JOHNS HOPKINS
APPLIED PHYSICS LABORATORY

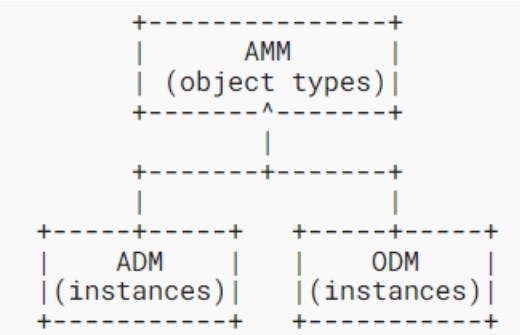
Background on Delay-Tolerant Networking Management Architecture (DTNMA)

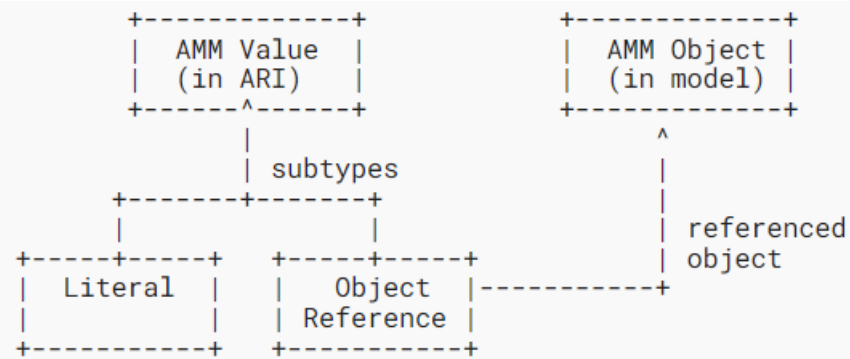
IETF 122 NETMOD WG

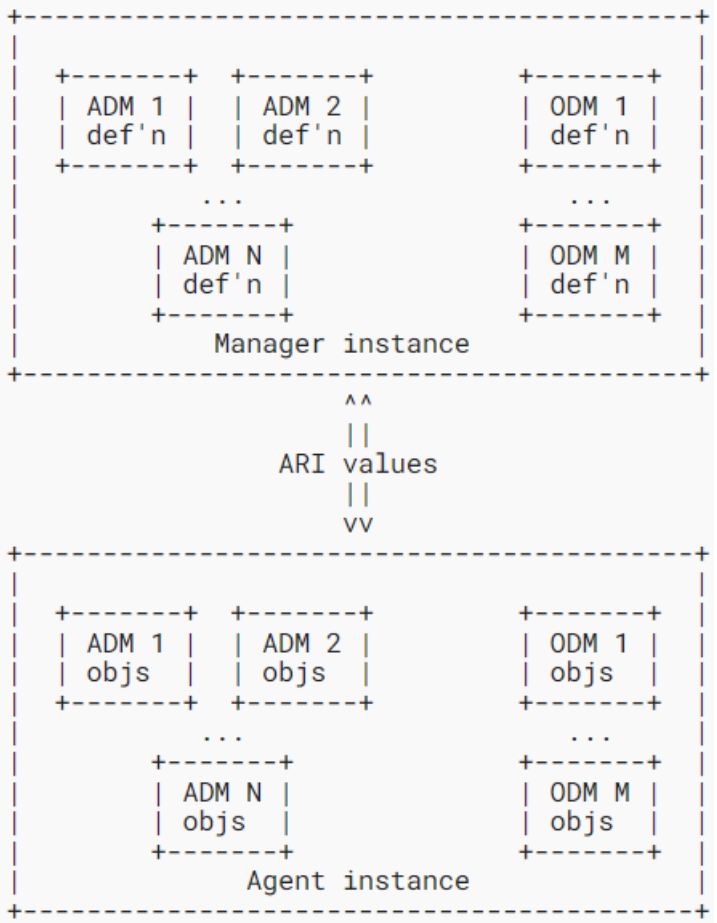
Brian Sipos
JHU/APL

DTNMA Overview

- The need for and scope of the DTNMA is defined in RFC 9675
  - Specifically Section 5 contrasts “current remote management approaches”
- The Application Management Model of DTNMA consists of:
  - A behavioral model based on object types with named/identified instances
  - A data model based on ARI values exchanged between Agent object instances and Manager entities
- Object instances reside within namespaces of the AMM
- Each namespace is either
  - A static application data model (ADM) or
  - A dynamic operational data model (ODM)



