Delay-Tolerant Networking
Internet-Draft
Intended status: Standards Track
Expires: January 3, 2019

E. Birrane
Johns Hopkins Applied Physics Laboratory
July 2, 2018

Asynchronous Management Protocol
draft-birrane-dtn-amp-05

Abstract

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

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

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

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

2. Requirements Language

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

3. Scope

3.1. Protocol Scope

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

It is assumed that the protocols used to carry AMP messages provide addressing, confidentiality, integrity, security, fragmentation/

reassembly, and other network functions. Therefore, these items are outside of the scope of this document.

3.2. Specification Scope

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

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

4. Terminology

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

5. Constraints and Assumptions

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

- o Intelligent Push of Information - Nodes in a challenged network cannot guarantee concurrent, bi-directional communications. Some links between nodes may be strictly unidirectional. AMP Agents "push" data to Managers rather than Managers "pulling" data from Agents.

- o Small Message Sizes - Smaller messages require smaller periods of viable transmission for communication, incur less retransmission cost, and consume fewer resources when persistently stored en route in the network. AMP minimizes message size wherever practical, to include binary data representations and predefined data definitions and templates.

- o Absolute and Custom Data Identification - All data in the system must be uniquely addressable, to include operator-specified information. AMP provides a compact encoding for identifiers.

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

6. Technical Notes

- Unless otherwise specified, multi-byte values in this specification are expected to be transmitted in network byte order (Big Endian).

- Character encodings for all text-based data types will use UTF-8 encodings.

- All AMP encodings are self-terminating. This means that, given an indefinite-length octet stream, each encoding can be unambiguously decoded from the stream without requiring additional information such as a length field separate from the data type definition.

- 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.

- In order to describe the encoding of data models specified in [I-D.birrane-dtn-adm], this specification must refer to both the data object being encoded and to the encoding of that data object. When discussing the encoded version of a data object, this specification uses the notation "E(data_object)" where E() refers to a conceptual encoding function. This notation is only provided as a means of clarifying the text and imposes no changes to the actual wire coding. For example, this specification will refer to the "macro" data object as "Macro" and to the encoding of a Macro as "E(Macro)".

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

Therefore, when transmitted on the wire, Field 1 will be received first, followed by Field 2 (if present).

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

Figure 1: Byte Field Formatting Example

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

7. AMP-Specific Concepts

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

7.1. Nicknames (NN)

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

7.1.1. Motivation for Compression

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

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

In particular, as specified in [I-D.birrane-dtn-adm], ARIs can be grouped by (1) their namespace and (2) the type of AMM object being

identified. For example, consider the following ARIs of type EDD defined in ADM1 with a Moderated Namespace of "/DTN/ADM1/".

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

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

7.1.2. ADM Enumeration

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

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

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

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

7.1.3. ADM Object Type Enumeration

An ADM Object Type Enumeration is an unsigned integer in the range of 0 - 19. This covers all of the standard areas for the ADM Template as defined in [I-D.birrane-dtn-adm]. Each of these types are enumerated in Table 1.

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

Table 1: ADM Type Enumerations

7.1.4. Nickname Definition

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

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

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

7.1.5. ADM Enumeration Considerations

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

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

Table 2: ADM Enumerations Labels

8. Encodings

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

8.1. CBOR Considerations

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

- All AMP encodings are of definite length and, therefore, indefinite encodings MUST NOT be used.

- AMP encodings MUST NOT use CBOR tags. Identification mechanisms in the AMP capture structure and other information such that tags are not necessary.

- Canonical CBOR MUST be used for all encoding. All AMP CBOR decoders MUST run in strict mode.

- Encodings MUST result in smallest data representations. There are several cases where the AMM defines types with less granularity than CBOR. For example, AMM defines the UINT type to represent unsigned integers up to 32 bits in length. CBOR supports separate definitions of unsigned integers of 8, 16, or 32 bits in length. In cases where an AMM type MAY be encoded in multiple ways in CBOR, the smallest data representation MUST be used. For example, UINT values of 0-255 MUST be encoded as a uint8_t, and so on.

