.

AMP accomplishes the network management function using open-loop, intelligent-push, asynchronous mechanisms that better scale as link challenges scale. The protocol is designed to support several desirable properties outlined in [AMA] and briefly listed below.

- o Intelligent Push of Information - The intelligent push of information eliminates the need for round-trip data exchange. This is a necessary consequence of operating in an open-loop system. AMP is designed to operate even in networks of solely unidirectional links.
- o Small Message Sizes - Smaller messages require smaller periods of viable transmission for communication, incur less retransmission cost, and consume fewer resources when persistently stored enroute in the network. AMP minimizes the size of a message whenever practical, to include packing and unpacking binary data, variable-length fields, and pre-configured data definitions.
- o Absolute and Custom Data Identification - Fine-grained identification allows data in the system to be explicitly addressed while flexible data identification allows users to define their own customized, addressed data collections. In both cases, the ability to define precisely the data required removes the need to query and transmit large data sets only to filter/downselect desired data at a receiving device.
- o Autonomous, Stateless Operation - AMP does not rely on session establishment or round-trip data exchange to perform network management functions. Wherever possible, the AMP is designed to be stateless. Where state is required, the AMP provides mechanisms to support transactions and graceful degradation when nodes in the network fail to synchronize on common definitions.
- o Compatibility with Low-Latency Network Management Protocols - AMP adopts an identifier approach compatible with the Managed Information Base (MIB) format used by Internet management protocols such as the Simple Network Management Protocol (SNMP), thus enabling management interfaces between challenged networks and unchallenged networks (such as the Internet).

1.2. Technical Notes

- o Multi-byte values in this specification are expected to be transmitted in network byte order (Big Endian).
- o Character encodings for all text-based data types will use UTF-8 encodings.
- o All data types defined by the AMP are self-terminating. This means that, given an indefinite-length octet stream, each data type can be unambiguously decoded from the stream without requiring additional information such as a length field separate from the data type definition.

- o Bit-fields in this document are specified with bit position 0 holding the least-significant bit (LSB). When illustrated in this document, the LSB appears on the right.
- o Illustrations of fields in this specification consist of the name of the field, the type of the field between []'s, and if the field is optional, the text "(opt)". An example is shown in Figure 1 below. In this illustration two fields (Field 1 and Field 2) are shown, with Field 1 of Type 1 and Field 2 of Type 2. Field 2 is also listed as being optional. Byte fields are shown in order of receipt, from left-to-right. Therefore, when transmitted on the wire, Field 1 will be received first, followed by Field 2 (if present).

+-----+-----+	
Field 1	Field 2
[TYPE 1]	[TYPE 2]
	(opt)
+-----+-----+	

Figure 1: Byte Field Formatting Example

1.3. Scope

1.3.1. Protocol Scope

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

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

1.3.2. Specification Scope

This document describes the format of the AMP messages exchanged amongst managing and managed devices in a challenged network. This document further describes the rationale behind key design decisions to the extent that such a description informs the operational deployment and configuration of an AMP implementation. This document does not address specific data configurations of AMP-enabled devices, nor does it discuss the interface between AMP and other management protocols, such as SNMP.

1.4. 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. Terminology

Note: The terms "Actor", "Agent", "Application Data Model", "Atomic Data", "Computed Data", "Control", "Literal", "Macro", "Manager", "Report Template", "Report Entry", and "Rule" are used without modification from the definitions provided in [AMA].

Additional terms critical to understanding the proper operation of the AMP are as follows.

- o Managed Item Definition (MID) - A parameterized structure used to uniquely identify all data and control definitions within the AMP. MIDs are a super-set of Object Identifiers (OIDs) and the mechanism by which the AMP maintains data compatibility with other management protocols. MIDs are defined in Section 3.3.
- o Report (RPT) - An ordered collection of report entries gathered by an Agent and provided to one or more Managers. Reports represent the fundamental unit of data exchange from an Agent to a Manager within the AMP. Report messages are defined in Section 7.4.
- o State-Based Rule (SRL) - A rule in the AMP whose action is performed if a defined predicate evaluates to true. SRLs are defined in Section 4.7.
- o Time-Based Rule (TRL) - A rule in the AMP whose action is performed at regular intervals. SRLs are defined in Section 4.6.

3. Data Model

This section identifies the data types used to capture information within the AMP.

3.1. Primitive Types

Primitive types are those that are not comprised of any other set of types known to the AMP.

3.1.1. Standard Numeric Types

The AMP supports types for unsigned bytes, 32/64-bit signed and unsigned integers, 32/64-bit floating point values, and strings, as outlined in Table 1.

AMP Type	Bit Width	Description
BYTE	8	unsigned byte value
INT	32	Signed integer in 2's complement
UINT	32	Unsigned integer in 2's complement
VAST	64	Signed integer in 2's complement
UVAST	64	Unsigned integer in 2's complement
REAL32	32	Single-precision, 32-bit floating point value in IEEE-754 format.
REAL64	64	Double-precision, 64-bit floating point value in IEEE-754 format.
STR	Varies	NULL-terminated series of characters in UTF-8 format.

Table 1: Standard Numeric Types

3.1.2. Self-Delimiting Numeric Value (SDNV)

The data type "SDNV" refers to a Self-Delimiting Numerical Value (SDNV) described in [RFC6256]. SDNVs are used in the AMP to capture any data items that are expected to be 8 bytes or less in total length. AMP Actors MAY reject any value encoded in an SDNV that is greater than 8 bytes in length.

One popular use of SDNVs in the AMP is to compress the representation of 32/64-bit integer values. This simplifies the AMP by not having to additionally support 8/16-bit versions of integers without incurring significant transmission waste when encoding small numbers into 32/64-bit representations.

3.1.3. Timestamp (TS)

A timestamp value can represent either a relative or absolute time within the AMP. An AMP relative time is defined as the number of seconds between two AMP events (such as the receipt of a control by an agent and the execution of that control). An AMP absolute time is defined as UTC time using the Unix/POSIX Epoch.

Since timestamps are a common component in AMP messages and controls, they should be made as small as possible. Therefore, timestamps in AMP do not add a special flag to determine whether the given time is an absolute or relative time. Instead, AMP defines a simple formula to unambiguously determine the type of time represented without increasing the overall size of a timestamp.

AMP uses September 9th, 2012 as the timestamp epoch (UTC time 1347148800). Times less than this value MUST be considered a relative time. Values greater than or equal to this epoch MUST be considered as absolute times. In all cases, the AMP timestamp is encoded as an SDNV to avoid the 32-bit 2038 UTC rollover problem.

The absolute time associated with a timestamp can be calculated unambiguously with the following pseudocode.

```
IF (timestamp < 1347148800) THEN
    absolute_time = current_time + timestamp
ELSE
    absolute_time = timestamp
```

3.2. Compound Types

Compound types are data types defined as an aggregation of other data types.

3.2.1. Binary Large Object (BLOB)

A Binary Large Object (BLOB) is an ordered collection of bytes prefaced by the number of bytes making up the BLOB. The format of a BLOB is illustrated in Figure 2. BLOBs are used in the AMP to capture variable data sets that are too large to efficiently store in an SDNV.

Binary Large Object Format

+-----+	+-----+	+-----+		+-----+
# Bytes	BYTE 1	BYTE 2	...	BYTE N
[SDNV]	[BYTE]	[BYTE]		[BYTE]
+-----+	+-----+	+-----+		+-----+

Figure 2: Binary Large Object Format

3.2.2. Data Collection (DC)

A Data Collection (DC) is an ordered set of BLOBs, prefaced by the number of BLOBs making up the collection. The format of a DC is illustrated in Figure 3.

Data Collection

+-----+	+-----+	+-----+		+-----+
# BLOBs	BLOB 1	BLOB 2	...	BLOB N
[SDNV]	[BLOB]	[BLOB]		[BLOB]
+-----+	+-----+	+-----+		+-----+

Figure 3: Data Collection Format

3.2.3. Typed Data Collection (TDC)

The Typed Data Collection (TDC) is a special kind of DC which encodes type information as the first BLOB in the collection. The TDC data type is used to capture typical "TLV" (type, length, value) information in the AMP.

The TDC format is illustrated in Figure 4

Typed Data Collection

+-----+	+-----+	+-----+		+-----+
# BLOBs	Type BLOB	Data BLOB 1	...	Data BLOB N
[SDNV]	[BLOB]	[BLOB]		[BLOB]
+-----+	+-----+	+-----+		+-----+

Figure 4: Typed Data Collection Format

The TDC fields are defined as follows.

BLOBs

This represents the number of BLOBS that comprise the TDC. Since the TDC has one BLOB for each data item in the collection, plus one additional BLOB for type information,

the # BLOBs value MUST be equal to one more than the number of data items in the collection.

Type BLOB

Each BYTE in the Type BLOB represents a type enumeration of a corresponding Data BLOB. For example, the 3rd BYTE in the Type BLOB holds the type enumeration of the 3rd Data BLOB in the TDC. Since there is exactly 1 byte per data item in the Type BLOB, the overall size of this BLOB MUST be the total number of data items in the TDC.

Data BLOB

The Nth Data BLOB holds the Nth data value in the collection.

For example, consider the following set of data values: {(UINT) 3, (REAL32) 3.14, (STR) "pi"}. The corresponding TDC would have 4 BLOBs. BLOB 1 would have length 3 and contain the enumerations for UINT, REAL32, and STR - encoded in one BYTE each. BLOBs 2, 3, and 4 would hold the original data. This example is illustrated in Figure 5.

Typed Data Collection Example

Data Set		TDC
+-----+ # Items = 3 +-----+		+-----+ # BLOBs = {4} +-----+
+-----+ (UINT) 3 +-----+	-----+ +-->	+-----+ TYPE BLOB = {UINT, REAL32, STR} +-----+
+-----+ (REAL32) 3.14 +-----+	-----+ +-->	+-----+ DATA BLOB 1 = {3} +-----+
+-----+ (STR) "pi" +-----+	---+ +---->	+-----+ DATA BLOB 2 = {3.14} +-----+
	+----->	+-----+ DATA BLOB 3 = {"pi"} +-----+

Figure 5: Typed Data Collection Example

The rationale for extracting data type information into a Type BLOB and placing that BLOB at the beginning of the TDC is to enable faster performance for type validators. With the Type BLOB, a validator can inspect one BLOB to ensure that the elements within the TDC match the expected type specifications. Without a Type BLOB, type information would need to be interspersed with data values throughout the TDC. In that case, a type validator would need to scan through the entire set of bytes comprising the TDC looking for type information. This would significantly alter the speed of type checking in the AMP.

The rationale for placing data values directly in a Data BLOB is to enable rapid navigation. As mentioned in Section 1.2, every data type defined in the AMP is deterministic in length. However, this determination may require deep inspection of the data in cases of variable-length headers and optional fields. By placing the data value in a Data BLOB, the length of the value may be asserted to allow a data parser to rapidly calculate the position of data item N in the TDC. The redundancy of storing a pre-calculated length for each data value when the data value length can be calculated from the data itself is a processing tradeoff made by AMP given the relative frequency with which the TDC is used to communicate Report and Control parameters.

3.2.4. Table (TBL)

A TBL is a names, typed, collection of tabular data with each row represented as a DC and each column defined by both a column name and a column type. Each row in the TBL MUST have the same length and the ith BLOB of each row DC MUST correspond to the ith column in the table.

The TBL format is illustrated in Figure 6

Table

Col Names	Col Types	# Rows	Row 1	...	Row N
[DC]	[BLOB]	[SDNV]	[DC]		[DC]

Figure 6: Table Format

The TBL fields are defined as follows.

Col Names

Column names are captured as a DC with the ith entry in the DC representing the name of the ith column. This DC MUST have a number of entries equal to the number of columns in the table. Each entry in the DC is considered to be of type STR.

NOTE: It is being considered to make this field a TDC instead of a DC to allow individual Col Names to be of different data types, instead of making them always be strings.

Col Types

Similar to the Type BLOB of the TDC, the Col Types BLOB contains one BYTE for each column in the TBL and this BYTE holds the type enumeration for the column. Therefore, the Col Types BLOB MUST have a length equal to the number of columns in the table and each BYTE in the BLOB MUST contain a correct enumeration of an AMP type. This field MUST contain one of the AMP data structure enumerations identified in Section 5.

Rows

This field captures the number of rows in the TBL. If the number of rows in the TBL is set to 0, that indicates there is no additional data after this field.

Row 1 .. Row N

Each row in the TBL is represented by a DC, with the i th BLOB in the DC representing the data in the i th column of the TBL. Each row DC MUST have a number of BLOBs equal to the number of columns.

The Figure below illustrates a table of data relating to months of the year on the left and the corresponding populated TBL structure for this table on the right.

Month	Ord	Days		Col Names DC = {"Month", "Ord", "Days"}
(STR)	(UINT)	(UINT)		Col Types BLOB = {STR, UINT, UINT}
Jan	1	31		Num Rows = 3
Oct	10	31		Row 1 DC = {"Jan", 1, 31}
June	6	30		Row 2 DC = {"Oct", 10, 31}
				Row 3 DC = {"June", 6, 30}

Figure 7: Table Example

3.3. Managed Identifiers (MIDs)

Structures defined and exchanged within the AMP must be uniquely identifiable both within a network and (when AMP is used in an overlay) across networks. This section describes the "Managed Identifier" (MID) used to provide unique naming for the AMP

structures defined in Section 4. The MID is a variable-length structure with optional fields.

The unique identifier at the core of a MID is based on the Object Identifier (OID) and its Basic Encoding Rules (BER) as identified in the ITU-T X.690 standard. The use of OIDs in the MID structure allows Agents and Managers to interface with other management schemes (such as SNMP) at management boundaries between challenged and unchallenged networks.

The MID consists of a mandatory flag BYTE, a mandatory OID, and optional annotations to assist with filtering, access control, and parameterization. The MID structure is illustrated in Figure 8.

MID format

Flags	Issuer	OID	Tag
[BYTE]	[SDNV]	[VARIED]	[SDNV]
	(opt)		(opt)

Figure 8: Managed Identifier Format

The MID fields are defined as follows.

Flags

Flags are used to describe the type of structure identified by the MID, identify which optional fields in the MID are present, and the encoding used to capture the component's OID. The layout of the flag byte is illustrated in Figure 9.

MID Flag Format

OID	TAG	ISS	STRUCT ID
7 6	5	4	3 2 1 0
MSB			LSB

Figure 9

STRUCT ID

The lower nibble of the MID flag identifies the kind of data structure being identified by this identifier. This field MUST contain one of the AMP data structure enumerations identified in Section 5.

Issuer Present (ISS)

Whether the issuer field is present (1) or not (0) for this MID. If this flag has a value of 1 then the issuer field MUST be present in the MID. Otherwise, the issuer field MUST NOT be present in the MID.

Tag Present (TAG)

Whether the tag field is present (1) or not (0) for this MID. If this flag has a value of 1 then the tag field MUST be present in the MID. Otherwise, the tag field MUST NOT be present.

OID Type (OID)

Whether the contained OID field represents a full OID (0), a parameterized OID (1), a compressed full OID (2), or a compressed, parameterized OID (3).

Issuer

This is a binary identifier representing a predetermined issuer name. The AMP protocol does not parse or validate this identifier, using it only as a distinguishing bit pattern to ensure MID uniqueness. This value, for example, may come from a global registry of organizations, an issuing node address, or some other network-unique marking. The issuer field MUST NOT be present for any MID defined as part of an ADM.

OID

The core of a MID is its encapsulated OID. Aside from the flag byte, this is the only other mandatory element within a MID. The AMP defines four types of OID references: Full OIDs, Parameterized OIDs, Compressed Full OIDs, and Compressed Parameterized OIDs, which are defined as follows.

Full OID

This is a binary representation of the full OID associated with the named value. The OID is encoded using a modified form of the ASN.1 Basic Encoding Rules (BER) for Object Identifiers (type value of 0x06). In the standard ASN.1 encoding, four octet sets are defined: identifier octets, length octets, contents octets, and end-of-contents octets. An AMP Full OID does not use the identifier, length, or end-of-contents octets. Instead, an AMP Full OID is comprised of two fields: the length in bytes of the encoded OID followed by the OID contents octets. It should be noted that this matches, exactly, the

definition of the BLOB type. The Full OID format is illustrated in Figure 10.

