Stream: Internet Engineering Task Force (IETF)
RFC: 9675
Category: Informational
Published: October 2024
ISSN: 2070-1721
Authors: E. Birrane, III *JHU/APL*; S. Heiner *JHU/APL*; E. Annis *JHU/APL*

RFC 9675
Delay-Tolerant Networking Management Architecture (DTNMA)

Abstract

The Delay-Tolerant Networking (DTN) architecture describes a type of challenged network in which communications may be significantly affected by long signal propagation delays, frequent link disruptions, or both. The unique characteristics of this environment require a unique approach to network management that supports asynchronous transport, autonomous local control, and a small footprint (in both resources and dependencies) so as to deploy on constrained devices.

This document describes a DTN Management Architecture (DTNMA) suitable for managing devices in any challenged environment but, in particular, those communicating using the DTN Bundle Protocol (BP). Operating over BP requires an architecture that neither presumes synchronized transport behavior nor relies on query-response mechanisms. Implementations compliant with this DTNMA should expect to successfully operate in extremely challenging conditions, such as over unidirectional links and other places where BP is the preferred transport.

Status of This Memo

This document is not an Internet Standards Track specification; it is published for informational purposes.

This document is a product of the Internet Engineering Task Force (IETF). It represents the consensus of the IETF community. It has received public review and has been approved for publication by the Internet Engineering Steering Group (IESG). Not all documents approved by the IESG are candidates for any level of Internet Standard; see Section 2 of RFC 7841.

Information about the current status of this document, any errata, and how to provide feedback on it may be obtained at https://www.rfc-editor.org/info/rfc9675.

Copyright Notice

Table of Contents

1. Introduction

This document describes a logical, informational Delay-Tolerant Networking Management Architecture (DTNMA) suitable for operating devices in a challenged architecture, such as those communicating using the DTN Bundle Protocol version 7 (BPv7) [RFC9171].

Challenged networks have certain properties that differentiate them from other kinds of networks. These properties, outlined in Section 2.2.1 of [RFC7228], include lacking end-to-end IP connectivity, having "serious interruptions" to end-to-end connectivity, and exhibiting delays longer than can be tolerated by end-to-end synchronization mechanisms (such as TCP).

These challenged network properties can be caused by a variety of factors such as physical constraints (e.g., long signal propagation delays and frequent link disruptions), administrative policies (e.g., quality-of-service prioritization, service-level agreements, and traffic management and scheduling), and off-nominal behaviors (e.g., active attackers and misconfigurations). Since these challenges are not solely caused by sparseness, instances of challenged networks will persist even in increasingly connected environments.

The DTN architecture, described in [RFC4838], has been designed for data exchange in challenged networks. Just as the DTN architecture requires new capabilities for transport and transport security, special consideration is needed for the operation of devices in a challenged network.

1.1. Purpose

This document describes how challenged network properties affect the operation of devices in such networks. This description is presented as a logical architecture formed from a union of best practices for operating devices deployed in challenged environments.

One important practice captured in this document is the concept of self-operation. Self-operation involves operating a device without human interactivity, without system-in-the-loop synchronous functions, and without a synchronous underlying transport layer. This means that devices determine their own schedules for data reporting, determine their own operational configuration, and perform their own error discovery and mitigation.

1.2. Scope

This document includes the information necessary to document existing practices for operating devices in a challenged network in the context of a logical architecture. A logical architecture describes the logical operation of a system by identifying components of the system (such as in a reference model), the behaviors they enable, and use cases describing how those behaviors result in the desired operation of the system.

Logical architectures are not functional architectures. Therefore, any functional design for achieving desired behaviors is out of scope for this document. The set of architectural principles presented here is not meant to completely specify interfaces between components.

The selection of any particular transport or network layer is outside of the scope of this document. The DTNMA does not require the use of any specific protocol such as IP, BP, TCP, or UDP. In particular, the DTNMA design does not presume the use of BPv7, IPv4, or IPv6.

> NOTE: As BPv7 is the preferred transport for networks conforming to the DTN architecture, the DTNMA should be considered for any BPv7 network deployment. However, the DTNMA may also be used in other networks that have similar needs for this particular style of self-operation. For this reason, the DTNMA does not require the use of BPv7 to transport management information.