8.2. AMM Type Encodings

8.2.1. Primitive Types

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

AMM Type	CBOR Major Type	Comments
BYTE	unsigned int or byte string	BYTEs are individually encoded as unsigned integers unless the are defined as part of a byte string, in which case they are encoded as a single byte in the byte string.
INT	unsigned integer or negative integer	INTs are encoded as positive or negative integers from (u)int8_t up to (u)int32_t.
UINT	unsigned integer	UINTs are unsigned integers from uint8_t up to uint32_t.
VAST	unsigned integer or negative integer	VASTs are encoding as positive or negative integers up to (u)int64_t.
UVAST	unsigned integer	VASTs are unsigned integers up to uint64_t.
REAL32	floating point	Up to an IEEE-754 Single Precision Float.
REAL64	floating point	Up to an IEEE-754 Double Precision Float.
STRING	text string	Uses CBOR encoding unmodified.
BOOL	Simple Value	0 is considered FALSE. 1 is considered TRUE.

Table 3: Standard Numeric Types

8.2.2. Derived Types

This section provides the CBOR encodings for AMM derived types.

8.2.2.1. Byte String Encoding

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

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

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

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

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

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

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

```
                               Potential Time values
                 ________________________/\________________________
                /                                                  \
                      Relative Times            Absolute Times
                <------------------------><------------------------>
                     0 - 558,230,400         558,230,401 - 2^64

                |------------------------|-------------------------|
                |                        |
       00:00:00 1 Jan 2000      00:00:00 9 Sep 2017
       Unix Time 946684800      Unix Time 1504915200
```

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

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

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

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

8.2.2.3. Type-Name-Value (TNV)

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

```
                       +---------+
                       |   TNV   |
                       | [ARRAY] |
                       +----++---+
                            ||
                            ||
           _______________/  \________________
          /                                   \
          +------------+-----------+----------+
          | Flags/Type |    Name   |   Value  |
          |   [UINT]   | [TXT STR] | [Varies] |
          |            |   (opt)   |   (opt)  |
          +------------+-----------+----------+
```

Figure 2: E(TNV) Format

The E(TNV) fields are defined as follows.

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

```
                   E(TNV) Flag/Type Byte Format

                    +------+---------------+
                    | Name |    Struct     |
                    | Flag |     Type      |
                    +------+---------------+
                    |  7   | 6 5 4 3 2 1 0 |
                    +------+---------------+
                    MSB                    LSB
```

Figure 3

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

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

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

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

8.2.3. Collections

8.2.3.1. Type-Name-Value Collection (TNVC)

A TNV Collection (TNVC) is a series of multiple TNV values. This is encoded as a CBOR array with each element in the array represented by the encoding of a TNV in accordance with Section 8.2.2.3.

8.2.3.2. Type-Then-Value Collection (TTVC)

A Types-Then-Value Collection (TTVC) provides a mechanism for communicating a typed set of values by separating the types from the values themselves. This construction is useful both for rapidly performing type verification and for efficiently omitted type information where appropriate.

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

In certain circumstances, a set of values can be communicated without any type information when type information can be inferred from context. In these circumstances, separating types from values allows for an efficient way to omit type information when necessary.

The TTVC is encoded as a CBOR array with either one or two elements. If the array has 1 element, then it MUST be a CBOR array of values. If the TTVC array has 2 elements, then it MUST contain a CBOR byte string of type information followed by a CBOR array of values. In cases where both types and values are present, the number of types MUST be the same as the number of elements as the array of values. The order of types MUST correspond to the order of values in the second array.

The E(TTVC) format is illustrated in Figure 4

```
                           E(TTVC) Format

                            +---------+
                            |   TTVC  |
                            | [ARRAY] |
                            +----++---+
                                 ||
                                 ||
                       ________/  \_________
                      /                     \
                      +-----------+---------+
                      |  Types    |  Values |
                      | [BYTESTR] | [ARRAY] |
                      |  (opt)    |         |
                      +-----------+---------+
```

Figure 4

These fields are defined as follows.