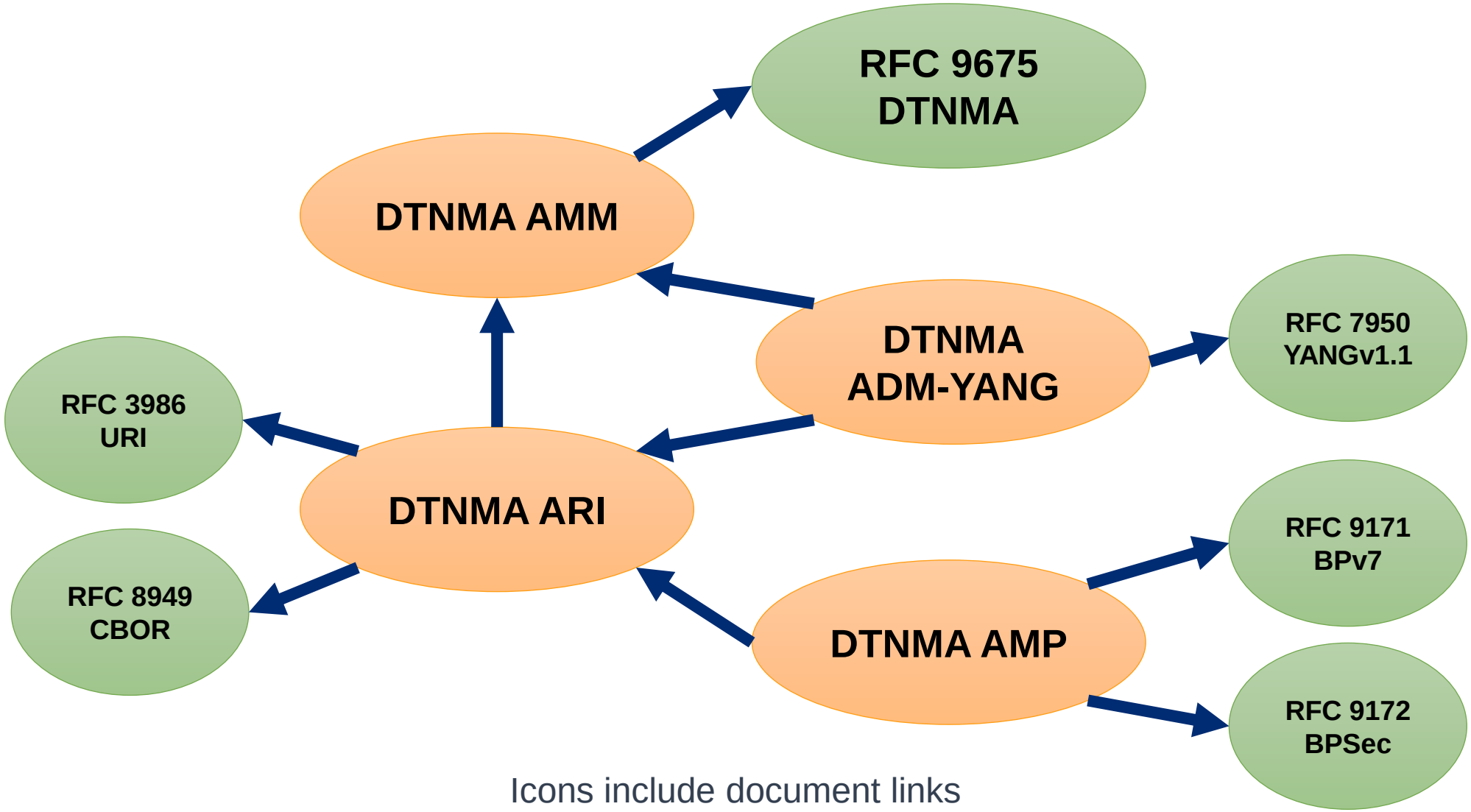




Current DTNMA Documents

- DTN Management Architecture (RFC 9675)
  - Informational explanation of the architecture and its concepts
- Application Management Model and Data Models (draft-ietf-dtn-amm-03)
  - Defines the structure and behavior of logical object and data models
  - Defines the requirements for an Agent—Manager protocol and its transport binding(s)
- Application Resource Identifier (draft-ietf-dtn-ari-04)
  - Defines the encoding syntax for AMM data values
  - Defines and initializes associated IANA code point registries in a new registry group
- Application Data Model YANG Syntax (draft-ietf-dtn-adm-yang-03)
  - Defines for ADMs all of the following in a cohesive way:
    file encoding, file naming, cross-reference control, feature control, documentation and lifecycle methods
  - Makes use of ARI text form for value encoding, and uses an ARI scheme namespace
- Asynchronous Management Protocol (draft-ietf-dtn-amp-01)
  - Simple binding of binary encoded ARI values onto BPv7 transport
  - Includes a version indication for protocol lifecycle management