Network features such as naming, addressing, routing, and communications security are out of scope for the DTNMA. It is presumed that any operational network communicating DTNMA messages would implement these services for any payloads carried by that network.

The interactions between and amongst the DTNMA and other management approaches are outside of the scope of this document.

1.3. Organization

The following nine sections provide details regarding the DTNMA.

Terminology: Section 2 identifies terms fundamental to understanding DTNMA concepts. Whenever possible, these terms align in both word selection and meaning with their use in other management protocols.

Challenged Network Overview: Section 3 describes important aspects of challenged networks and necessary approaches for their management.

Desirable Design Properties: Section 4 defines those properties of the DTNMA needed to operate within the constraints of a challenged network. These properties are similar to the specification of system-level requirements of a DTN management solution.

Current Remote Management Approaches: Section 5 provides a brief overview of existing remote management approaches. Where possible, the DTNMA adopts concepts from these approaches.

Motivation for New Features: Section 6 provides an overall motivation for this work. It also explains why a management architecture for challenged networks is useful and necessary.

Reference Model: Section 7 defines a reference model that can be used to analyze the DTNMA independently of an implementation or implementation architecture. This model identifies the logical components of the system and the high-level relationships and behaviors amongst those components.

Desired Services: Section 8 identifies and defines the DTNMA services provided to network and mission operators.

Logical Autonomy Model: Section 9 provides an example data model that can be used to analyze DTNMA control and data flows. This model is based on the DTNMA reference model.

Use Cases: Section 10 presents multiple use cases accommodated by the DTNMA. Each use case is presented as a set of control and data flows referencing the DTNMA reference model and logical autonomy model.

2. Terminology

This section defines terminology that is either unique to the DTNMA or necessary for understanding the concepts defined in this specification.

Timely Data Exchange: The ability to communicate information between two (or more) entities within a required period of time. In some cases, a 1-second exchange may not be timely; in other cases, a 1-hour exchange may be timely.

Local Operation: The operation of a device by an application co-resident on that device. Local operators are applications running on a device, and there might be one or more of these applications working independently or as one to perform the local operations function. Absent error conditions, local operators are always expected to be available to the devices they manage.

Remote Operation: The operation of a device by an application running on a separate device. Remote operators communicate with operated devices over a network. Remote operators are not always expected to be available to the devices they operate.

DTN Management: The management, monitoring, and control of a device that does not depend on stateful connections, timely data exchange of management messages, or system-in-the-loop synchronous functions. DTN management is accomplished as a fusion of local operation and remote operation techniques; remote operators manage the configuration of local operators who provide monitoring and control of their co-resident devices.

DTNMA Agent (DA): A role associated with a managed device responsible for reporting performance data, accepting policy directives, performing autonomous local control, error handling, and data validation. DAs exchange information with DTNMA Managers (DMs) operating on the same device and/or on remote devices in the network. A DA is a type of local operator.

DTNMA Manager (DM): A role associated with a managing device responsible for configuring the behavior of, and eventually receiving information from, DAs. DMs interact with one or more DAs located on the same device and/or on remote devices in the network. A DM is a type of remote operator.

Controls: Procedures run by a DA to change the behavior, configuration, or state of an application or protocol managed by that DA. These include procedures to manage the DA itself, such as having the DA produce performance reports or applying new management policies.

Externally Defined Data (EDD): Typed information made available to a DA by its hosting device but not computed directly by the DA itself.

Data Report: A typed, ordered collection of data values gathered by one or more DAs and provided to one or more DMs. Reports comply with the format of a given data report schema.

Data Report Schema: A named, ordered collection of data elements that represent the schema of a data report.

Rule: Unit of autonomous specification that provides a stimulus-response relationship between time or state on a DA and the actions or operations to be run as a result of that time or state.

3. Challenged Network Overview

The DTNMA provides network management services able to operate in challenged network environments for which the DTN architecture was created. This section describes what is meant by the term "challenged network", the important properties of such a network, and observations on impacts to management approaches.

3.1. Challenged Network Constraints

Constrained networks are defined as networks where "some of the characteristics pretty much taken for granted with link layers in common use in the Internet at the time of writing are not attainable" [RFC7228]. This broad definition captures a variety of potential issues relating to physical, technical, and regulatory constraints on message transmission. Constrained networks typically include nodes that regularly reboot or are otherwise turned off for long periods of time, transmit at low or asynchronous bitrates, and/or have very limited computational resources.