Types
: The Types field is encoded as a CBOR byte string with each element of the array encoded as a byte. Each byte represents a type and MUST match a type enumeration as defined in [I-D.birrane-dtn-adm].

Values
: The Values array is encoded as a CBOR array with each element of the array encoded as a CBOR byte string containing the CBOR encoding of the value, based on its type as required by this specification.

8.2.3.3. ARI Collections (AC)

An ARI collection is an ordered collection of ARI values. It is encoded as a CBOR array with each element being an encoded ARI, as illustrated in Figure 5.

```
                          E(AC) Format

                          +---------+
                          |    AC   |
                          | [ARRAY] |
                          +----++---+
                               ||
                               ||
                      ________/  \_________
                     /                     \
                     +-------+     +-------+
                     | ARI 1 | ... | ARI N |
                     | [ARI] |     | [ARI] |
                     +-------+     +-------+
```

Figure 5

8.2.3.4. Expressions (EXPR)

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

The Expression object is encoded as a CBOR byte string whose format is illustrated in Figure 6.

```
                         E(EXPR) Format

                     +--------+-----------+
                     |  Type  | Expression |
                     | [BYTE] |    [AC]    |
                     +--------+-----------+
```

Figure 6

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

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

8.3. AMM Resource Identifier (ARI)

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

8.3.1. Encoding ARIs of Type LITERAL

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

The E(ARI) of a LIT object is encoded as a CBOR byte string and consists of a mandatory flag BYTE and the value of the LIT.

The E(ARI) structure for LIT types is illustrated in Figure 7.

```
                    E(ARI) Literal Format

                    +--------+----------+
                    | Flags  |  Value   |
                    | [BYTE] | [VARIES] |
                    +--------+----------+
```

Figure 7

These fields are defined as follows.

Flags
The Flags byte identifies the object as being of type LIT and also captures the primitive type of the following value. The layout of this byte is illustrated in Figure 8.

```
               E(ARI) Literal Flag Byte Format

                +------------+-------------+
                | VALUE TYPE | STRUCT TYPE |
                +------------+-------------|
                | 7  6  5  4 | 3  2  1  0  |
                +------------+-------------+
                 MSB                    LSB
```

Figure 8

Value Type
The high nibble of the flag byte describes the type of the value of the ARI being encoded. This type

MUST be one of the AMM data types defined in [I-D.birrane-dtn-adm] as a "Primitive Type".

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

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

8.3.2. Encoding Non-Literal ARIs

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

The E(ARI) is encoded as a CBOR byte string and consists of a mandatory flag byte, an encoded object name, and optional annotations to assist with filtering, access control, and parameterization. The E(ARI) structure is illustrated in Figure 9.

E(ARI) General Format

```
+--------+---------+-----------+---------+---------+-----------+
| Flags  |   NN    |   Name    |  Parms  |  Issuer |    Tag    |
| [BYTE] | [UVAST] | [BYTESTR] | [TTVC]  | [UVAST] | [BYTESTR] |
|        |  (opt)  |           |  (opt)  |  (opt)  |    opt)   |
+--------+---------+-----------+---------+---------+-----------+
```

Figure 9

These fields are defined as follows.

Flags
Flags describe the type of structure and which optional fields are present in the encoding. The layout of the flag byte is illustrated in Figure 10.

E(ARI) General Flag Byte Format

```
+----+------+-----+-----+-------------+
| NN | PARM | ISS | TAG | STRUCT TYPE |
+----+------+-----+-----+-------------+
| 7  |  6   |  5  |  4  |3  2  1  0   |
+----+------+-----+-----+-------------+
MSB                                  LSB
```

Figure 10

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

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

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

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

Structure Type (STRUCT TYPE)
The lower nibble of the E(ARI) flag byte identifies the kind of structure being identified. This field

MUST contain one of the AMM object types defined in [I-D.birrane-dtn-adm].

Nickname (NN)
: This optional field contains the Nickname as calculated according to Section 7.1.