DTNMA Normative References



Icons include document links

DTNMA Application Data Models

The logical structure of an ADM has been informed by best practices of:

- A fixed-depth hierarchy of identifiers (organization, model, object type, object)
- Unique identifiers at each level (text name and integer enumeration)
- Each ADM (namespace) is encoded as a “module” as a unit of definition
  - Module syntax needs to be both human-friendly and machine-readable
- Revision markings at module level
- Lifecycle status markings at module level and object instance level
- Internal documentation (descriptions and references) at module and object instance level
- Per-model “feature” breakdown to allow fine-grained implementation conformance
- Definition of each object instance in the ADM namespace
  - Each object instance has a type, name, and enumeration
  - Some object types can have parameters, operands, produced or result values

An Encoded Form for ADMs

- We desire an ADM encoded form based on the ADM logical structure and the goals:
  - Well-defined file naming
  - Human-friendly text encoding
  - Module cross-referencing mechanism
  - Syntax de-duplication mechanism
- The ADM encoding originally started as a JSON syntax document, but
  - JSON is made for machines, not humans, to read and edit
  - No standard way to enforce structure with tooling via a JSON “schema”
  - Aspects like file naming and cross-referencing would need to be invented
- YANG, as a module language and text encoding, already provides most of the capabilities needed by the logical ADM and desired in the goals above
  - YANG was not originally designed to separate its functional layers between its encoding, syntax, modular semantics, and data modeling keywords
  - Next slide gives a comparison and breakdown
  - YANG also has inbuilt support for extension keywords, themselves defined in YANG modules

Summary of Remote Management Sublayers

- Layering depths of existing remote management and modeling is varied
- Some examples from protocols identified in Section 5 of DTNMA are expanded below
  - The yellow box indicates the focus of the ADM needs

<table>
<tr><th>Aspect</th><th>SNMP</th><th>NETCONF</th><th>RESTCONF</th><th>CORECONF</th><th>DTNMA AMP</th><th>gNMI</th></tr>
<tr><td>Data transport</td><td>UDP (+)</td><td>SSH</td><td>HTTP</td><td>CoAP</td><td>BPv7</td><td>HTTPv2</td></tr>
<tr><td>Data encoding</td><td>ASN.1 BER</td><td>XML</td><td>XML / JSON</td><td>CBOR + SID</td><td>ARI</td><td>Protobuf enc.</td></tr>
<tr><td>Behavior model</td><td>SNMPv2</td><td>NETCONF</td><td>RESTCONF</td><td>CORECONF</td><td>AMM objects</td><td>gNMI</td></tr>
<tr><td>Data model</td><td rowspan="2">SMIv2</td><td colspan="3" rowspan="2">YANGv1 + NMDA</td><td>AMM typing</td><td>Protobuf DDL</td></tr>
<tr><td>Feature control</td><td rowspan="6">YANGv1</td><td>N/A</td></tr>
<tr><td>Module refs.</td><td rowspan="3">ASN.1 schema</td><td>Protobuf DDL</td></tr>
<tr><td>Revision keeping</td><td>N/A</td></tr>
<tr><td>Module encoding</td><td rowspan="2">Protobuf DDL</td></tr>
<tr><td>File Naming</td><td>Loosely defined</td></tr>
</table>

Summary of YANG Use and Side Effects

- The ADM-YANG draft is using YANG for its modular syntax
  - Each ADM *is encoded as a* YANG module, not YANG-as-a-schema for ADM *structure* in some other form
- All of the module encoding, naming, *etc.* is directly reused, as defined in Section 4
- Integration occurs using existing keyword extension mechanisms
  - These AMM keywords are used to define AMM object instances within ADMs
- This has precedence with `yang-data` in RFC 8040 and `structure` in RFC 8791
  - These use YANG as an XML infoset model for data not intended to be part of a *CONF datastore
- The use of ADM-YANG occurs separately from the existing YANG ecosystem
  - The namespace scheme for ADM-YANG is distinct from any used by *CONF models
  - The AMM keywords do not mix sub-statements or concepts from the YANG data model keywords
  - ADM modules will not import from non-ADM modules and vice-versa
  - ADM organization and module registration occurs in separate IANA registry group

Conclusions

- The application data models of the DTNMA need a concrete syntax
  - The syntax must be human friendly and machine readable
  - It must be modular, support long lifecycles, and make use of existing tooling where possible
- YANG provides a well-designed and well-supported modular syntax
  - Reuse is preferred over re-inventing “mundane” aspects of modules with lifecycles
- Other options have been considered but found to be lacking relative to YANG
- There is a need to proceed with ADM definitions and implementations
  - Part of this work is to divide definitions among cohesive and future-looking modules