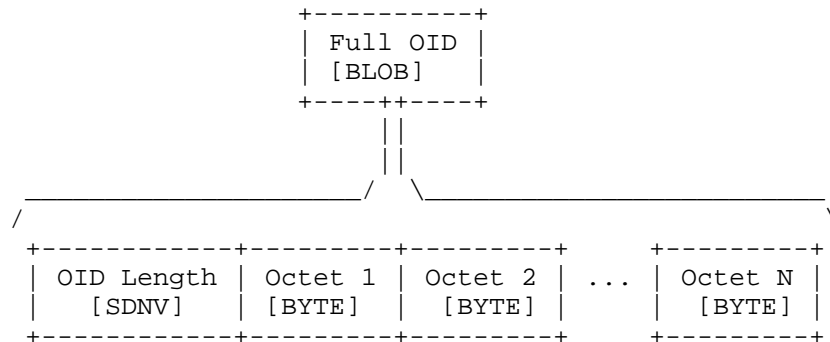


Figure 10: Full OID Format

Parameterized OID

The parameterized OID is represented as a Full OID followed by one or more parameters. Parameterized OIDs are used to templatz the specification of data items and otherwise provide parameters to Controls without requiring potentially unmanageable growth of a Full OID namespace. The format of a parameterized OID is given in Figure 11.

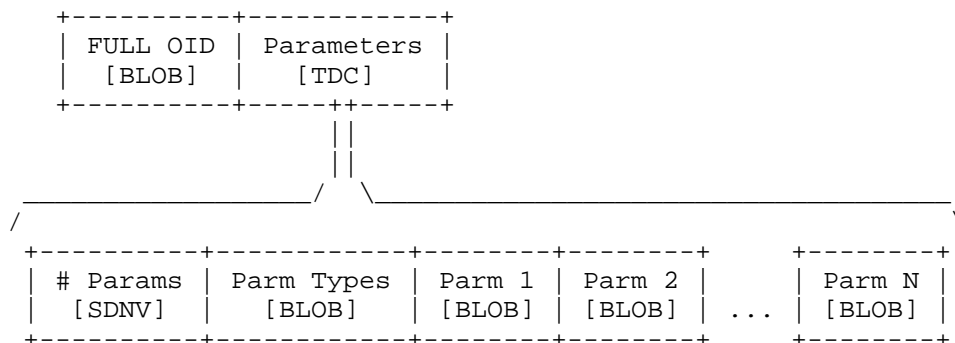


Figure 11: Parameterized OID Format

Compressed OID

Since many related OIDs share a common and lengthy hierarchy there is opportunity for significant message size savings by defining a shorthand for commonly-used portions of the OID tree. A partial

OID is a tuple consisting of a nickname for a pre-defined portion of the OID tree, followed by a relative OID. Nicknames are defined in Section 3.4. The format of a compressed OID is given in Figure 12.

Nickname	Relative OID
[SDNV]	[BLOB]

Figure 12: Compressed OID Format

Compressed Parameterized OID

A compressed, parameterized OID is similar to a compressed OID. In this instance, the tuple contained in this field is the nickname for the pre-defined portion of the OID tree (as an SDNV) followed by a parameterized OID whose hierarchy begins at the place identified by the nickname. The format of a compressed OID is given in Figure 13.

Compressed Parameterized OID Format

Nickname	Relative OID	Parameters
[SDNV]	[BLOB]	[TDC]

Figure 13: Compressed Parameterized OID Format

Tag

A value used to disambiguate multiple MIDs with the same OID/Issuer combination. The definition of the tag is left to the discretion of the MID issuer. Options for tag values include an issuer-known version number or a hashing of the data associated with the MID. The tag field MUST NOT be present for any MID defined as part of an ADM.

3.4. Nicknames

There are several strategies for reducing the overall size of an OID in an operational system. The AMP method for OID size reduction is to publish global enumerations that represent strategic nodes in an OID tree. This published, global enumeration is called a Nickname.

As mentioned in the discussion of compressed OIDs above, a nickname is used in lieu of a portion of the OID tree. ADMs may define their own nicknames so long as their definitions do not conflict with the

definitions of nicknames in other ADMs. AMP does not provide the ability to assign nicknames dynamically.

Like other numeric types, nicknames are encoded as SDNVs allowing them to be of arbitrary length. For example, 3 bytes of SDNV can encode over 2 million nicknames. Assuming ADMs are allotted 10 nicknames each, this approach can accommodate over 200,000 ADMs before requiring a 4th byte for nickname information.

Additionally, since nicknames are globally unique, neither an AMP Agent or Manager is ever required to expand a compressed OID to assert uniqueness or perform other identification. It is recommended that compressed OIDs be used whenever possible.

3.5. Parameters

Parameterized OIDs provide a powerful mechanism for customizing behavior for certain AMP structures. Parameterized values in AMP are formally defined in ADMs with a well-known, static typing. When an ADM specifies that an identified AMP structure may be parameterized, the specification MUST list the number of expected parameters and the type associated with each parameter. When a particular instance of a parameterized AMP structure is generated by an Agent or a Manager, the MID identifying that instance MUST contain a parameterized OID and the parameters associated with the OID MUST match in number and type the specification.

3.5.1. Optional Parameters

When parameterizing an AMP structure, some parameters may be optional with default values defined if parameters are omitted. The use of optional parameters helps keep MID values small when using default values for parameters is a common case, rather than forcing all parameters to be provided all the time.

Since each individual parameter in a TDC is represented as a BLOB, a parameter can be omitted by specifying a length of 0 BYTES for the Data BLOB holding the parameter. If a parameter is omitted and is not considered optional by the parameterized AMP structure, this MUST be considered an error.

3.5.2. Parameter Evaluation

The type value associated with the TDC in a parameter list is only used to provide type-checking safety to ensure that the given parameters match expected parameter types. It is important to understand that the types in the parameter TDC DO NOT define the parameterized interface - only the ADM defines the typed interface.

Parameters within the TDC may be represented in one of two ways: the parameter itself (parameter by value), or an expression used to determine the parameter (parameter by evaluation).

3.5.2.1. Parameter By Value

When specifying a parameter using a value, the BYTE representing the parameter type MUST be set to the expected parameter type and the BLOB representing the parameter contents MUST be the parameter value.

For example, consider a parameterized OID that takes 1 parameter, which it expects to be an unsigned integer (UINT). When populating this parameter by value, the type of the populated parameter field MUST be UINT and the parameter value MUST be the unsigned integer.

3.5.2.2. Parameter By Evaluation

When the value of a parameter is likely to change, an Expression (EXPR) may be substituted for the parameter value. When it comes time to interpret the parameter value, the current value of the Expression is calculated and used as the parameter value.

A parameter defined by evaluation MUST be of type EXPR, and the type of the EXPR must be equal to the expected type of the parameter. Expressions and Expression types are discussed in Section 3.6.2.

NOTE: If the expected type of the parameter is already EXPR, and a parameter of type EXPR is provided, then the system MUST treat the situation as if it were a parameter by value. AMP DOES NOT support an EXPR which references another EXPR as doing so leads to significant confusion in implementations and the possibility of circular reference.

3.5.2.3. Identifying Parameter Approach

The determination of whether a parameter has been provided by value or by evaluation is made by comparing the given type of the parameter to the expected type of the parameter.

If the parameter type and the expected type match, then the parameter MUST be considered by value. If the parameter type is an EXPR and the EXPR type matches the expected type, then the parameter MUST be considered by evaluation of the EXPR. In any other case, the parameter MUST be considered invalid as being from a type mismatch.

3.6. Special Types

In addition to the data types already mentioned, the following special data types are also defined.

3.6.1. MID Collections (MC)

A MID collection is comprised of a value identifying the number of MIDs in the collection, followed by each MID, as illustrated in Figure 14.

+-----+-----+			+-----+	
# MIDs	MID 1	...	MID N	
[UINT]	[MID]		[MID]	
+-----+-----+			+-----+	

Figure 14: MID Collection

3.6.2. Expressions (EXPR)

Expressions apply mathematical operations to values to generate new values on an Agent. The EXPR type in AMP is a collection of MIDs that represent a postfix notation stack of data, Literal, and Operator types. For example, the infix expression $A * (B * C)$ is represented as the sequence $A B C * *$. The format of an expression is illustrated in Figure 15.

+-----+-----+	
Type	Expression
[BYTE]	[MC]
+-----+-----+	

Figure 15

Type

The enumeration representing the type of the result of the evaluated expression.

Expression

An expression is represented in the AMP as a MID collection, where each MID in the ordered collection represents the data, Literals, and/or Operations that comprise the Expression.

3.6.3. Predicate (PRED)

Predicates are Expressions whose values are interpreted as a Boolean. The value of zero MUST be considered "false" and all other values MUST be considered "true".

4. AMP Structures

This section identifies the AMP structures that implement the AMA logical data model.

4.1. AMA Overview

The AMA defines a series of logical components that should be included as part of an AMP. These components are summarized from the AMA in the following table.

AMA Component	Summary Description	AMP Structure
Atomic Data	A typed, measured value whose definition and value determination occurs externally to the AMP.	Externally Defined Data
Computed Data	A typed, computed value whose definition and value determination occurs within the AMP.	Variable
Report Entry	Collection of Atomic and/or Computed data and/or other Reports.	Report Entry
Control	Parameterized opcode for any action that can be taken by an Agent.	Control
Rule	A pre-configured response to a pre-defined time or state on an Agent.	State-Based Rule, Time-Based Rule
Macro	An ordered collection of Controls.	Macro
Literal	A constant used when evaluating Rules or determining the value of Computed Data.	Literal
Operator	An opcode representing a mathematical function known to an Agent.	Operator

AMP Logical Components

The AMP implements these logical components in largely a one-to-one fashion with a few exceptions. This section describes the format of

these structures in the context of the aforementioned AMP data types. NOTE: The expression of these structures is only to describe how they appear in messages exchanged between and amongst Agents and Managers. Individual software applications may choose their own internal representation of these structures.

4.2. Externally Defined Data (EDD)

Externally defined data (EDD) are defined as part of ADMs for various applications and protocols. These represent values that are calculated outside of the context of Agents and Managers, such as those values measured by firmware. As such, their value is defined external to the AMP system.

4.2.1. Definition

The representation of these data is simply their identifying MIDs. The representation of an EDD is illustrated in Figure 16.

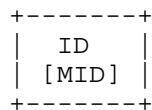


Figure 16: Externally Defined Data Format

ID

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

4.2.2. Processing

Managers

- o Store the MID for each known EDD definition.
- o Associate a data type to each known EDD definition.
- o Encode EDD MIDs in Controls to Agents, as appropriate.

Agents

- o Store the MID for each known EDD definition.
- o Associate a data type to each known EDD definition.

- o Calculate the value of an EDD definition when required, such as when generating a Report Entry or evaluating an Expression.

4.3. Variables (VAR)

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

V1 = E1 * E2

V2 = V1 + E3

4.3.1. Definition

VARs are defined by the triplet (ID, TYPE, EXPR) as illustrated in Figure 17.

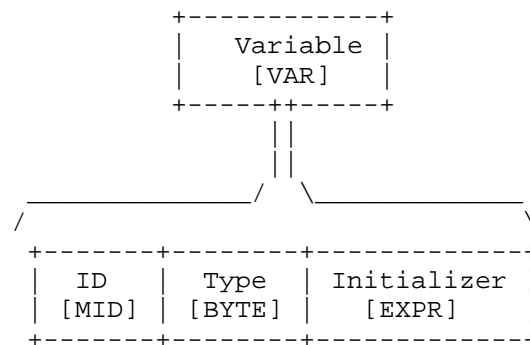


Figure 17: Variable Format

ID

This is the MID identifying the VAR. When defined in an ADM this MID MUST NOT have an ISSUER field and MUST NOT have a TAG field. When defined outside of an ADM, the MID MUST have an ISSUER field and MAY have a TAG field. This ID MUST NOT encapsulate a parameterized OID.

Type

This is the type of the VAR, and acts as a static cast for the result of the initializing Expression. Note, it is possible to specify a type different than the resultant type of the initializing Expression. For example, if an

Expression adds two single-precision floating point numbers, the VAR MAY have an integer type associated with it. This BYTE is populated with the enumeration of the associated type and MUST be defined as one of the numeric data types outlined in Section 5.

Initializer

The initial value of the VAR is given by an initializing Expression. In the case where the type of the VAR is an EXPR, then the initializer is simply copied as the value of the VAR. In the case where the type of the VAR is anything other than EXPR, then the initializer Expression is evaluated and the resultant value is copied into the VAR as its value. Once the initializer Expression has been used to calculate an initial value for the VAR it may be discarded.

4.3.2. Processing

Managers

- o Store the MID for each ADM-defined VAR definition.
- o Send requests to Agents to add, list, describe, and remove VAR definitions.
- o Remember custom VAR definitions.
- o Encode VAR MIDs in Controls to Agents, as appropriate.

Agents

- o Store the MID for each ADM-defined VAR definition.
- o Calculate the value of VARs when required, such as during Rule evaluation, calculating other VAR values, and generating Reports.
- o Add, remove, list, and describe custom VAR definitions.

4.4. Report Template (RPTT), Report Entry (RPTE)

A Report is an AMP message whose format is described in Section 7.4. This message is populated with Report Entries that contain data formatted in accordance with Report Templates.

A Report Template is the ordered set of data descriptions that describe how values will be represented in a corresponding Report Entry. Templates can be viewed as a schema that describes how to interpret a Report Entry, since these entries do not embed schema or

name information in them. Templates contain no values and are either defined in an ADM or configured between Managers and Agents.

A Report Entry is a set of data values populated using a given Report Template. A Report Entry contains only data values and no template definitions. By removing definition information from a Report Entry, the volume of information sent from the Agent to the Manager is greatly reduced. When a Report Entry is generated as capturing the result of a Control, the Report Template for the Control is assumed to be known to both the generating Agent and all receiving Managers.

4.4.1. Definition

A Report Template is modeled as a MC, as each data definition in the template is identified by a MID.

A Report Entry is a TDC identified by a MID and generated to capture the return value of a Control. Generated Report Entries **MUST** be collected by an Agent periodically, placed in an AMP Report message, and sent to one or more Managers.

When a Report Entry is generated in accordance with a named Report Template, the entry identifier **MUST** be the same as the template defining the data in the entry. When a Report Entry is generated absent a defined Report Template, then the entry identifier **MUST** be the MID of the Control generating the report.

The definition of a Report Entry is illustrated in Figure 18.

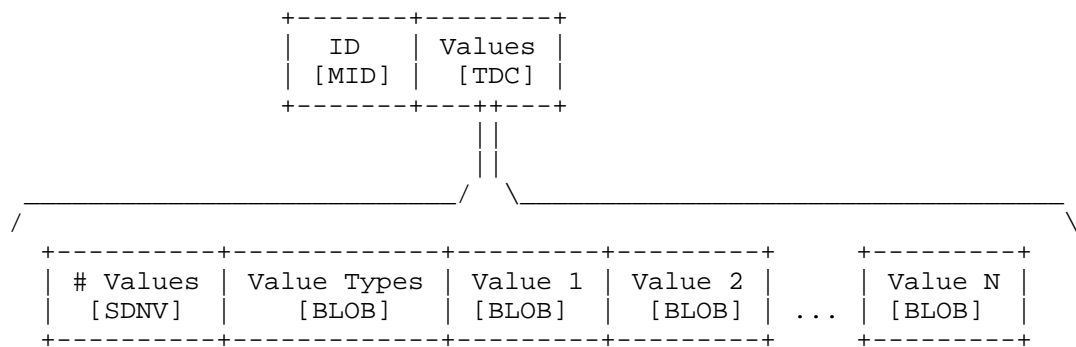


Figure 18: Report Entry Format

ID

This is the MID identifying the source used to build the entry. If this field identifies a template, and the Report

Template is defined in an ADM, this MID MUST NOT have an ISSUER field and MUST NOT have a TAG field. If the Report Template is not defined in an ADM then this MID MUST have an ISSUER field and MAY have a TAG field. If this field identifies a Control then the MID MUST NOT have an ISSUER field and MUST NOT have a TAG field.

A Report Template MID MAY be parameterized. If the Report Template MID is parameterized, the parameters MUST be used (in the same number and order) to customize any parameterized data in the report when generating values for the Report Entry.

Values

This is the TDC containing all of the data values that comprise the Report Entry. It is important to note that data values may be other Report Entries.

4.4.2. Processing

Managers

- o Store the MID for each ADM-defined Report Templates.
- o Send requests to Agents to add, list, describe, and remove custom Report Templates.
- o Remember custom Report Templates when processing Report Entries received by Agents.
- o Encode Report Template MIDs in Controls to Agents, as appropriate.