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

Parameters
: The parameters field is represented as a Types Then Value Collection (TTVC) as defined in Section 8.2.3.2. The overall number of items in the collection represents the number of parameters. The types of the TTVC represent the types of each parameter, with the first listed type associated with the first parameter, and so on. The values, if present, represent the values of the parameters, with the first listed value being the value of the first parameter, and so on.

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

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

8.4. ADM Object Encodings

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

Note: The encoding of an object refers to the way in which the complete object can be encoded such that the object as it exists on a

Manager may be re-created on an Agent, and vice-versa. In cases where both a Manager and an Agent already have the definition of an object, then only the encoded ARI of the object needs to be communicated. This is the case for all objects defined in a published ADM and any user-defined object that has been synchronized between an Agent and Manager.

8.4.1. Externally Defined Data (EDD)

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

The representation of these data is simply their identifying ARIs. The representation of an EDD is illustrated in Figure 11.

```
                 E(EDD) Format

                   +-------+
                   |  ID   |
                   | [ARI] |
                   +-------+
```

Figure 11

ID
: This is the ARI identifying the EDD. Since EDDs are always defined solely in the context of an ADM, this ARI MUST NOT have an ISSUER field and MUST NOT have a TAG field. This ARI may be defined with parameters.

8.4.2. Constants (CONST)

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

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

The format of a Constant is illustrated in Figure 12.

```
                         E(CONST) Format

                            +-------+
                            |  ID   |
                            | [ARI] |
                            +-------+

                            Figure 12
```

ID

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

8.4.3. Controls (CTBR)

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

The format of a Control is illustrated in Figure 13.

```
                         E(CTBR) Format

                            +-------+
                            |  ID   |
                            | [ARI] |
                            +-------+

                            Figure 13
```

ID

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

8.4.4. Macros (MAC)

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

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

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

The format of a Macro is illustrated in Figure 14.

```
                          E(MAC) Format

                     +-------+------------+
                     |  ID   | Definition |
                     | [ARI] |    [AC]    |
                     +-------+------------+
```

Figure 14

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

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

8.4.5. Operators (OPER)

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

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

Operators are always evaluated in the context of an Expression. The encoding of an Operator is illustrated in Figure 15.

```
                            E(OP) Format

                              +-------+
                              |  ID   |
                              | [ARI] |
                              +-------+
```

Figure 15

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

8.4.6. Report Templates (RPTT)

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

A Report Template is modeled as an AC, as each data definition in the template is identified by an ARI.

```
                           E(RPTT) Format

                         +-----------------+
                         | Report Contents |
                         |      [AC]       |
                         +-----------------+
```

Figure 16

8.4.7. Report (RPT)

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

A Report object is comprised of the identifier of the template used to populate the report, an optional timestamp, and the contents of the report. A Report is encoded as a CBOR array with either 2 or 3 elements. If the array has 2 elements then the optional Timestamp MUST NOT be in the Report encoding. If the array has 3 elements then the optional timestamp MUST be included in the Report encoding. The Report encoding is illustrated in Figure 17.

```
                        E(RPT) Format

                         +---------+
                         |   RPT   |
                         | [ARRAY] |
                         +---++----+
                             ||
                             ||
               _____________/  \_____________
              /                              \
             +----------+-----------+---------+
             | Template | Timestamp | Entries |
             |   [ARI]  |   [TS]    | [TTVC]  |
             |          |   (opt)   |         |
             +----------+-----------+---------+
```

Figure 17

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

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

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

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

8.4.8. State-Based Rules (SBR)

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

Examples of SBRs include:

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

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

An SBR object is encoded as a CBOR array with 5 elements as illustrated in Figure 18.

```
                          E(SBR) Format

                          +---------+
                          |   SBR   |
                          | [ARRAY] |
                          +---++----+
                              ||
                              ||
          ___________________/  \___________________
         /                                          \
         +-------+-------+--------+--------+--------+
         |  ID   | START |  COND  | COUNT  | ACTION |
         | [ARI] | [TV]  | [EXPR] | [UINT] |  [AC]  |
         |       |       |        |        |        |
         +-------+-------+--------+--------+--------+
```

Figure 18

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

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

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

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

ACTION
The collection of Controls and/or Macros to run as part of the action. This is encoded as an AC in accordance with Section 8.2.3.3 with the stipulation that every ARI in this collection MUST be of type CTRL or MAC.