Agents

- o Store the MID for each ADM-defined Report Template.
- o Populate Report Entries for transmission to Managers when required by a Control.
- o Add, remove, list, and describe custom Report Templates.
- o Agents SHOULD collect multiple Report Entries into a single Report AMP message for transmission to a Manager rather than sending multiple, individual Report messages to a Manager with one Report Entry per Report message.

4.5. Control

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

4.5.1. Definition

The format of a Control is illustrated in Figure 19.

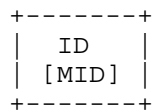


Figure 19: Control Format

ID

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

4.5.2. Processing

Managers

- o Store the MID for each ADM-defined Control definition.
- o Store the number of parameters and each parameter type for parameterized Controls.
- o Encode Control MIDs in other Controls to Agents, as appropriate.

Agents

- o Store the MID for each ADM-defined Control definition.
- o Implement Controls in firmware and run Controls with appropriate parameters when necessary in the context of Manager direction and Rule execution.
- o Communicate "return" values from Controls back to Managers as Report Entries where appropriate.

4.6. Time-Based Rule (TRL)

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

Examples of TRLs include:

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

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

4.6.1. Definition

The format of a TRL is illustrated in Figure 20.

+	-----+	-----+	-----+	-----+	-----+					
	ID		START		PERIOD		COUNT		ACTION	
	[MID]		[TS]		[UINT]		[UINT]		[MC]	
+	-----+	-----+	-----+	-----+	-----+					

Figure 20: Time-Based Rule Format

ID

This is the MID identifying the TRL. When a TRL is defined in an ADM this MID MUST NOT have an ISSUER field and MUST NOT have a TAG field. When the TRL is defined outside of an ADM, the MID MUST have an ISSUER field and MAY have a TAG field. This ID MUST NOT encapsulate a parameterized OID.

START

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

PERIOD

The number of seconds to wait between running the action associated with the TRL.

COUNT

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

ACTION

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

4.6.2. Processing**Managers**

- o Send requests to Agents to add, list, describe, and remove custom TRL definitions.
- o Remember custom TRL definitions when processing Reports received by Agents.
- o Send requests to Agents to suspend/resume the evaluation of TRLs.
- o Encode TRL MIDs in Controls to Agents, as appropriate.

Agents

- o Run the actions associated with TRLs in accordance with their start time and period.
- o Add, remove, list, and describe custom TRL definitions.
- o Suspend and resume the evaluation of a TRL when directed by a Manager or another Rule.
- o Report on the status of TRLs.

4.7. State-Based Rule (SRL)

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

Examples of SRLs include:

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

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

4.7.1. Definition

The format of a SRL is illustrated in Figure 21.

ID	START	COND	COUNT	ACTION
[MID]	[TS]	[PRED]	[UINT]	[MC]

Figure 21: State-Based Rule Format

ID

This is the MID identifying the SRL. When a report is defined in an ADM this MID MUST NOT have an ISSUER field and MUST NOT have a TAG field. When the SRL is defined outside of an ADM, the MID MUST have an ISSUER field and MAY have a TAG field. This ID MUST NOT encapsulate a parameterized OID.

START

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

CONDITION

The Predicate which, if true, results in the SRL running the associated action.

COUNT

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

ACTION

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

4.7.2. Processing

Managers

- o Send requests to Agents to add, list, describe, suspend, resume, and remove custom SRL definitions.

- o Remember custom SRL definitions when processing Report Entries received by Agents.
- o Encode SRL MIDs in Controls to Agents, as appropriate.

Agents

- o Run the actions associated with SRLs in accordance with their start time and evaluation of their predicate.
- o Add, remove, list, and describe custom SRL definitions.
- o Suspend and resume SRL evaluation when commanded by a Manager or another Rule.

4.8. Macro

Macros in the AMP are ordered collections of MIDs (an MC) that contain Controls or other Macros. When run by an Agent, each MID in the MC is run in order.

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

While the MIDs representing any given Control may be parameterized, the MID associated with a Macro MAY NOT be parameterized.

4.8.1. Definition

The format of a Macro is illustrated in Figure 22.

+-----+-----+	
ID	Definition
[MID]	[MC]
+-----+-----+	

Figure 22: Macro Format

ID

This is the MID identifying the Macro. When a Macro is defined in an ADM this MID MUST NOT have an ISSUER field and MUST NOT have a TAG field. When the Macro is defined outside of an ADM, the MID MUST have an ISSUER field and MAY have a TAG field. This ID MUST NOT encapsulate a parameterized OID.

Definition

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

4.8.2. Processing

Managers

- o Store the MID for each ADM-defined Macro definition.
- o Send requests to Agents to add, list, describe, and remove custom Macro definitions.
- o Encode macro MIDs in Controls to Agents, as appropriate.

Agents

- o Store the MID for each ADM-defined Macro definition.
- o Remember custom Macro definitions and run Macros when appropriate, such as when responding to a run-Macro Control or when executing the action of a TRL or SRL.
- o Add, remove, list, and describe custom Macro definitions.

4.9. Literal

Literals in the AMP represent constants defined in an ADM. Examples of constants that could be defined in an ADM include common mathematical values such as PI or well-known Epochs such as the UNIX Epoch.

The ADM definition of a Literal MUST include the type of the Literal value. Since ADM definitions are preconfigured on Agents and Managers in an AMA the type information for a given Literal is therefore known by all actors in the system.

If the MID identifying the Literal encapsulates a non-parameterized OID, then the value is given in the ADM and Agents and Managers can lookup this value in their set of pre-configured data.

If the MID identifying the Literal encapsulates a parameterized OID, then the parameters to the OID define the value of the Literal. Users wishing to create a new Literal will create a MID with whatever parameters are necessary to create the value. The documentation of the ADM defining the Literal MUST describe how parameters result in the calculation of the Literal value.

4.9.1. Definition

The format of a Literal is illustrated in Figure 23.

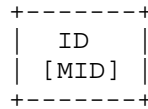


Figure 23: Control Format

ID

This is the MID identifying the Literal. Since Literal definitions are always provided in an ADM, this MID MUST NOT have an ISSUER field and MUST NOT have a TAG field.

4.9.2. Processing

Managers

- o Store the MID for each ADM-defined Literal definition.
- o Encode Literal MIDs in controls to Agents, as appropriate.

Agents

- o Store the MID for each ADM-defined Literal definition.
- o Calculate the value of Literals where appropriate, such as when generating a Report Entry or when evaluating an Expression.

4.10. Operator

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

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

4.10.1. Definition

Operators are always evaluated in the context of an Expression. The format of an Operator is illustrated in Figure 24.

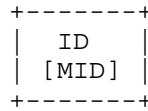


Figure 24: Operator Format

ID

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

4.10.2. Processing

Managers

- o Store the MID for each ADM-defined Operator definition.
- o Encode Operator MIDs in Controls to Agents, as appropriate.

Agents

- o Store the MID for each ADM-defined Operator definition.
- o Store the number of parameters expected for each Operator.
- o Calculate the value of applying an Operator to a given set of parameters, such as when evaluating an Expression.

5. Data Type IDs and Enumerations

This section lists the IDs and enumerations for data types outlined in this section. IDs are the text abbreviations used in this specification and in ADMs to identify data types. Enumerations associate data types with a numeric value. These enumerations MUST be used whenever a data type is represented as a numerical representation.

NOTE: Type enumerations are always represented as a BYTE in the AMP.

IDs and enumerations are grouped by the kind of data they represent, as follows. AMP structure identifiers occupy enumerations 0 - 8 and represent AMP data structures that are formally identified by a MID.

Basic data types occupy enumerations 9-18 and represent primitive data types in the AMP specification. Compound and special types occupy enumerations 19-25 and represent other data types known to the AMP specification.

AMP Structure	ID	Enumeration	Numeric
Externally Defined Data	EDD	0	No
Variable	VAR	1	No
Report	RPT	2	No
Control	CTRL	3	No
State-Based Rule	SRL	4	No
Time-Based Rule	TRL	5	No
Macro	MACRO	6	No
Literal	LIT	7	No
Operator	OP	8	No
Basic Data Type	ID	Enumeration	Numeric
BYTE	BYTE	9	No
Signed 32-bit Integer	INT	10	Yes
Unsigned 32-bit Integer	UINT	11	Yes
Signed 64-bit Integer	VAST	12	Yes
Unsigned 64-bit Integer	UVAST	13	Yes
Single-Precision Floating Point	REAL32	14	Yes
Double-Precision Floating Point	REAL64	15	Yes
Self-Delineating Numerical Value	SDNV	16	No
Timestamp	TS	17	No
Character String	STR	18	No

Compound/Special Data Type	ID	Enumeration	Numeric
Binary Large Object	BLOB	19	No
Managed Identifier	MID	20	No
MID Collection	MC	21	No
Expression	EXPR	22	No
Data Collection	DC	23	No
Typed Data Collection	TDC	24	No
Table	TBL	25	No

5.1. Numeric Promotions

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

The listing of legal promotions in the AMP are listed in Figure 25. In this Figure, operands are listed across the top row and down the first column. The resultant type of the promotion is listed in the table at their intersection.

	INT	UINT	VAST	UVAST	REAL32	REAL64
INT	INT	INT	VAST	UNK	REAL32	REAL64
UINT	INT	UINT	VAST	UVAST	REAL32	REAL64
VAST	VAST	VAST	VAST	VAST	REAL32	REAL64
UVAST	UNK	UVAST	VAST	UVAST	REAL32	REAL64
REAL32	REAL32	REAL32	REAL32	REAL32	REAL32	REAL64
REAL64	REAL64	REAL64	REAL64	REAL64	REAL64	REAL64

Figure 25: AMP Numeric Promotions

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

5.2. Numeric Conversions

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

6. Application Data Model Template

6.1. Overview

An application data model (ADM) specifies the set of AMP components associated with a particular application or protocol. The purpose of the ADM is to provide a guaranteed interface for the management of an application or protocol over AMP that is independent of the nuances of its software implementation. In this respect, the ADM is conceptually similar to the Managed Information Base (MIB) used by SNMP, but contains additional information relating to command opcodes and more expressive syntax for automated behavior.

Any implementation claiming compliance with a given ADM must collect all identified EDDs, compute all identified Variables, perform identified Controls and Macros, generate Report Entries to defined Report Templates, and understand identified Literals and Operators.

6.2. Template

Each ADM specifies the globally unique identifiers and descriptions for all EDDs, Variables, Controls, Literals, Macros, Report Templates, and Operators associated with the application or protocol managed by the ADM.

6.2.1. ADM Metadata

ADM metadata consist of the items necessary to uniquely identify the ADM itself. The required metadata items include the following.

Item	Type	Description	Req.
Name	STR	The human-readable name of the ADM.	Y
Version	STR	Version of the ADM encoded as a string.	Y
OID Nickname N	OID	ADMs provide an ordered list of nicknames that can be used by other MIDs in the ADM definition to defined compressed OIDs. There can an arbitrary number of nicknames defined for an ADM.	N

Table 2: ADM Terminology

6.2.2. ADM Information Capture

The ADM Data Section consist of all components in the "data" category associated with the managed application or protocol. The information that must be provided for each of these items is as follows.

Name

Every component in an ADM MUST be given a human-readable, consistent name that uniquely identifies the component in the context of the application or protocol. These names will be used by human-computer interfaces for manipulating components.

MID

The managed identifier that describes this data item. MIDs in components identified by an ADM MUST NOT contain an ISSUER field and MUST NOT contain a TAG field. In cases where the OID is parameterized, the parameter values are not included in the ADM MID definition as parameters are provided at runtime.

OID

A human-readable version of the OID encapsulated in the MID for the component (e.g., 1.2.3.4). When a nickname is used to represent an compressed OID, the nickname enumeration is included in this field enclosed by square brackets. For example, if OID nickname 0 refers to the OID prefix 1.2.3.4.5, then the OID 1.2.3.4.5.6 may be listed more compactly as [0].6

Description

Every component in an ADM MUST be given a human-readable, consistent description that provides a potential user with a compact, effective summary of the item.

Type

For components that evaluate to a data value, the data type for that value must be represented.

Parameters

For components with a parameterized OID, the ADM MUST provide the expected number of parameters. A value of 0 indicates that the OID has no parameters and MUST NOT be used for any MID which has a parameterized OID. When omitted, the number of parameters is considered 0.

Parameter N Name

Each parameter of a parameterized component must be given a name.

Parameter N Description

Each parameter of a parameterized component must be given a summary that describes how the parameter will be used by the application or protocol. This description MUST note if the parameter is optional.

Parameter N Type

Each parameter of a parameterized component must be given a type that describes the structure capturing the parameter value.

6.3. The Agent ADM

The full set of EDDs, Variables, Report Templates, Controls, Rules, Macros, Literals, and Operators that can be understood by an AMP Agent have been separated into an AMP Agent ADM. Just as the AMP uses ADMs to manage applications and protocols, the ADM model is also used to implement the functionality of the Agent.

7. Functional Specification

This section describes the format of the messages that comprise the AMP protocol. The AMP message specification is limited to three basic communications:

- Adding an Agent to the list of managed devices known to a Manager.
- Sending a Macro of one or more Controls to an Agent.
- Receiving a Report of one or more Report Entries from an Agent.

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

7.1. Message Group Format

Individual messages within the AMP are combined into a single group for communication with another AMP Actor. Messages within a group **MUST** be received and applied as an atomic unit. The format of a message group is illustrated in Figure 26. These message groups are assumed communicated amongst Agents and Managers as the payloads of encapsulating protocols which **MAY** provide additional security and data integrity features.

+-----+	+-----+	+-----+		+-----+
# Msgs	Timestamp	Message 1	...	Message N
[SDNV]	[TS]	[VARIES]		[VARIES]
+-----+	+-----+	+-----+		+-----+

Figure 26: AMP Message Group Format

Msgs

The number of messages that are together in this message group.

Timestamp

The creation time for this messaging group. This timestamp **MUST** be an absolute time. Individual messages may have their own creation timestamps based on their type, but the group timestamp also serves as the default creation timestamp for every message in the group.

Message N

The Nth message in the group.

7.2. Message Format

Each message identified in the AMP specification adheres to a common message format, illustrated in Figure 27, consisting of a message header, a message body, and an optional trailer.

+-----+	+-----+	+-----+
Header	Body	Trailer
[BYTE]	[VARIES]	[VARIES]
		(opt.)
+-----+	+-----+	+-----+

Figure 27: AMP Message Format

Header

The message header BYTE is shown in Figure 28. The header identifies a message context and opcode as well as flags that control whether a Report Entry should be generated on message success (Ack) and whether a Report Entry should be generated on message failure (Nack).

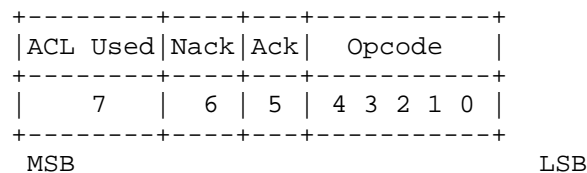


Figure 28: AMP Common Message Header

Opcode

The opcode field identifies the opcode of the message.

ACK Flag

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

NACK Flag

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

ACL Used Flag

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

Body

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

Trailer

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

7.3. Register Agent (0x00)

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

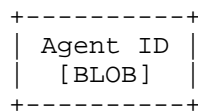


Figure 29: Register Agent Message Body

Agent ID

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

7.4. Data Report (0x12)

Reports capture information generated by Agents and transmitted to Managers in the AMP. Since the AMP is an asynchronous protocol there is no explicit association between the contents of a Report and a generating action by either a Manager or an Agent.

Reports are an ordered collection of Report Entries collected from a managed device.

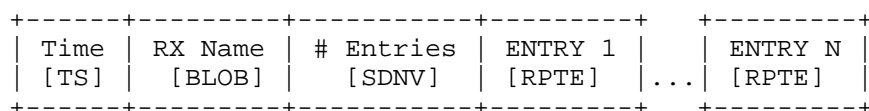


Figure 30: Data Report Message Body

Time

The time at which the Report was generated by the AMP Actor.

RX Name

The identifier of the Manager meant to receive this report.

Entries
The number of Report Entries in the Report.

ENTRY N
The Nth Report Entry.

7.5. Perform Control (0x1A)

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

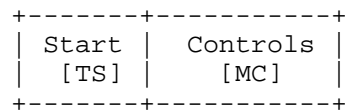


Figure 31: Perform Control Message Body

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

Controls
The collection of MIDs that represent the Controls and/or Macros to be run by the AMP Actor.

8. IANA Considerations

At this time, this protocol has no fields registered by IANA. However, such a registry MUST be established to capture certain data elements provided in ADMs, such as nicknames and root OIDs.

9. Security Considerations

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

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

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

10. References

10.1. Informative References

[AMA] Birrane, E., "Asynchronous Management Architecture", draft-birrane-dtn-ama-00 (work in progress), August 2015.

[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.

10.2. 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>>.

[RFC6256] Eddy, W. and E. Davies, "Using Self-Delimiting Numeric Values in Protocols", RFC 6256, DOI 10.17487/RFC6256, May 2011, <<http://www.rfc-editor.org/info/rfc6256>>.

Appendix A. Acknowledgements

The following participants contributed technical material, use cases, and useful thoughts on the overall approach to this protocol specification: Jeremy Pierce-Mayer of INSYEN AG contributed the concept of the typed data collection and early type checking in the protocol and has agreed to document the access control list and error reporting portion of the specification.

Authors' Addresses

Edward J. Birrane
Johns Hopkins Applied Physics Laboratory

Email: Edward.Birrane@jhuapl.edu

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

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

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

S. Burleigh
JPL, Calif. Inst. Of Technology
K. Fall
Carnegie Mellon University / SEI
E. Birrane
APL, Johns Hopkins University
October 29, 2016

Bundle Protocol
draft-ietf-dtn-bpbis-06.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 May 2, 2017.

Copyright Notice

Copyright (c) 2016 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.....	14
4. Bundle Format.....	14
4.1. BP Fundamental Data Structures.....	15
4.1.1. CRC Type.....	15
4.1.2. CRC.....	15
4.1.3. Bundle Processing Control Flags.....	15
4.1.4. Block Processing Control Flags.....	17
4.1.5. Identifiers.....	18
4.1.5.1. Endpoint ID.....	18
4.1.5.2. Node ID.....	19
4.1.6. DTN Time.....	19
4.1.7. Creation Timestamp.....	19
4.1.8. Block-type-specific Data.....	19
4.2. Bundle Representation.....	20
4.2.1. Bundle.....	20
4.2.2. Primary Bundle Block.....	20
4.2.3. Canonical Bundle Block Format.....	22
4.3. Extension Blocks.....	23
4.3.1. Current Custodian.....	24
4.3.2. Previous Node.....	24
4.3.3. Bundle Age.....	24
4.3.4. Hop Count.....	25
5. Bundle Processing.....	25
5.1. Generation of Administrative Records.....	25
5.2. Bundle Transmission.....	26
5.3. Bundle Dispatching.....	27
5.4. Bundle Forwarding.....	27
5.4.1. Forwarding Contraindicated.....	29
5.4.2. Forwarding Failed.....	29

5.5. Bundle Expiration.....	30
5.6. Bundle Reception.....	30
5.7. Local Bundle Delivery.....	31
5.8. Bundle Fragmentation.....	32
5.9. Application Data Unit Reassembly.....	33
5.10. Custody Transfer.....	34
5.10.1. Custody Acceptance.....	34
5.10.2. Custody Release.....	35
5.11. Custody Transfer Success.....	35
5.12. Custody Transfer Failure.....	35
5.13. Custody Transfer Deferral.....	36
5.14. Bundle Deletion.....	36
5.15. Discarding a Bundle.....	37
5.16. Canceling a Transmission.....	37
6. Administrative Record Processing.....	37
6.1. Administrative Records.....	37
6.1.1. Bundle Status Reports.....	38
6.1.2. Custody Signals.....	40
6.2. Generation of Administrative Records.....	43
6.3. Reception of Custody Signals.....	44
7. Services Required of the Convergence Layer.....	44
7.1. The Convergence Layer.....	44
7.2. Summary of Convergence Layer Services.....	44
8. Security Considerations.....	45
9. IANA Considerations.....	46
10. References.....	46
10.1. Normative References.....	46
10.2. Informative References.....	47
11. Acknowledgments.....	47
12. Significant Changes from RFC 5050.....	48
Appendix A. For More Information.....	49
Appendix B. CDDL expression.....	50

1. Introduction

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

It has also, as expected, identified opportunities for making the protocol simpler, more capable, and easier to use. The present document, standardizing the Bundle Protocol (BP), is adapted from RFC 5050 in that context.

This document describes version 7 of BP.

Delay Tolerant Networking is a network architecture providing communications in and/or through highly stressed environments. Stressed networking environments include those with intermittent connectivity, large and/or variable delays, and high bit error rates. To provide its services, BP 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:

- . Custodial forwarding
- . 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]. [TUT] contains a tutorial-level overview of DTN concepts.

BP's location within the standard protocol stack is as shown in Figure 1. BP uses underlying "native" 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).

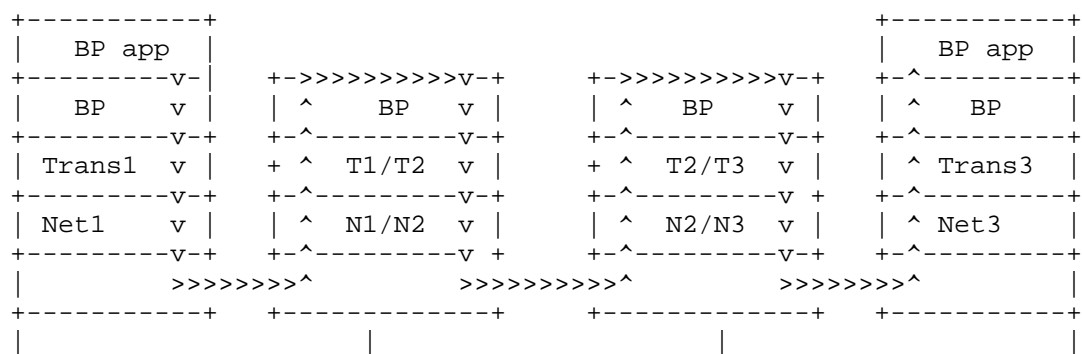




Figure 1: The Bundle Protocol in the Protocol Stack Model

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.

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.

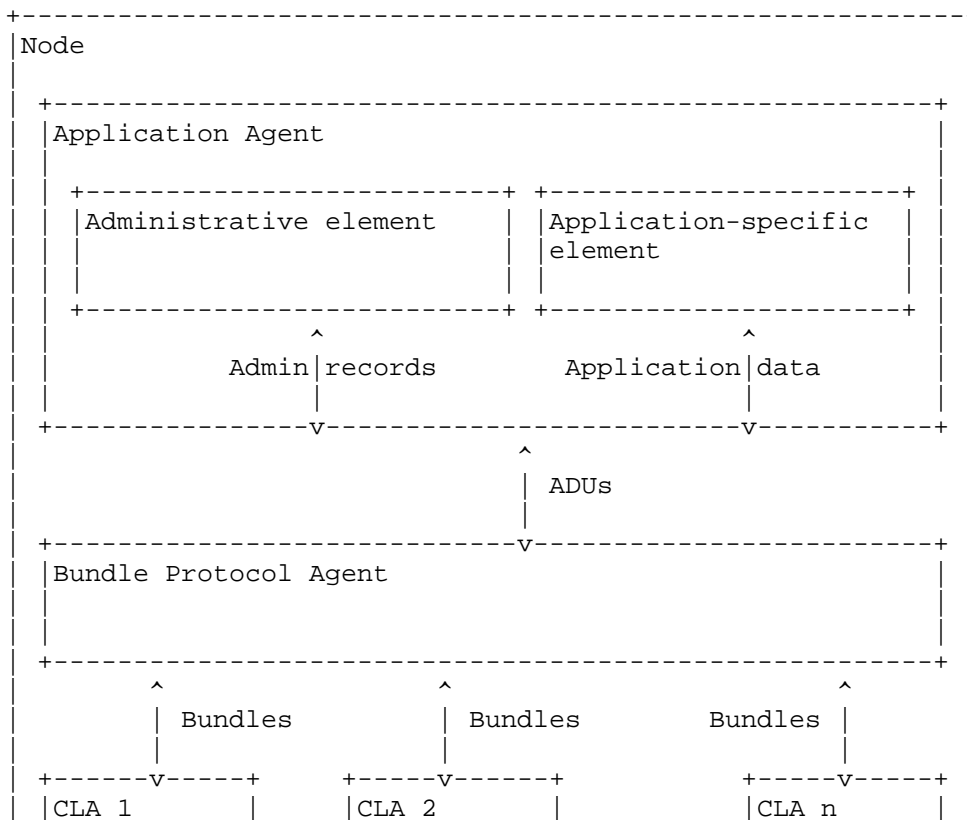
Block - A bundle protocol block is one of the protocol data structures that together constitute a well-formed bundle.

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 $0 < N < M$.

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".



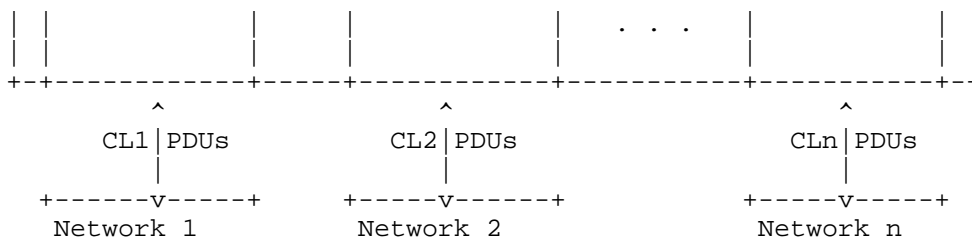


Figure 2: Components of a BP Node

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 custody signals, and accepts delivery of and processes any custody signals 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 formats of some fundamental administrative records (and of no other application data units) are defined in this specification.

Bundle endpoint - A bundle endpoint (or simply "endpoint") is a set of zero or more bundle nodes that all identify themselves for BP purposes by some common identifier, called a "bundle endpoint ID"

(or, in this document, simply "endpoint ID"; endpoint IDs are described in detail in Section 4.4.1 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).

Minimum reception group - The minimum reception group of an endpoint is the minimum number of members of the endpoint (nodes) at which the bundle must have been delivered in order for the bundle to be considered delivered to the endpoint.

Transmission - A transmission is an attempt by a node's BPA to cause copies of a bundle to be delivered at all nodes in the minimum reception group of some endpoint (the bundle's destination) in

response to a transmission request issued by the node's application agent.

Forwarding - To forward a bundle to a node is to invoke the services of a CLA 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.

Custodian - A custodian of a bundle is a node that has determined that it will retain a copy of that bundle for an indefinite period of time, forwarding and possibly re-forwarding the bundle as appropriate, until it detects one of the conditions under which it may cease being a custodian of that bundle (discussed later).

Taking custody - To take custody of a bundle is to become a custodian of that bundle.

Accepting custody - To accept custody of a bundle is to take custody of the bundle, mark the bundle in such a way as to indicate this custodianship to nodes that subsequently receive copies of the bundle, and announce this custodianship to all current custodians of the bundle.

Refusing custody - To "refuse custody" of a bundle is to notify all current custodians of that bundle that an opportunity to take custody of the bundle has been declined.

Releasing custody - To release custody of a bundle is to cease to be a custodian of the bundle.

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 is, again, wholly an implementation matter. 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.

The minimum reception group of an endpoint may be any one of the following: (a) ALL of the nodes registered in an endpoint that is permitted to contain multiple nodes (in which case forwarding to the endpoint is functionally similar to "multicast" operations in the Internet, though possibly very different in implementation); (b) ANY N of the nodes registered in an endpoint that is permitted to contain multiple nodes, where N is in the range from zero to the cardinality of the endpoint; or (c) THE SOLE NODE registered in a singleton endpoint (in which case forwarding to the endpoint is functionally similar to "unicast" operations in the Internet).

The nature of the minimum reception group for a given endpoint can typically be determined from the endpoint's ID. For some endpoint ID "schemes", the nature of the minimum reception group is fixed - in a manner that is defined by the scheme - for all endpoints identified under the scheme. For other schemes, the nature of the minimum reception group is indicated by some lexical feature of the "scheme-specific part" of the endpoint ID, in a manner that is defined by the scheme.

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

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 a CLA 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.

Whenever, for some implementation-specific reason, a node's BPA finds it impossible to immediately deliver a bundle that is deliverable, delivery of the bundle has failed. In this event, the delivery failure action associated with the applicable registration must be taken. Delivery failure action MUST be one of the following:

- . defer delivery of the bundle subject to this registration until
 - (a) this bundle is the least recently received of all bundles currently deliverable subject to this registration and (b)
 - either the registration is polled or else the registration is in the Active state; or
- . abandon delivery of the bundle subject to this registration.

An additional implementation-specific delivery deferral procedure MAY optionally be associated with the registration.

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

While the state of a registration is Active, reception of a bundle that is deliverable subject to this registration MUST cause the bundle to be delivered automatically as soon as it is the next bundle that is due for delivery according to the BPA's bundle delivery scheduling policy, an implementation matter.

Normally only registrations' registered delivery failure actions cause deliveries to be abandoned.

Custody of a bundle MAY be taken only if the destination of the bundle is a singleton endpoint. Custody MAY be released only when either (a) notification is received that some other node has accepted custody of the same bundle; (b) notification is received that the bundle has been delivered at the (sole) node registered in the bundle's destination endpoint; (c) the current custodian chooses to fragment the bundle, releasing custody of the original bundle and taking custody of the fragments instead, or (d) the bundle is explicitly deleted for some reason, such as lifetime expiration.

The custody transfer mechanism in BP provides a means of recovering from data loss along the path to the destination node. When the custodian of a bundle forwards that bundle it SHOULD set a retransmission timer; reception of a responding custody signal of any kind prior to timer expiration MUST disable that timer. Upon expiration of that timer, the custodian MUST re-forward the bundle. When a bundle for which custody has been taken arrives at a node from which it must be forwarded, and that node determines that it will forward the bundle but will not take custody, the receiving node SHOULD send a "custody delegation" signal back to the custodian indicating the next node to which the bundle will be forwarded together with an estimate of the interval that will elapse between the time the bundle was received and the time at which it will be forwarded. This mechanism is intended to facilitate accurate timeout interval calculation for this bundle.

Computation of the timeout interval for a bundle's custody transfer timer (i.e., determination of the moment at which a responding custody signal is expected) is an implementation matter and may be dynamically responsive to changes in connectivity. In some environments it may be impossible to compute this interval with

operationally satisfactory accuracy; in such environments the use of custody transfer services is contraindicated.

Alternatively, when custody transfer for a given bundle is not requested, 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. This approach may make more efficient use of links than custody transfer because a convergence-layer protocol may perform finer-grained retransmission than custody transfer does, retransmitting only the specific portions of a transmitted bundle that were not received, rather than the entire bundle. However, in some environments there may be segments of the end-to-end path for which no reliable convergence-layer protocol is available; in such environments the use of reliable convergence-layer protocols wherever possible can reduce the incidence of data loss.

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.

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 a CBOR unsigned integer.

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)
- . Custody transfer requested for this bundle. (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 custody transfer request processing.
- 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 the custody transfer requested flag value must be zero and all status report request flag values must be zero.

If the custody transfer requested flag value is 1, then the source node is requesting that every receiving node accept custody of the bundle.

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's custody transfer requested flag value must be zero, 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): custody request processing status reports are requested.
- . 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): custody transfer is requested.
- . 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)
- . Status report must be transmitted 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): status report must be transmitted 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 [URIREG] 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 9. [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, 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 (that is, nodes that are not at all moments able to determine the current UTC time to within 30 seconds), bundle creation time MUST be set to zero and the counter used as the source of the bundle sequence count MUST NEVER be reset to zero. In any case, a source Bundle Protocol Agent MUST NEVER create two distinct bundles with the same source node ID and bundle creation timestamp. The combination of source node ID and bundle creation timestamp serves to identify a single transmission request, enabling it to be acknowledged by the receiving application (provided the source node ID is not the null endpoint ID).

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

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

CRC: 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 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 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 (TBD). 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 (TBD). 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. Current Custodian

The Current Custodian block, block type 5, identifies a node that is known to have accepted custody of the bundle. The block-type-specific data of this block is the node ID of a custodian, which SHALL take the form of an endpoint ID represented as described in Section 4.1.5.2. above. The bundle MAY contain one or more occurrences of this type of block.

4.3.2. 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 MUST contain one (1) occurrence of this type of block. If present, the previous node block MUST be the FIRST block following the primary block, as the processing of other extension blocks may depend on its value.

4.3.3. 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.