8.4.9. Table Templates (TBLT)

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

The TBLT Object is encoded as a CBOR array that MUST contain either 2 or 3 elements. If the array is of size 2, then the column names array MUST NOT be present in the encoding. If the array is of size 3 then the column names array MUST be present in the encoding. The format of the TBLT Object Array is illustrated in Figure 19.

```
                      E(TBLT) Format

                        +---------+
                        |  TBLT   |
                        | [ARRAY] |
                        +---++----+
                            ||
                            ||
             ____________/  \_____________
            /                              \
            +-------+----------+---------+
            |  ID   |   Types  |  Names  |
            | [ARI] | [BYTESTR] | [ARRAY] |
            |       |          |  (opt)  |
            +-------+----------+---------+
```

Figure 19

The elements of the TBLT array are defined as follows.

ID

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

Types

The types field captures the data types of each column comprising the table template. Each type is represented by its enumeration as defined in [I-D.birrane-dtn-adm]. This field is encoded as a CBOR byte string with the length of the string equal to the number of columns in the template and each column type enumeration encoded as a byte. The Nth byte in the byte string represents the type of the Nth column.

Names

When present, column names are encoded as a CBOR array. This array MUST have the same number of elements as bytes in the Types byte string. Each element in the Names array is encoded as a CBOR text string and represents the string representation of the column name. The Nth text string in the array represents the name of the Nth column.

8.4.10. Tables (TBL)

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

A Table object is encoded as a CBOR array, with the first element of the array identifying the Table Template and each subsequent element identifying a row in the table. The format of the TBL Object Array is illustrated in Figure 20.

```
                       E(TBL) Format

                        +---------+
                        |   TBL   |
                        | [ARRAY] |
                        +---++----+
                            ||
                            ||
              ______________/  \_______________
             /                                  \
             +---------+--------+     +--------+
             | TBLT ID |  Row 1 |     |  Row N |
             |  [ARI]  | [TTVC] | ... | [TTVC] |
             +---------+--------+     +--------+
```

Figure 20

The TBL fields are defined as follows.

Template ID (TBLT ID)
This is the ARI of the table template describing the format of the table and is encoded in accordance with Section 8.3.

Row
Each row of the table is represented as a series of values with optional type information to aid in type checking table contents to column types. Each row is encoded as a TTVC and MAY include type information. AMP implementations should consider the impact of including type information for every row on the overall size of the encoded table.
Each TTVC representing a row MUST contain the same number of elements as there are columns in the referenced Table Template.

8.4.11. Time-Based Rules (TBR)

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

Examples of TBRs include:

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

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

The TBR object is encoded as a CBOR array with 5 elements as illustrated in Figure 21.

```
                               E(TBR) Format

                                +---------+
                                |   TBR   |
                                | [ARRAY] |
                                +---++----+
                                    ||
                                    ||
                ___________________/  \___________________
               /                                          \
               +-------+-------+--------+--------+--------+
               |  ID   | START | PERIOD | COUNT  | ACTION |
               | [ARI] | [TV]  | [UINT] | [UINT] |  [AC]  |
               +-------+-------+--------+--------+--------+
```

Figure 21

ID
: This is the ARI identifying the TBR and is encoded in accordance with Section 8.3. If this ARI contains parameters they MUST include a name in support of pass-by-name to each element of the Action AC.

START
: The time at which the TBR condition should start to be evaluated.

PERIOD
: The number of seconds to wait between running the action associated with the TBR.

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

ACTION
: The collection of Controls and/or Macros to run as part of the action. This is encoded as an ARI Collection in accordance with Section 8.2.3.3 with the stipulation that every ARI in this collection MUST represent either a Control or a Macro.