4.3.4. 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 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: a bundle SHOULD be deleted when its hop count exceeds its hop limit. Procedures for determining the appropriate hop limit for a block are beyond the scope of this specification. 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 or a custody signal), the bundle protocol agent itself is responsible for causing a new bundle to be transmitted, conveying that record. In concept, the bundle protocol agent discharges this responsibility by directing the administrative element of the node's application agent to construct the record and request its transmission as detailed in Section 6 below. In practice, the manner in which administrative record generation is accomplished is an implementation matter, provided the constraints noted in Section 6 are observed.

Under some circumstances, the requesting of status reports could result in an unacceptable increase in the bundle traffic in the network. For this reason, the generation of status reports is mandatory only in two cases:

- . the reception of a bundle containing at least one block that cannot be processed, for which the value of the "transmit status report if block could not be processed" block processing flag is 1, and
- . the deletion of a bundle for which custody transfer is requested.

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

Notes on administrative record terminology:

- . A "bundle reception status report" is a bundle status report with the "reporting node received bundle" flag set to 1.
- . A "custody transfer status report" is a bundle status report with the "reporting node attempted custody transfer" flag set to 1.
- . A "bundle forwarding status report" is a bundle status report with the "reporting node forwarded the bundle" flag set to 1.
- . A "bundle delivery status report" is a bundle status report with the "reporting node delivered the bundle" flag set to 1.
- . A "bundle deletion status report" is a bundle status report with the "reporting node deleted the bundle" flag set to 1.
- . A "current custodian" of a bundle is a node identified in a Current Custodian extension block of that bundle.

5.2. Bundle Transmission

The steps in processing a bundle transmission request are:

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

Step 2: Transmission of the bundle is initiated. An outbound bundle MUST be created per the parameters of the bundle transmission request, with the retention constraint "Dispatch pending". The source node ID of the bundle MUST be either the 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 3: Processing proceeds from Step 1 of Section 5.4.

5.3. Bundle Dispatching

The steps in dispatching a bundle are:

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

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

5.4. Bundle Forwarding

The steps in forwarding a bundle are:

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

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

- . The bundle protocol agent MUST determine which node(s) to forward the bundle to. The bundle protocol agent MAY choose either to forward the bundle directly to its destination node(s) (if possible) or to forward the bundle to some other node(s) for further forwarding. The manner in which this decision is made may depend on the scheme name in the destination endpoint ID and/or 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: If the bundle's custody transfer requested flag (in the bundle processing flags field) is set to 1, then the custody transfer procedure defined in Section 5.10 MUST be followed.

Step 5: For each node selected for forwarding, the bundle protocol agent MUST invoke the services of the selected convergence layer adapter(s) in order to effect the sending of the bundle to that node. Determining the time at which the bundle 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 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, then at the last possible moment before the CLA initiates conveyance of the bundle node via the CL protocol the bundle age value MUST be increased by the difference between the current time and the time at which the bundle was received (or, if the local node is the source of the bundle, created).

Step 6: When all selected convergence layer adapters have informed the bundle protocol agent that they have concluded their data sending procedures with regard to this bundle:

- . If the "request reporting of bundle forwarding" flag in the bundle's status report request field is set to 1, then a bundle forwarding status report SHOULD be generated, destined for the bundle's report-to endpoint ID. If the bundle has the retention

constraint "custody accepted" and all of the nodes to which the bundle was forwarded are known to be unable to send bundles back to this node, then the reason code on this bundle forwarding status report MUST be "forwarded over unidirectional link"; otherwise, the reason code MUST be "no additional information".

- . The bundle's "Forward pending" retention constraint MUST be removed.

5.4.1. Forwarding Contraindicated

The steps in responding to contraindication of forwarding 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, (a) if the bundle's custody transfer requested flag (in the bundle processing flags field) is set to 1, then the custody transfer procedure defined in Section 5.10 MUST be followed; (b) when -- at some future time - the forwarding of this bundle ceases to be contraindicated, processing proceeds from Step 5 of Section 5.4.

5.4.2. Forwarding Failed

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

Step 1: If the bundle's custody transfer requested flag (in the bundle processing flags field) is set to 1, custody transfer failure must be handled. The bundle protocol agent MUST handle the custody transfer failure by generating a custody signal of type 1 (custody refusal) for the bundle, destined for the bundle's current custodian(s); the custody signal MUST contain a reason code corresponding to the reason for which forwarding was determined to be contraindicated. (Note that discarding the bundle will not delete it from the network, since each current custodian still has a copy.) In addition, if the "request reporting of custody transfer attempted" flag in the bundle's status report request field is set to 1, a custody transfer status report with the same reason code SHOULD be generated, destined for the report-to endpoint ID of the bundle.

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

Step 2: If the bundle's destination endpoint is an endpoint of which the node is a member, then the bundle's "Forward pending" retention constraint MUST be removed. Otherwise, the bundle MUST be deleted: the bundle deletion procedure defined in Section 5.14 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, then a bundle reception status report with reason code "No additional information" SHOULD be generated, destined for the bundle's report-to endpoint ID.

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

- . If the block processing flags in that block indicate that a status report is requested in this event, then a bundle reception status report with reason code "Block unintelligible" SHOULD be generated, destined for the bundle's report-to endpoint ID.

- . If the block processing flags in that block indicate that the bundle must be deleted in this event, then the bundle protocol agent MUST delete the bundle for the reason "Block unintelligible"; the bundle deletion procedure defined in Section 5.14 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 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: If the bundle's custody transfer requested flag (in the bundle processing flags field) is set to 1 and the bundle has the same source node ID, creation timestamp, and (if the bundle is a fragment) fragment offset as another bundle that (a) has not been discarded and (b) currently has the retention constraint "Custody accepted", then custody transfer redundancy MUST be handled; otherwise, processing proceeds from Step 5. The bundle protocol agent MUST handle custody transfer redundancy by generating a custody signal of type 1 (custody refusal) for this bundle with reason code "Redundant reception", destined for this bundle's current custodian, and removing this bundle's "Dispatch pending" retention constraint.

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

5.7. Local Bundle Delivery

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

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

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

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

Step 3: As soon as the bundle has been delivered:

- . If the "request reporting of bundle delivery" flag in the bundle's status report request field is set to 1, then a bundle delivery status report SHOULD be generated, destined for the bundle's report-to endpoint ID. Note that this status report only states that the payload has been delivered to the application agent, not that the application agent has processed that payload.
- . If the bundle's custody transfer requested flag (in the bundle processing flags field) is set to 1, custodial delivery MUST be reported. The bundle protocol agent MUST report custodial delivery by generating a custody signal of type 0 (custody acceptance) for the bundle, destined for the bundle's current custodian(s).

5.8. Bundle Fragmentation

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

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

Any bundle that has any Current Custodian extension block citing any node other than the local node MUST NOT be fragmented. This restriction aside, any bundle whose primary block's bundle processing flags do NOT indicate that it must not be fragmented MAY

be fragmented at any time, for any purpose, at the discretion of the bundle protocol agent.

Fragmentation SHALL be constrained as follows:

- . The concatenation of the payloads of all fragments produced by fragmentation MUST always be identical to the payload of the 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.
- . If the local node is a custodian of the fragmented bundle, then the BPA MUST release custody of the fragmented bundle before fragmentation occurs and MUST take custody of every fragment.

5.9. Application Data Unit Reassembly

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

- . This byte array -- the reassembled application data unit -- MUST replace the payload of this fragment.
- . The BPA MUST take custody of each fragmentary bundle whose payload is a subset of the reassembled application data unit,

- for which custody transfer is requested but the BPA has not yet taken custody.
- . The BPA MUST then release custody of every fragment whose payload is a subset of the reassembled application data unit, for which it has taken custody.
- . The "Reassembly pending" retention constraint MUST be removed from every other fragment whose payload is a subset of the reassembled application data unit.

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

5.10. Custody Transfer

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

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

5.10.1. Custody Acceptance

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

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

If the "request reporting of custody transfer attempted" flag in the bundle's status report request field is set to 1, a custody transfer status report with reason code 0 SHOULD be generated, destined for the report-to endpoint ID of the bundle. However, if a bundle reception status report was generated for this bundle (Step 2 of Section 5.6) but has not yet been transmitted, then this report SHOULD be generated by simply turning on the "Reporting node attempted custody transfer" flag in that earlier report.

The bundle protocol agent MUST generate a custody signal of type 0 (custody acceptance) for the bundle, destined for the bundle's current custodian(s).

The bundle protocol agent MUST assert the new current custodian for the bundle. It does so by deleting all of the bundle's existing

Current Custodian extension blocks and inserting a new Current Custodian extension block whose value is the node ID of the local node.

If the value of a custody transfer timer interval for this bundle can be calculated with operationally satisfactory accuracy, then the bundle protocol agent SHOULD set a custody transfer countdown timer for the bundle; upon expiration of this timer prior to expiration of the bundle itself and prior to custody transfer success for this bundle, the custody transfer failure procedure detailed in Section 5.12 MUST be followed. The manner in which the countdown interval for such a timer is determined is an implementation matter.

The bundle SHOULD be retained in persistent storage if possible.

5.10.2. Custody Release

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

5.11. Custody Transfer Success

Upon receipt of a custody signal of type 0 (custody acceptance) at a node that is a custodial node of the bundle identified in the custody signal, custody of the bundle MUST be released as described in Section 5.10.2.

5.12. Custody Transfer Failure

Custody transfer is determined to have failed at a custodial node for a given bundle when either (a) that node's custody transfer timer for that bundle (if any) expires or (b) a custody signal of type 1 (custody refusal) for that bundle is received at that node.

Upon determination of custody transfer failure due to expiration of a custody transfer countdown timer, the bundle protocol agent MUST re-forward the bundle, possibly on a different route (Section 5.4).

Upon determination of custody transfer failure due to reception of a custody signal of type 1 (custody refusal), the action taken by the bundle protocol agent is implementation-specific and may depend on the reason code cited for the refusal. For example, if the custody signal's reason code was "Depleted storage", the bundle protocol agent might choose to re-forward the bundle, possibly on a different route (Section 5.4). If the reason code was "Redundant reception",

on the other hand, this might cause the bundle protocol agent to release custody of the bundle and to revise its algorithm for computing countdown intervals for custody transfer timers.

5.13. Custody Transfer Deferral

Upon receipt of a bundle for which custody transfer retransmission service has been requested, which the bundle protocol agent plans to forward but for which it elects not to accept custody, the bundle protocol agent **SHOULD** generate a custody signal of type 2 (custody delegation) for the bundle, destined for the bundle's current custodian(s).

Custody transfer is determined to have been deferred at a custodial node for given bundle when a custody signal of type 2 (custody delegation) for that bundle is received at that node. The action taken by the bundle protocol agent in this event is implementation-specific. Notionally, this is an opportunity for the bundle protocol agent to revise its retransmission timeout interval for this bundle, based on the information provided in the custody signal: the next candidate custodian for the bundle is now known, and the minimum length of time before a custody acceptance signal will arrive can now be adjusted accordingly.

5.14. Bundle Deletion

The steps in deleting a bundle are:

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

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

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

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

5.15. 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.

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

Every administrative record consists of:

- . 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.		
+-----+-----+-----+-----+-----+-----+					
	2		Custody signal.		
+-----+-----+-----+-----+-----+-----+					

```
| (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 types 1 and 2 are provided below. Details of the CBOR representation of other types of administrative record type are included in the specifications defining those records.

6.1.1. Bundle Status Reports

The transmission of "bundle status reports" under specified conditions is an option that can be invoked when transmission of a bundle is requested. These reports are intended to provide information about how bundles are progressing through the system, including notices of receipt, custody transfer, 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 5 elements. The five items of the bundle status information array shall provide information on the following five status assertions, in this order:

- . Reporting node received bundle.
- . Reporting node attempted custody transfer. When this status is asserted, a reason code value of 0 ("No additional information") SHALL indicate that custody was accepted.
- . 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.

(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.

6.1.2. Custody Signals

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

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

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

+-----+-----+-----+-----+-----+	
Value	Meaning
+=====+=====+=====+=====+=====+	
0	Custody acceptance. The reporting node
	accepted custody of the bundle.
+-----+-----+-----+-----+-----+	
1	Custody refusal. The reporting node
	refused custody of the bundle.
+-----+-----+-----+-----+-----+	
2	Custody delegation: the bundle will be
	forwarded but custody was not taken.
+-----+-----+-----+-----+-----+	
(other)	Reserved for future use.
+-----+-----+-----+-----+-----+	

Figure 5: Custody Signal Type Codes

The second item of the custody signal array SHALL be additional information amplifying the signal type code, represented as a CBOR array. The number of elements in the array SHALL be either 2 (if the signal type is 2) or 1 (otherwise).

When signal type is 0 or 1, the sole item in the additional information array SHALL be a custody signal reason code, represented as a CBOR unsigned integer. Valid custody signal reason codes are defined as follows:

+-----+-----+-----+		
Value	Meaning	
+=====+=====+=====+		
0	No additional information.	
+-----+-----+-----+		
1	Reserved for future use.	
+-----+-----+-----+		
2	Reserved for future use.	
+-----+-----+-----+		
3	Redundant (reception by a node that is a	
	custodial node for this bundle).	
+-----+-----+-----+		
4	Depleted storage.	
+-----+-----+-----+		
5	Destination endpoint ID unintelligible.	
+-----+-----+-----+		
6	No known route destination from here.	
+-----+-----+-----+		
7	No timely contact with next node on route.	
+-----+-----+-----+		
8	Block unintelligible.	
+-----+-----+-----+		
(other)	Reserved for future use.	
+-----+-----+-----+		

Figure 6: Custody Signal Reason Codes

When signal type is 2, the first item in the additional information array SHALL be the node ID of the node to which the reporting node anticipates forwarding the bundle, represented as described in Section 4.1.5.2. above, and the second item in this array SHALL be an estimate of the number of seconds that will have elapsed since reception of the bundle before the anticipated forwarding begins, represented as a CBOR unsigned integer.

The third item of the custody signal array SHALL be the source node ID identifying the source of the bundle for which custodial activity is being reported, represented as described in Section 4.1.5.2. above.

The fourth item of the custody signal array SHALL be the creation timestamp of the bundle for which custodial activity is being reported, represented as described in Section 4.1.7. above.

The fifth item of the custody signal array SHALL be present if and only if the bundle for which custodial activity 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 custody signal array SHALL be present if and only if the bundle for which custodial activity 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 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.

6.3. Reception of Custody Signals

For each received custody signal that has signal type zero (custody acceptance), the administrative element of the application agent MUST direct the bundle protocol agent to follow the custody transfer success procedure in Section 5.11.

For each received custody signal that has signal type 1 (custody refusal), the administrative element of the application agent MUST direct the bundle protocol agent to follow the custody transfer failure procedure in Section 5.12.

For each received custody signal that has signal type 2 (custody delegation), the administrative element of the application agent MUST direct the bundle protocol agent to follow the custody delegation procedure in Section 5.13.

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. Security Considerations

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

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

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

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

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

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

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

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

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

In particular, while blocks MAY be added to bundles transiting intermediate nodes, removal of blocks with the 'Discard block if it can't be processed' flag 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.

9. IANA Considerations

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

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

10. References

10.1. Normative References

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

[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.

10.2. 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.

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

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

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

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

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

11. Acknowledgments

This work is freely adapted from [RFC5050], which was an effort of the Delay Tolerant Networking Research Group. The following DTNRG participants contributed significant technical material and/or inputs to that document: Dr. Vinton Cerf of Google, Scott Burleigh, Adrian Hooke, and Leigh Torgerson of the Jet Propulsion Laboratory, Michael Demmer of the University of California at Berkeley, Robert Durst, Keith Scott, and Susan Symington of The MITRE Corporation,

Kevin Fall of 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.

12. Significant Changes from RFC 5050

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

- . Clarify the difference between transmission and forwarding.
- . Amplify discussion of custody transfer. Move current custodian to an extension block, of which there can be multiple occurrences (possible support for the MITRE idea of multiple concurrent custodians, from several years ago); define that block in this specification.
- . Introduce the concept of "node ID" as functionally distinct from endpoint ID, while having the same syntax.
- . 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.
- . Clean up a conflict between fragmentation and custody transfer that Ed Birrane pointed out.

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) 2016 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: bytes]

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
    custody-transfer-status-reports-are-requested: 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
    custody-transfer-is-requested: 3
    bundle-must-not-be-fragmented: 2
    payload-is-an-administrative-record: 1
    bundle-is-a-fragment: 0
)

crc = uint

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,  
    ? 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 /= current-custodian-block  
  
current-custodian-block = [5, canonical-block-common, eid]  
  
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-attempted-custody-transfer: 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: Boolean,
    ? time-of-assertion: dtn-time)

admin-record-choice /= custody-signal

custody-signal = [2, [custody-signal-type-code: uint,
    custody-signal-information,
    admin-common]]

custody-signal-information = custody-reason-code: uint / delegation-
information

delegation-information = (
    next-hop-node: eid,
    seconds-until-forwarding: uint)
```

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

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

Delay-Tolerant Networking
Internet-Draft
Intended status: Standards Track
Expires: May 3, 2017

E. Birrane
K. McKeever
JHU/APL
October 30, 2016

Bundle Protocol Security Specification
draft-ietf-dtn-bpsec-03

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 May 3, 2017.

Copyright Notice

Copyright (c) 2016 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. Motivation	3
1.2. Supported Security Services	3
1.3. Specification Scope	4
1.4. Related Documents	5
1.5. Terminology	5
2. Key Properties	7
2.1. Block-Level Granularity	7
2.2. Multiple Security Sources	7
2.3. Mixed Security Policy	8
2.4. User-Selected Ciphersuites	8
2.5. Deterministic Processing	9
3. Security Block Definitions	9
3.1. Block Identification	10
3.2. Block Representation	10
3.3. Block Integrity Block	13
3.4. Block Confidentiality Block	14
3.5. Block Interactions	16
3.6. Parameters and Result Fields	17
3.7. BSP Block Example	18
4. Canonical Forms	20
4.1. Technical Notes	20
4.2. Primary Block Canonicalization	21
4.3. Non-Primary-Block Canonicalization	22
5. Security Processing	22
5.1. Bundles Received from Other Nodes	23
5.1.1. Receiving BCB Blocks	23
5.1.2. Receiving BIB Blocks	23
5.2. Bundle Fragmentation and Reassembly	24
6. Key Management	25
7. Policy Considerations	25
8. Security Considerations	26
8.1. Attacker Capabilities and Objectives	27
8.2. Attacker Behaviors and BPsec Mitigations	28
8.2.1. Eavesdropping Attacks	28
8.2.2. Modification Attacks	28
8.2.3. Topology Attacks	29
8.2.4. Message Injection	30
9. Ciphersuite Authorship Considerations	30
10. Defining Other Security Blocks	31
11. Conformance	32
12. IANA Considerations	32
12.1. Bundle Block Types	32
12.2. Cipher Suite Flags	32
12.3. Parameters and Results	33
13. References	34

13.1. Normative References	34
13.2. Informative References	34
Appendix A. Acknowledgements	35
Authors' Addresses	35

1. Introduction

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

1.1. Motivation

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

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

1.2. Supported Security Services

This specification supports end-to-end integrity and confidentiality services associated with BP bundles.

Integrity services ensure data within a bundle are not changed. Data changes may be caused by processing errors, environmental conditions, or intentional manipulation. An integrity service is one that provides sufficient confidence to a data receiver that data has not changed since its value was last asserted.

Confidentiality services ensure that the values of some data within a bundle can only be determined by authorized receivers of the data. When a bundle traverses a DTN, many nodes in the network other than the destination node MAY see the contents of a bundle. A confidentiality service allows a destination node to generate data values from otherwise encrypted contents of a 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 predict in the overlay and therefore makes the concept of hop-by-hop authentication difficult or impossible to enforce at the overlay.
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 can be viewed as a special case of data integrity. As such, it is possible to develop policy that provides a version of authentication using the integrity mechanisms defined in this specification.

1.3. Specification Scope

This document describes the Bundle Protocol Security Specification (BPsec), which provides security services for blocks within a bundle. This includes the data specification for individual BP extension blocks and the processing instructions for those blocks.

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

This specification does not address individual cipher suite implementations. The definition and enumeration of cipher suites should be undertaken in separate specification documents.

This specification does not address the implementation of security policy and does not provide a security policy for the BPsec. Security policies are typically based on the nature and capabilities of individual networks and network operational concepts. However, this specification does recommend 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.

1.4. Related Documents

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

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

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

The Bundle Security Protocol [RFC6257] and Streamlined Bundle Security Protocol [SBSP] introduce the concepts of security blocks for security services. BPsec is based off of these documents.

1.5. 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 those terms whose definition is important to the understanding of concepts within this specification.

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

The application of these terms applied to a sample network topology is shown in Figure 1. This figure shows four bundle nodes (BN1, BN2, BN3, BN4) residing above some transport layer(s). Three distinct transport and network protocols (T1/N1, T2/N2, and T3/N3) are also shown.

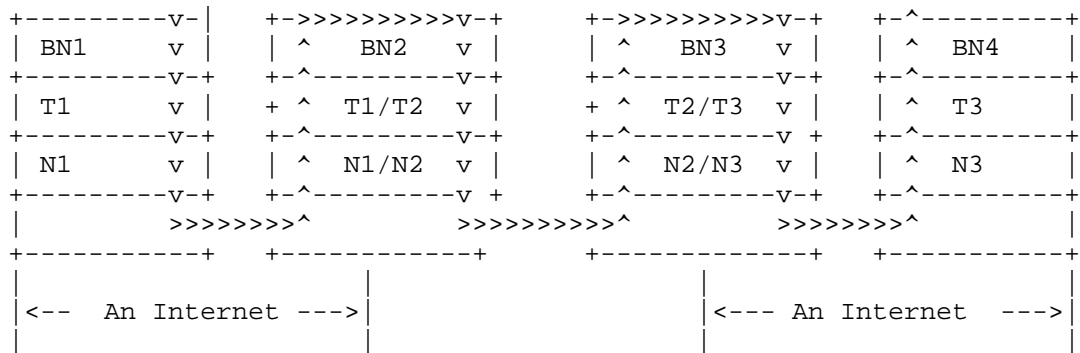


Figure 1: Bundle Nodes Sitting Above the Transport Layer.

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

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

The following security-specific terminology is also defined to clarify security operations in this specification.

- 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.
- o Security Target - the block within a bundle that receives a security-service as part of a security-operation.
- 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.

2. Key Properties

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

2.1. Block-Level Granularity

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

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

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

Each security block in a bundle will be associated with a specific security operation.

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 node adds a new extension block to a bundle, that extension block may have

security services applied to it by that waypoint. Similarly, a waypoint node may add a security service to an existing extension block, consistent with its security policy. For example, a node representing a boundary between a trusted part of the network and an untrusted part of the network may wish to apply payload encryption for bundles leaving the trusted portion of the network.

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

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

2.3. Mixed Security Policy

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

Extension blocks representing security services **MUST** have their block processing flags set such that the block will be treated appropriately by non-security-aware nodes.

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

2.4. User-Selected Ciphersuites

The security services defined in this specification rely on a variety of cipher suites providing integrity signatures, ciphertext, and other information necessary to populate security blocks. Users may wish to select different cipher suites to implement different security services. For example, some users may wish to use a SHA-256

based hash for integrity whereas other users may require a SHA-384 hash 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

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

3. Security Block Definitions

There are two types of security blocks that may be included in a bundle. These are 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 (when possible) be verified by any node in between the BIB security source and the bundle destination. BIBs MAY be added to, and removed from, bundles as a matter of security policy.

The BCB indicates that the security target(s) has been encrypted, in whole or in part, at the BCB security source in order to protect its content while in transit. The BCB may be decrypted by appropriate nodes in the network, up to and including the bundle destination, as a matter of security policy.

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

If the same security service is to be applied to multiple security targets, and cipher suite parameters for each security service are identical, then the set of security operations can be represented as a single security block with multiple security targets. In such a case, all security operations represented in the security block MUST be applied/evaluated together.

3.1. Block Identification

This specification requires that every target block of a security operation be uniquely identifiable. The definition of the extension block header from [BPBIS] provides such a mechanism in the "Block Number" field, which provides a unique identifier for a block within a bundle. Within this specification, a security target will be identified by its unique Block Number.

A security block MAY apply to multiple security targets if and only if all cipher suite parameters, security source, and key information are common for the security operation. In such a case, the security block MUST contain security results for each covered security target. The use of multiple security targets in a security block provides an efficiency mechanism so that identical ciphersuite information does not need to be repeated across multiple security blocks.

3.2. 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
- o Block Data Length
- o Block Type Specific Data Fields

The structure of the BIB and BCB Block Type Specific Data fields are identical and illustrated in Figure 2. In this figure, field names prefaced with an '*' are optional and their inclusion in the block is indicated by the Cipher Suite Flags field.

Field Name	Field Data Type
# Security Targets	Unsigned Integer
Security Targets	Array (Unsigned Integer)
Cipher Suite ID	Unsigned Integer
Cipher Suite Flags	Unsigned Integer
Security Source	URI - OPTIONAL
Cipher Parameters	Byte Array - OPTIONAL
Security Result	Byte Array

Figure 2: BIB and BCB Block Structure

Where the block fields are identified as follows.

- o # Security Targets - The number of security targets for this security block. This value MUST be at least 1.
- o Security Targets - This array contains the unique identifier of the blocks targetted by this security operation. Each security target MUST represent a block present in the bundle. A security target MUST NOT be repeated in this array.
- o Cipher suite ID - Identifies the cipher suite used to implement the security service represented by this block and applied to each security target.
- o Cipher suite flags - Identifies which optional security block fields are present in the block. The structure of the Cipher Suite Flags field is shown in Figure 3. The presence of an optional field is indicated by setting the value of the corresponding flag to one. A value of zero indicates the corresponding optional field is not present. The BPSEC Cipher Suite Flags are defined as follows.

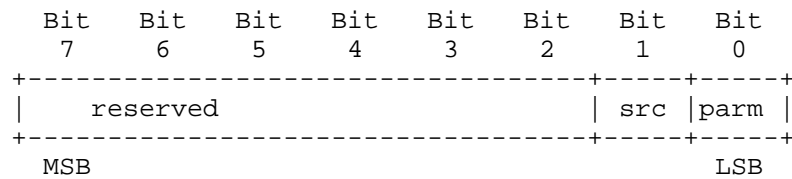


Figure 3: Cipher Suite Flags

Where:

- * bits 7-2 are reserved for future use.
 - * src - bit 1 indicates whether the Security Source is present in the block.
 - * parm - bit 0 indicates whether or not the Cipher Suite Parameters field is present in the block.
 - o (OPTIONAL) Security Source (URI) - This identifies the node that inserted the security service in the bundle. If the security source is not present then the source MAY be inferred from the bundle source, the previous hop, or some other node as defined by security policy.
 - o (OPTIONAL) Parameters (Byte Array) - Compound field of the following two items.
 - * Length (Unsigned Integer) - specifies the length of the next field, which captures the parameters data.
 - * Data (Byte Array) - A byte array encoding one or more cipher suite parameters, with each parameter represented as a Type-Length-Value (TLV) triplet, defined as follows.
 - + Type (Byte) - The parameter type.
 - + Length (Unsigned Integer) - The length of the parameter.
 - + Value (Byte Array) - The parameter value.
- See Section 3.6 for a list of parameter types that MUST be supported by BPSEC implementations. BPSEC cipher suite specifications MAY define their own parameters to be represented in this byte array.
- o Security Result (Byte Array) - A security result is the output of an appropriate cipher suite specific calculation (e.g., a

signature, Message Authentication Code (MAC), or cipher-text block key). There MUST exist one security result for each security target in the security block. A security result is a multi-field component, described as follows.

- * Total Length (Unsigned Integer) - specifies the length, in bytes, of the remaining security result information.
- * Results (Byte Array) - This field captures each of the security results, catenated together, one for each security target covered by the security block. Each result is captured by the four-tuple of (Target, Type, Len, Value). The meaning of each is given below.
 - + Target (Optional) (Unsigned Integer) - If the security block has multiple security targets, the target field is the Block Number of the security target to which this result field applies. If the security block only has a single security target, this field is omitted.
 - + Type (Unsigned Integer) - The type of security result field.
 - + Length (Unsigned Integer) - The length of the result field.
 - + Value (Byte Array) - The results of the cipher suite specific calculation.

3.3. Block Integrity Block

A BIB is an ASB with the following characteristics:

The Block Type Code value MUST be 0x02.

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

A security target for a BIB MUST NOT reference a security block defined in this specification (e.g., a BIB or a BCB).

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

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 3.6.

The security result captures the result of applying the cipher suite calculation (e.g., the MAC or signature) to the relevant parts of the security target, as specified in the cipher suite definition. This field **MUST** be present.

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

Notes:

- o Since OP(integrity, target) is allowed only once in a bundle per target, it is **RECOMMENDED** that users wishing to support multiple integrity signatures for the same target define a multi-signature cipher suite.
- 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 3.5.
- 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.

3.4. Block Confidentiality Block

A BCB is an ASB with the following characteristics:

The Block Type Code value **MUST** be 0x03.

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

A security target for a BCB MAY reference the payload block, a non-security extension block, or a BIB block. A security target in a BCB MUST NOT be another BCB.

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

Any additional bytes generated as a result of encryption and/or authentication processing of the security target SHOULD be placed in an "integrity check value" field (see Section 3.6) or other such appropriate area in the security result of the BCB.

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 3.6.

The security result MUST be present in the BCB. This compound field normally contains fields such as an encrypted bundle encryption key and/or authentication tag.

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

Fragmentation, reassembly, and custody transfer are adversely affected by a change in size of the payload due to ambiguity about what byte range of the block is actually in any particular fragment. Therefore, when the security target of a BCB is the bundle payload, the BCB MUST NOT alter the size of the payload block body data. Cipher suites SHOULD place any block expansion, such as authentication tags (integrity check values) and any padding generated by a block-mode cipher, into an integrity check value item in the security result field (see Section 3.6) of the BCB. This "in-place" encryption allows fragmentation, reassembly, and custody transfer to operate without knowledge of whether or not encryption has occurred.

Notes:

- o The cipher suite MAY process less than the entire original security target body data. If the cipher suite processes less than the complete, original security target body data, the BCB for that security target MUST specify, as part of the cipher suite parameters, which bytes of the body data are protected.

- o The BCB's "discard" flag may be set independently from its security target's "discard" flag. Whether or not the BCB's "discard" flag is set is an implementation/policy decision for the encrypting node. (The "discard" flag is more properly called the "Discard if block cannot be processed" flag.)
- o A BCB MAY include information as part of additional authenticated data to address parts of the target block, such as EID references, that are not converted to cipher-text.

3.5. 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 intermediate node would be unable to check the integrity result of a block because the block contents have been encrypted after the integrity signature was generated. To address this concern, the following processing rules MUST be followed.

- o If confidentiality is to be applied to a target, it MUST also be applied to 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 3.3, a BIB MUST NOT have a BCB as its security target. BCBs may embed integrity results as part of cipher suite parameters.

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.

3.6. Parameters and Result Fields

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

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

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

Cipher suite parameters and result fields.

Type	Name	Description	Field
0	Reserved		
1	Initialization Vector (IV)	A random value, typically eight to sixteen bytes.	Cipher Suite Parameters
2	Reserved		
3	Key Information	Material encoded or protected by the key management system and used to transport an ephemeral key protected by a long-term key.	Cipher Suite Parameters
4	Content Range	Pair of Unsigned Integers (offset,length) specifying the range of payload bytes to which an operation applies. The offset MUST be the offset within the	Cipher Suite Parameters

		original bundle, even if the current bundle is a fragment.	
5	Integrity Signatures	Result of BIB digest or other signing operation.	Security Results
6	Unassigned		
7	Salt	An IV-like value used by certain confidentiality suites.	Cipher Suite Parameters
8	BCB Integrity Check Value (ICV) / Authentication Tag	Output from certain confidentiality cipher suite operations to be used at the destination to verify that the protected data has not been modified. This value MAY contain padding if required by the cipher suite.	Security Results
9-255	Reserved		

Table 1

3.7. BSP Block Example

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

Block in Bundle	ID
Primary Block	B1
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, target=B8, B9)	B7
BIB (encrypted by B7) OP(integrity, target=B9)	B8
Payload Block	B9

Figure 4: Sample Use of BSP Blocks

In this example a bundle has four non-security-related blocks: the primary block (B1), three 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 from the prior BPSEC-BPA to this BPSEC-BPA.
- o An integrity signature on the payload (B10). This is accomplished by a BIB block (B8).

- o Confidentiality for the payload block and it's integrity signature. This is accomplished by a BCB block, B7, encrypting B8 and B9.