8.4.12. Variables (VAR)

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

Variable objects are encoded as a CBOR byte string whose format is illustrated in Figure 22.

```
                          E(VAR) Format

                          +------------+
                          |  Variable  |
                          | [BYTESTR]  |
                          +-----++-----+
                                ||
                                ||
                ______________/  \____________
               /                              \
               +-------+--------+-------------+
               |  ID   |  Type  | Initializer |
               | [ARI] | [BYTE] |   [EXPR]    |
               +-------+--------+-------------+
```

Figure 22

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

Type
: This field captures the data type of the Variable encoded as a BYTE. This data type MUST be one of the data types defined in [I-D.birrane-dtn-adm].
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. In cases where this is the case, numeric conversions MUST be handled in accordance with [I-D.birrane-dtn-adm].

Initializer
: The initial value of the Variable is calculated by an Expression encoded in accordance with Section 8.2.3.4.

9. Functional Specification

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

9.1. AMP Message Summary

The AMP message specification is limited to three basic communications:

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

Table 4: ADM Message Type Enumerations

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

9.2. Message Group Format

Individual messages within the AMP are combined into a single group for communication with another AMP Actor. Messages within a group MUST be received and applied as an atomic unit. The format of a message group is illustrated in Figure 23. These message groups are assumed communicated amongst Agents and Managers as the payloads of encapsulating protocols which should provide additional security and data integrity features as needed.

A message group is encoded as a CBOR array with at least 2 elements, the first being the time the group was created followed by 1 or more messages that comprise the group. The format of the message group is illustrated in Figure 23.

```
                        AMP Message Group Format

                             +---------------+
                             | Message Group |
                             |    [ARRAY]    |
                             +------++-------+
                                    ||
                 ___________________||___________________
                /                                         \
                +-----------+-----------+     +-----------+
                | Timestamp | Message 1 | ... | Message N |
                |    [TS]   | [BYTESTR] |     | [BYTESTR] |
                +-----------+-----------+     +-----------+
```

Figure 23

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

Message N
The Nth message in the group.

9.3. Message Format

Each message identified in the AMP specification adheres to a common message format, illustrated in Figure 24, consisting of a message header, a message body, and an optional trailer.

Each message in the AMP is encode as a CBOR byte string formatted in accordance with Figure 24.

```
                           AMP Message Format

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

Figure 24

Header
The message header BYTE is shown in Figure 25. The header identifies a message context and opcode as well as flags that

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

AMP Common Message Header

```
+----------+-----+------+-----+----------+
| Reserved | ACL | Nack | Ack |  Opcode  |
+----------+-----+------+-----+----------+
|  7   6   |  5  |   4  |  3  |  2  1  0 |
+----------+-----+------+-----+----------+
 MSB                                 LSB
```

Figure 25

Opcode
The opcode field identifies which AMP message is being represented.

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

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

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

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

Trailer

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

9.4. Register Agent

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

The body of this message is the name of the new agent, encoded as illustrated in Figure 26.

```
                  Register Agent Message Body

                         +-----------+
                         |  Agent ID |
                         | [BYTESTR] |
                         +-----------+
```

Figure 26

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

9.5. Report Set

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

The body of this message contains information on the recipient of the reports followed by one or more Reports. The body is encoded as illustrated in Figure 27.

```
                     Report Set Message Body

             +----------+--------+     +--------+
             | RX Names |  RPT 1 |     |  RPT N |
             |  [ARRAY] |  [RPT] | ... |  [RPT] |
             +----------+--------+     +--------+
```

Figure 27

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

RPT N
: The Nth Report encoded in accordance with Section 8.4.7.

9.6. Perform Control

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

The body of this message is the start time for the controls followed by the controls themselves, as illustrated in Figure 28.

```
              Perform Control Message Body

                  +-------+-----------+
                  | Start |  Controls |
                  |  [TV] |    [AC]   |
                  +-------+-----------+
```

Figure 28

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

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

10. IANA Considerations

A Nickname registry needs to be established.

11. Security Considerations

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

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

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

12. References

12.1. Informative References

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

12.2. Normative References

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

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

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

Appendix A. Acknowledgements

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

Author's Address

Edward J. Birrane
Johns Hopkins Applied Physics Laboratory

Email: Edward.Birrane@jhuapl.edu