4. Canonical Forms

By definition, an integrity service determines whether any aspect of a block was changed from the moment the security service was applied at the security source until the point of current evaluation. To successfully verify the integrity of a block, the data passed to the verifying cipher suite **MUST** be the same bits, in the same order, as those passed to the signature-generating cipher suite at the security source.

However, [BPBIS] does not specify a single on-the-wire encoding of bundles. In cases where a security source generates a different encoding than that used at a receiving node, care **MUST** be taken to ensure that the inputs to cipher suites at the receiving node is a bitwise match to inputs provided at the security source.

This section provides guidance on how to create a canonical form for each type of block in a bundle. This form **MUST** be used when generating inputs to cipher suites for use by BPSec blocks.

This specification does not define any security operation over the entire bundle and, therefore, provides no canonical form for a serialized bundle.

4.1. Technical Notes

The following technical considerations hold for all canonicalizations in this section.

- o Any numeric fields defined as variable-length **MUST** be expanded to their "unpacked" form. For example, a 32-bit integer value **MUST** be unpacked to a four-byte representation.
- o Each block encoding **MUST** follow the CBOR encodings provided in [BPBISCBOR].
- o Canonical forms are not transmitted, they are used to generate input to a cipher suite for security processing at a security-aware node.
- o Reserved flags **MUST NOT** be included in any canonicalization as it is not known if those flags will change in transit.

- o These canonicalization algorithms assume that endpoint IDs themselves are immutable and they are unsuitable for use in environments where that assumption might be violated.
- o Cipher suites MAY define their own canonicalization algorithms and require the use of those algorithms over the ones provided in this specification. In the event of conflicting canonicalization algorithms, cipher suite algorithms take precedence over this specification.

4.2. Primary Block Canonicalization

The primary block canonical form is the same as the CBOR encoding of the block, with certain modifications to account for allowed block changes as the bundle traverses the DTN. The fields that compromise the primary block, and any special considerations for their representation in a canonical form, are as follows.

- o The Version field is included, without modification.
- o The Bundle Processing Flags field is used, with modification. Certain bundle processing flags MAY change as a bundle transits the DTN without indicating an integrity error. These flags, which are identified below, MUST NOT be represented in the canonicalized form of the bundle processing flags and, instead, be represented by the bit 0.
 - * Reserved flags.
 - * Bundle is a Fragment flag.
- o The CRC Type, Destination EID, Source Node ID, Report-To EID, Creation Timestamp, and Lifetime fields are included, without modification.
- o The fragment ID field MAY change if the bundle is fragmented in transit and, as such, this field MUST NOT be included in the canonicalization.
- o The CRC field MAY change at each hop - for example, if a bundle becomes fragmented, each fragment will have a different CRC value from the original signed primary block. As such, this field MUST NOT be included in the canonicalization.

4.3. Non-Primary-Block Canonicalization

All non-primary blocks (NPBs) in [BPBIS] share the same block structure and should be canonicalized in the same way.

Canonicalization for NPBs is dependent on whether the security operation being performed is integrity or confidentiality. Integrity operations consider every field in the block, whereas confidentiality operations only consider the block-type-specific data. Since confidentiality is applied to hide information (replacing plaintext with ciphertext) it provides no benefit to include in the confidentiality calculation information that **MUST** remain readable, such as block fields other than the block-type-specific data.

The fields that comprise a NPB, and any special considerations for their representation in a canonical form, are as follows.

- o The Block Type Code field is included, without modification, for integrity operations and omitted for confidentiality operations.
- o The Block Number field is included, without modification, for integrity operations and omitted for confidentiality operations.
- o The Block Processing Control Flags field is included, without modification, for integrity operations and omitted for confidentiality operations, with the exception of reserved flags which are treated as 0 in both cases.
- o The CRC type and CRC fields are included, without modification, for integrity operations and omitted for confidentiality operations.
- o The Block Type Specific Data field is included, without modification, for both integrity and confidentiality operations, with the exception that in some cases only a portion of the payload data is to be processed. In such a case, only those bytes are included in the canonical form and additional cipher suite parameters are required to specify which part of the field is included.

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 relevant parts of 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 body data

If a BCB contains multiple security targets, all security targets MUST be processed if the BCB is processed by the Node. The effect of this is to be the same 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 responsibility of verifying the BIB security target and whether to remove the BIB prior to delivering data to an application at the node or forwarding the bundle.

A BIB MUST NOT be processed if the security target of the BIB is also the security target of a BCB in the bundle. Given the order of operations mandated by this specification, when both a BIB and a BCB

share a security target, it means that the security target MUST have been encrypted after it was integrity signed and, therefore, the BIB cannot be verified until the security target has been decrypted by processing the BCB.

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

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

If a BIB contains multiple security targets, all security targets MUST be processed if the BIB is processed by the Node. The effect of this is to be the same 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 and security services have been applied to that bundle, the fragmentation rules described in [BPBIS] MUST be followed. As defined there and repeated here for completeness, only the payload may be fragmented; security blocks, like all extension blocks, can never be fragmented.

Due to the complexity of bundle fragmentation, including the possibility of fragmenting bundle fragments, integrity and confidentiality operations are not to be applied to a bundle

representing a fragment (i.e., a bundle whose "bundle is a Fragment" flag is set in the Bundle Processing Control Flags field). Specifically, a BCB or BIB MUST NOT be added to a bundle fragment, even if the security target of the security block is not the payload. When integrity and confidentiality must be applied to a fragment, we RECOMMEND that encapsulation be used instead.

6. Key Management

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

7. 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.

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

- o Adding a BIB to a security target that has already been encrypted by a BCB is not allowed. Therefore, we recommend three methods to add an integrity signature to an encrypted security target.
 - 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 delay-tolerant networking 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 throughout the DTN. This section provides additional discussion on security threats that BPSEC will face and describe in additional detail 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. Ciphersuite 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.

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. Conformance

All implementations are strongly RECOMMENDED to provide some method of hop-by-hop verification by generating a hash to some canonical form of the bundle and placing an integrity signature on that form using a BIB.

12. IANA Considerations

This protocol has fields that have been registered by IANA.

12.1. Bundle Block Types

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

Additional Entries for the Bundle Block-Type Codes Registry:

Value	Description	Reference
2	Block Integrity Block	This document
3	Block Confidentiality Block	This document

Table 2

12.2. Cipher Suite Flags

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

The registration policy for this registry is: Specification Required

The Value range is: Variable Length

Cipher Suite Flag Registry:

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

Table 3

12.3. Parameters and Results

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

The registration policy for this registry is: Specification Required

The Value range is: 8-bit unsigned integer.

Cipher Suite Parameters and Results Type Registry:

Value	Description	Reference
0	reserved	Section 3.6
1	initialization vector (IV)	Section 3.6
2	reserved	Section 3.6
3	key information	Section 3.6
4	content-range (pair of Unsigned Integers)	Section 3.6
5	integrity signature	Section 3.6
6	unassigned	Section 3.6
7	salt	Section 3.6
8	BCB integrity check value (ICV)	Section 3.6
9-191	reserved	Section 3.6
192-250	private use	Section 3.6
251-255	reserved	Section 3.6

Table 4

13. References

13.1. Normative References

- [BPBIS] Burleigh, S., Fall, K., and E. Birrane, "Bundle Protocol", draft-ietf-dtn-bpbis-04 (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.

13.2. Informative References

- [BPBISCBOR] Burleigh, S., "Bundle Protocol CBOR Representation Specification", draft-burleigh-dtn-rs-cbor-01 (work in progress), April 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: RFC7242 (if approved)
Intended status: Standards Track
Expires: April 22, 2017

B. Sipos
RKF Engineering
M. Demmer
UC Berkeley
J. Ott
Aalto University
S. Perreault
October 19, 2016

Delay-Tolerant Networking TCP Convergence Layer Protocol Version 4
draft-ietf-dtn-tcpclv4-00

Abstract

This document describes a revised protocol for the TCP-based convergence layer for Delay-Tolerant Networking (DTN). The protocol revision is based on implementation issues in the original [RFC7242] and updates to the Bundle Protocol contents, encodings, and convergence layer requirements in [I-D.ietf-dtn-bpbis]. The majority of this specification is unchanged from TCPCL version 3.

Status of This Memo

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

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

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

This Internet-Draft will expire on April 22, 2017.

Copyright Notice

Copyright (c) 2016 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	6
4. Session Establishment	7
4.1. Contact Header	8
4.2. Validation and Parameter Negotiation	10
5. Established Session Operation	11
5.1. Message Type Codes	11
5.2. Upkeep and Status Messages	12
5.2.1. Session Upkeep (KEEPALIVE)	12
5.2.2. Message Rejection (REJECT)	13
5.3. Session Security	14
5.3.1. TLS Handshake Result	14
5.3.2. Example TLS Initiation	15
5.4. Bundle Transfer	16
5.4.1. Bundle Transfer ID	16
5.4.2. Bundle Length (LENGTH)	16
5.4.3. Bundle Data Transmission (DATA_SEGMENT)	17
5.4.4. Bundle Acknowledgments (ACK_SEGMENT)	18
5.4.5. Bundle Refusal (REFUSE_BUNDLE)	19
6. Session Termination	20
6.1. Shutdown Message (SHUTDOWN)	21
6.2. Idle Session Shutdown	23
7. Security Considerations	23
8. IANA Considerations	24
8.1. Port Number	24
8.2. Protocol Versions	25
8.3. Message Types	25
8.4. REFUSE_BUNDLE Reason Codes	26
8.5. SHUTDOWN Reason Codes	27
8.6. REJECT Reason Codes	27
9. Acknowledgments	28
10. References	28
10.1. Normative References	28
10.2. Informative References	29
Appendix A. Significant changes from RFC7242	29
Authors' Addresses	30

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.

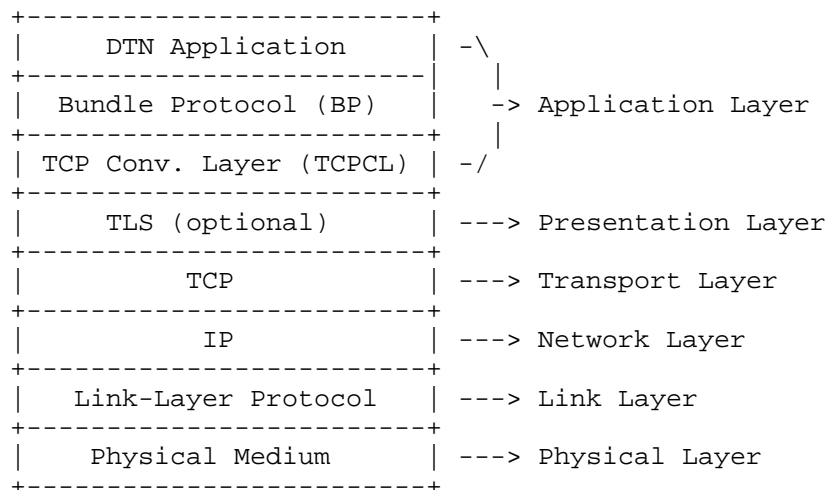


Figure 1: The Locations of the Bundle Protocol and the TCP Convergence-Layer Protocol above the Internet Protocol Stack

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 when a bundle node initiates a TCP connection to be established 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.

Transmission: Transmission refers to the procedures and mechanisms (described below) for conveyance of a bundle from one node to another.

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 transmitted in either direction. Each bundle is transmitted in one or more logical segments of formatted bundle data. Each logical data segment consists of a DATA_SEGMENT message header, 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. The first segment for a bundle MUST set the 'start' flag, and the last one MUST set the 'end' flag in the DATA_SEGMENT message header.

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 (ACK_SEGMENT). 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 ACK_SEGMENT code followed by a count containing the cumulative length of the bundle that has been received. The sending node MAY transmit multiple DATA_SEGMENT messages without necessarily waiting for the corresponding ACK_SEGMENT 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 REFUSE_BUNDLE

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 (ACK_SEGMENT or REFUSE_BUNDLE) but no further data messages (DATA_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 DATA_SEGMENT messages without necessarily waiting for the corresponding ACK_SEGMENT 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 DATA_SEGMENT messages for different bundles without necessarily waiting for ACK_SEGMENT 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.

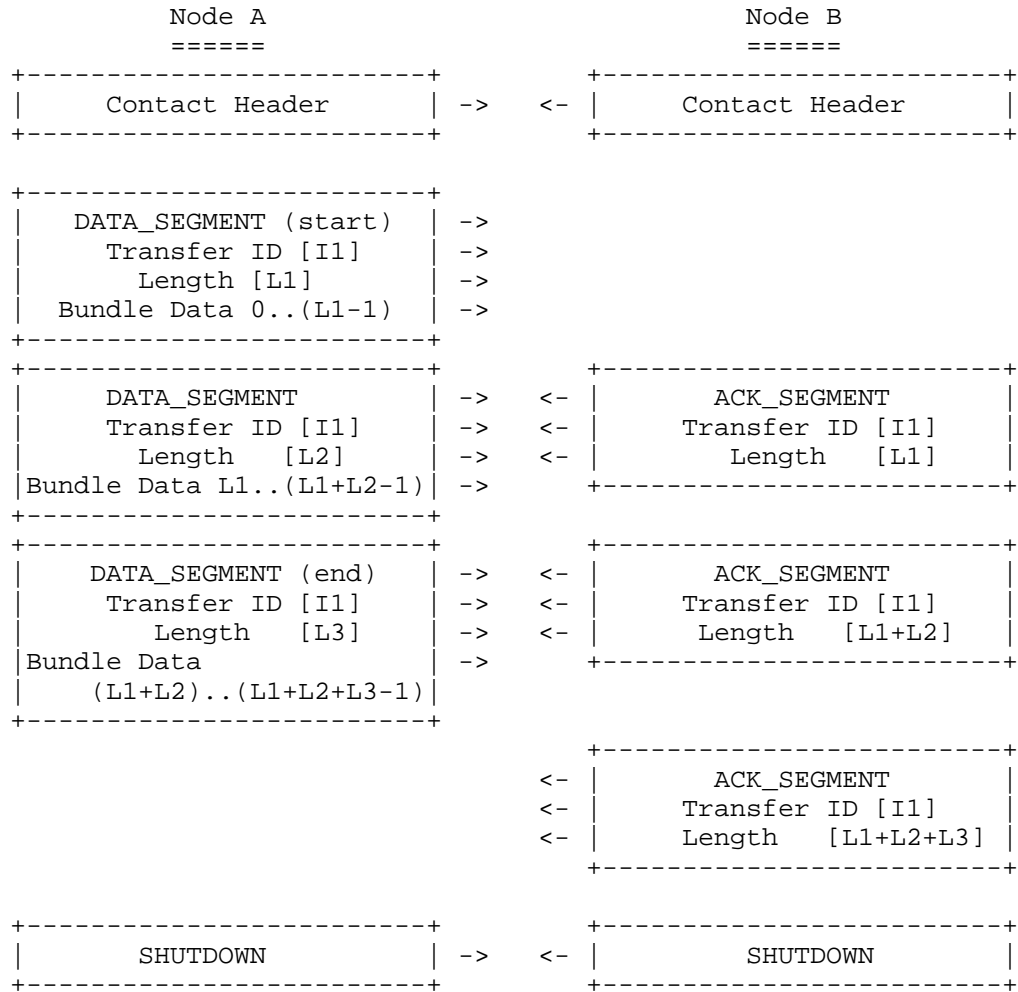


Figure 2: A Simple Visual Example of the Flow of Protocol Messages on a Single TCP Session between Two Nodes (A and B)

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. Port number 4556 has been assigned by IANA as the well-known port number for the TCP convergence layer. Other port numbers MAY be used per local configuration. Determining a peer's port number (if different from the well-known TCPCL port) is up to the implementation.

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:

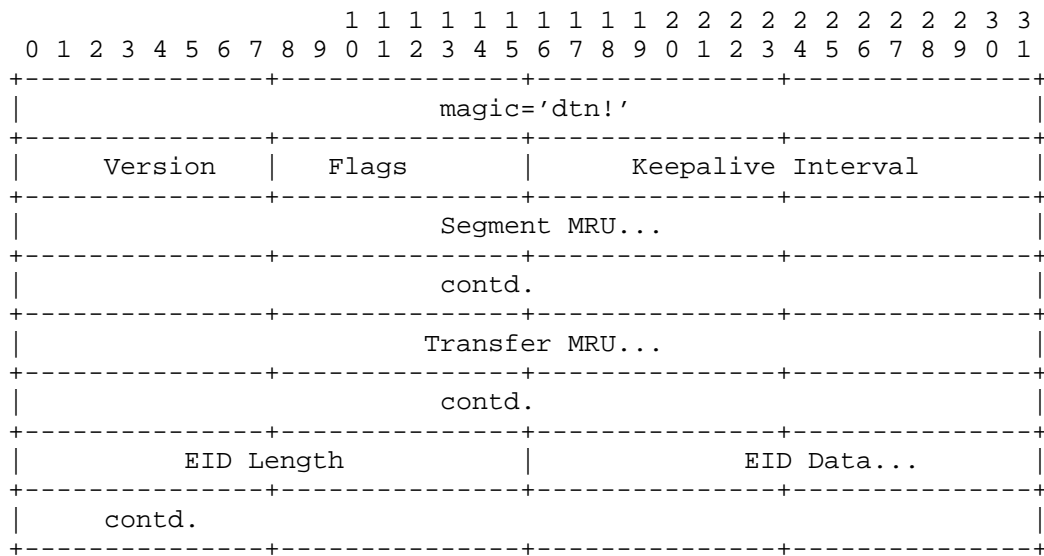


Figure 3: Contact Header Format

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 DATA_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 is a special case which indicates the lack of EID rather than a truly empty EID. A non-zero-length EID Data contains 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>.

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

Table 1: Contact Header 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.

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.

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 mechanisms 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
+---+---+---+---+---+
| type | flags |
+---+---+---+---+---+
```

Figure 4: Format of the One-Octet Message Header

type: Indicates the type of the message as per Table 2 below.

flags: Optional flags defined based on message type.

The types and values for the message type code are as follows.

Type	Code	Description
DATA_SEGMENT	0x1	Indicates the transmission of a segment of bundle data, as described in Section 5.4.3.
ACK_SEGMENT	0x2	Acknowledges reception of a data segment, as described in Section 5.4.4.
REFUSE_BUNDLE	0x3	Indicates that the transmission of the current bundle SHALL be stopped, as described in Section 5.4.5.
KEEPALIVE	0x4	KEEPALIVE message for the session, as described in Section 5.2.1.
SHUTDOWN	0x5	Indicates that one of the nodes participating in the session wishes to cleanly terminate the session, as described in Section 6.
LENGTH	0x6	Contains the length (in octets) of the next bundle, as described in Section 5.4.2.

Table 2: 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 2) 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 Table 2) 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 (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 or LENGTH message received after feature negotiation has disabled those features), there is a protocol-level message to signal this condition in the form of a REJECT reply.

The format of a REJECT message follows:

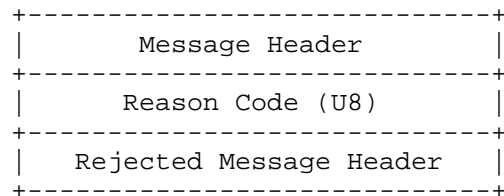


Figure 5: Format of REJECT Messages

The Rejected Message Header is a copy of the Message Header to which the REJECT message is sent as a response. The REJECT Reason Code is an 8-bit unsigned integer and indicates why the REJECT itself was sent. The specified values of the reason code are:

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 3: 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.

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". Unless the TLS parameters change between two sequential handshakes, the subsequent handshake is likely to fail just as the earlier one.

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 below where the client/requester role is represented by the prefix "C" and the server/responder role is represented by the prefix "S". Unordered or "simultaneous" actions are shown as "C/S".

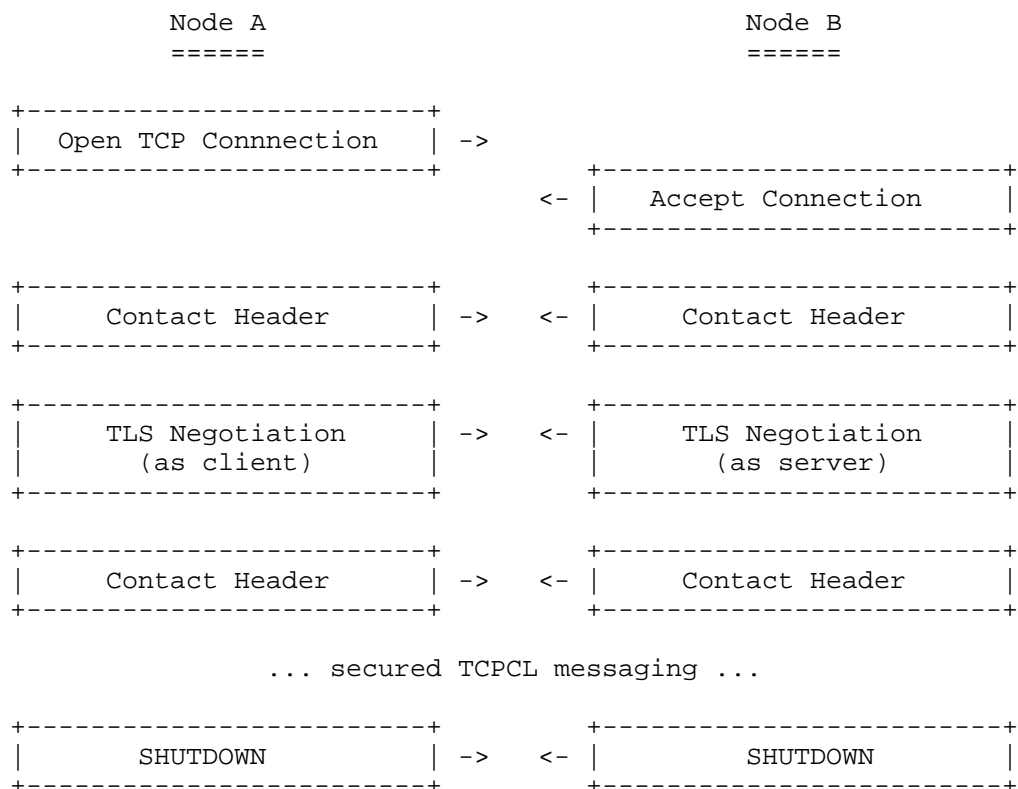


Figure 6: A simple visual example of TCPCL TLS Establishment between two nodes

5.4. Bundle Transfer

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

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. The Transfer ID provides a similar behaivior to a datagram sequence number. A Transfer ID does not attempt to address uniqueness of the bundle data itself and has no relation to concepts such as bundle fragmentation. Transmitting the same bundle repeatedly, or fragments of the same bundle, or any other combination will result in a unique Transfer ID for each transmission sequence.

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. Bundle Length (LENGTH)

The LENGTH message contains the total length, in octets, of the next bundle, formatted as a 64-bit unsigned integer. Its purpose 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. The Total Bundle Length field within a LENGTH message SHALL be used as informative data by the receiver. If, for whatever reason, the actual total legnth of bundle data received differs from the value indicated by the LENGTH message, the receiver SHOULD accept the full set of bundle data as valid.

The format of the LENGTH message is as follows:

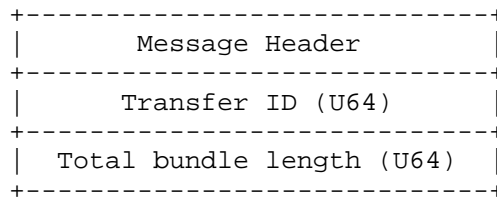


Figure 7: Format of LENGTH Messages

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

A receiver MAY send a BUNDLE_REFUSE message as soon as it receives a LENGTH message without waiting for the next DATA_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 LENGTH message not immediately before the start of a starting DATA_SEGMENT the receiver SHALL send a REJECT message with a Reason Code of "Message Unexpected".

5.4.3. Bundle Data Transmission (DATA_SEGMENT)

Each bundle is transmitted in one or more data segments. The format of a DATA_SEGMENT message follows in Figure 8 and its use of header flags is shown in Figure 9.

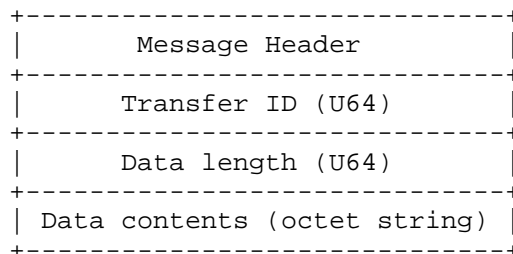


Figure 8: Format of DATA_SEGMENT Messages

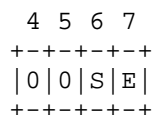


Figure 9: Format of DATA_SEGMENT Header flags

The type portion of the message header contains the value 0x1.

The flags portion of the message header octet contains two optional values in the two low-order bits, denoted 'S' and 'E' above. The 'S' bit MUST be set to one if it precedes the transmission of the first segment of a new bundle. The 'E' bit MUST be set to one when transmitting the last segment of a bundle. In the case where an entire transfer is accomplished in a single segment, both the 'S' and 'E' bits MUST be set to one.

Following the message header, the length field is a 64-bit unsigned integer containing the number of octets of bundle data that are transmitted in this segment. Following this length is the actual data contents.

Once a transmission of a bundle has commenced, the node MUST only send segments containing sequential portions of that bundle until it sends a segment with the 'E' bit set.

5.4.4. Bundle Acknowledgments (ACK_SEGMENT)

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 ACK_SEGMENT message follows in Figure 10 and its use of header flags is the same as for DATA_SEGMENT (shown in Figure 9). The flags of an ACK_SEGMENT message SHALL be identical to the flags of the DATA_SEGMENT message for which it is a reply.

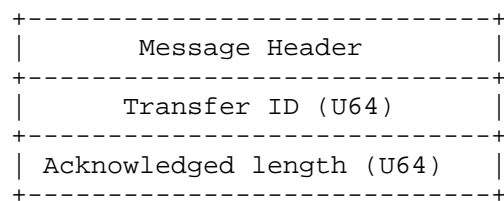


Figure 10: Format of ACK_SEGMENT Messages

To transmit an acknowledgment, a node first transmits a message header with the ACK_SEGMENT type code and all flags set to zero, then

transmits a 64-bit unsigned integer containing the cumulative length in octets of the received segment(s) of the current bundle. The length **MUST** fall on a segment boundary. That is, only full segments can be acknowledged.

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. Bundle Refusal (REFUSE_BUNDLE)

As bundles can be large, the TCPCL supports an optional mechanisms 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 LENGTH or DATA_SEGMENT message, the node **MAY** transmit a REFUSE_BUNDLE 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.

The format of the message is as follows:

```

+-----+
|           Message Header           |
+-----+
|           Transfer ID (U64)        |
+-----+
```

Figure 11: Format of REFUSE_BUNDLE Messages

```

      4 5 6 7
+---+---+
| RCode |
+---+---+
```

Figure 12: Format of REFUSE_BUNDLE Header flags

The RCode field, which stands for "reason code", contains a value indicating why the bundle was refused. The following table contains semantics for some values. Other values **MAY** be registered with IANA, as defined in Section 8.

Name	RCode	Semantics
Unknown	0x0	Reason for refusal is unknown or not specified.
Completed	0x1	The receiver now has the complete bundle. The sender MAY now consider the bundle as completely received.
No Resources	0x2	The receiver's resources are exhausted. The sender SHOULD apply reactive bundle fragmentation before retrying.
Retransmit	0x3	The receiver has encountered a problem that requires the bundle to be retransmitted in its entirety.

Table 4: REFUSE_BUNDLE Reason Codes

The receiver MUST, for each bundle preceding the one to be refused, have either acknowledged all DATA_SEGMENTS or refused the bundle. This allows the sender to identify the bundles accepted and refused by means of a simple FIFO list of segments and acknowledgments.

The bundle refusal MAY be sent before the entire data segment is received. If a sender receives a REFUSE_BUNDLE message, the sender MUST complete the transmission of any partially sent DATA_SEGMENT message (so that the receiver stays in sync). 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 DATA_SEGMENT for the same bundle after transmitting a REFUSE_BUNDLE message since messages MAY cross on the wire; if this happens, subsequent segments of the bundle SHOULD also be refused with a REFUSE_BUNDLE message.

Note: If a bundle transmission is aborted in this way, the receiver MAY not receive a segment with the 'E' flag set to '1' for the aborted bundle. The beginning of the next bundle is identified by the 'S' bit set to '1', indicating the start of a new bundle.

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 node SHOULD acknowledge all received data segments before sending a SHUTDOWN message to end the session.

The format of the SHUTDOWN message is as follows:

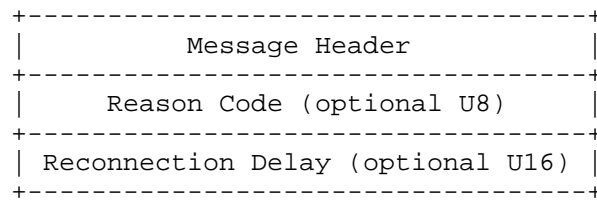


Figure 13: Format of SHUTDOWN Messages

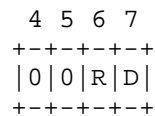


Figure 14: Format of SHUTDOWN Header flags

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 header flags and transmit a one-octet reason code immediately following the message header. The specified values of the reason code are:

Name	Code	Description
Idle timeout	0x00	The session is being closed due to idleness.
Version mismatch	0x01	The node cannot conform to the specified TCPCL protocol version.
Busy	0x02	The node is too busy to handle the current session.
Contact Failure	0x03	The node cannot interpret or negotiate contact header option.
TLS failure	0x04	The node failed to negotiate TLS session and cannot continue the session.
Resource Exhaustion	0x05	The node has run into some resource limit and cannot continue the session.

Table 5: 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 header 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 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, then the node MUST NOT transmit the SHUTDOWN message but still SHOULD close the TCP connection. In particular, if the session is to be closed (for whatever reason) while a node is in the process of transmitting a bundle data segment, the receiving node is still expecting segment data and might erroneously interpret the SHUTDOWN message to be part of the data segment.

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 4). 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 session 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 MAY be able to obtain bundles that it SHOULD not have. Therefore, a node SHALL NOT use the endpoint identifier conveyed in the TCPCL session message to derive a peer node's identity unless it can corroborate it via other means.

These concerns MAY be mitigated through the use of the Bundle Security Protocol [RFC6257]. In particular, the Bundle Authentication Block defines mechanism for secure exchange of bundles between DTN nodes. Thus, an implementation could delay trusting the presented endpoint identifier until the node can securely validate that its peer is in fact the only member of the given singleton endpoint.

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 REFUSE_BUNDLE 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]

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.

Parameter	Value
Service Name:	dtn-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. Message Types

IANA has created, under the "Bundle Protocol" registry, a sub-registry titled "Bundle Protocol TCP Convergence-Layer Message Types" and initialized it with the contents below. The registration procedure is RFC Required.

Code	Message Type
0x0	Reserved
0x1	DATA_SEGMENT
0x2	ACK_SEGMENT
0x3	REFUSE_BUNDLE
0x4	KEEPALIVE
0x5	SHUTDOWN
0x6	LENGTH
0x7	REJECT
0x8	STARTTLS
0x9--0xf	Unassigned

Message Type Codes

8.4. REFUSE_BUNDLE Reason Codes

IANA has created, under the "Bundle Protocol" registry, a sub-registry titled "Bundle Protocol TCP Convergence-Layer REFUSE_BUNDLE Reason Codes" and initialized it with the contents of Table 3. 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

REFUSE_BUNDLE Reason Codes

8.5. SHUTDOWN Reason Codes

IANA has created, under the "Bundle Protocol" registry, a sub-registry titled "Bundle Protocol TCP Convergence-Layer SHUTDOWN Reason Codes" and initialized it with the contents of Table 4. 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

SHUTDOWN Reason Codes

8.6. 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 REJECT Reason Codes" and initialized it with the contents of Table 4. The registration procedure is RFC Required.

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

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>>.

[I-D.ietf-dtn-bpbis]
Burleigh, S., Fall, K., and E. Birrane, "Bundle Protocol",
draft-ietf-dtn-bpbis-05 (work in progress), September
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>>.
- [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-bpsec]
Birrane, E. and K. McKeever, "Bundle Protocol Security
Specification", draft-ietf-dtn-bpsec-02 (work in
progress), July 2016.

Appendix A. Significant changes from RFC7242

The areas in which changes from [RFC7242] have been made to existing
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 a bundle transfer identification number to all bundle-related messages (LENGTH, DATA_SEGMENT, ACK_SEGMENT, REFUSE_BUNDLE).
- o Use flags in ACK_SEGMENT to mirror flags from DATA_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 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
1229 19th Street NW
Washington, DC 20036
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