Separately, a challenged network is defined as one that "has serious trouble maintaining what an application would today expect of the end-to-end IP model" [RFC7228]. Links in such networks may be impacted by attenuation, propagation delays, mobility, occultation, and other limitations imposed by energy and mass considerations. Therefore, systems relying on such links cannot guarantee timely end-to-end data exchange.

> NOTE: Because challenged networks might not provide services expected of the end-to-end IP model, devices in such networks might not implement networking stacks associated with the end-to-end IP model. This means that devices might not include support for certain transport protocols (TCP/QUIC/UDP), web protocols (HTTP), or internetworking protocols (IPv4/IPv6).

By these definitions, a "challenged" network is a special type of "constrained" network, where constraints prevent timely end-to-end data exchange. As such, "All challenged networks are constrained networks ... but not all constrained networks are challenged networks ... Delay-Tolerant Networking (DTN) has been designed to cope with challenged networks" [RFC7228].

Solutions that work in constrained networks might not be solutions that work in challenged networks. In particular, challenged networks exhibit the following properties that impact the way in which the function of network management is considered.

- Timely end-to-end data exchange cannot be guaranteed to exist at any given time between any two nodes.
- Latencies on the order of seconds, hours, or days must be tolerated.
- Managed devices cannot be guaranteed to always be powered so as to receive ad hoc management requests (even requests with artificially extended timeout values).
- Individual links may be unidirectional.
- Bidirectional links may have asymmetric data rates.

- The existence of external infrastructure, services, systems, or processes such as a Domain Name System (DNS) or a Certificate Authority (CA) cannot be guaranteed.

3.2. Topology and Service Implications

The set of constraints that might be present in a challenged network impacts both the topology of the network and the services active within that network.

Operational networks handle cases where nodes join and leave the network over time. These topology changes may or may not be planned, they may or may not represent errors, and they may or may not impact network services. Challenged networks differ from other networks not in the presence of topological change but in the likelihood that impacts to topology result in impacts to network services.

The difference between topology impacts and service impacts can be expressed in terms of connectivity. Topological connectivity usually refers to the existence of a path between an application message source and destination. Service connectivity, alternatively, refers to the existence of a path between a node and one or more services needed to process -- often just in time -- application messaging. Examples of service connectivity include access to infrastructure services such as a Domain Name System (DNS) or a CA.

In networks that might be partitioned most of the time, it is less likely that a node would concurrently access both an application endpoint and one or more network service endpoints. For this reason, network services in a challenged network should be designed to allow for asynchronous operation. Accommodating this use case often involves the use of local caching, pre-placing information, and not hard-coding message information at a source that might change when a message reaches its destination.

> NOTE: One example of rethinking services in a challenged network is the securing of BPv7 bundles. The Bundle Protocol Security (BPSec) [RFC9172] security extensions to BPv7 do not encode security destinations when applying security. Instead, BPSec requires nodes in a network to identify themselves as security verifiers or acceptors when receiving and processing secured messages.

3.2.1. Tiered Management

Network operations and management approaches need to adapt to the topology and service impacts encountered in challenged networks. In particular, the roles and responsibilities of "managers" and "agents" need to be different than what would be expected of unchallenged networks.

When connectivity to a manager cannot be guaranteed, agents will need to rely on locally available information and local autonomy to react to changes at the node. When an agent uses local autonomy for self-operation, it acts as a local operator serving as a proxy for an absent remote operator.

Therefore, the role of a "manager" must become one of a remote operator generating configurations and other essential updates for the local operator "agents" that are co-resident on a managed device.

This approach creates a two-tiered management architecture. The first tier is the management of the local operator configuration using any one of a variety of standard mechanisms, models, and protocols. The second tier is the operation of the local device through the local operator.

The DTNMA defines the DTNMA Manager (DM) as a remote operator application and the DTNMA Agent (DA) as an agent acting as a local operator application. In this model, which is illustrated in Figure 1, the DM and DA can be viewed as applications, with the DM producing new configurations and the DA receiving those configurations from an underlying management mechanism.

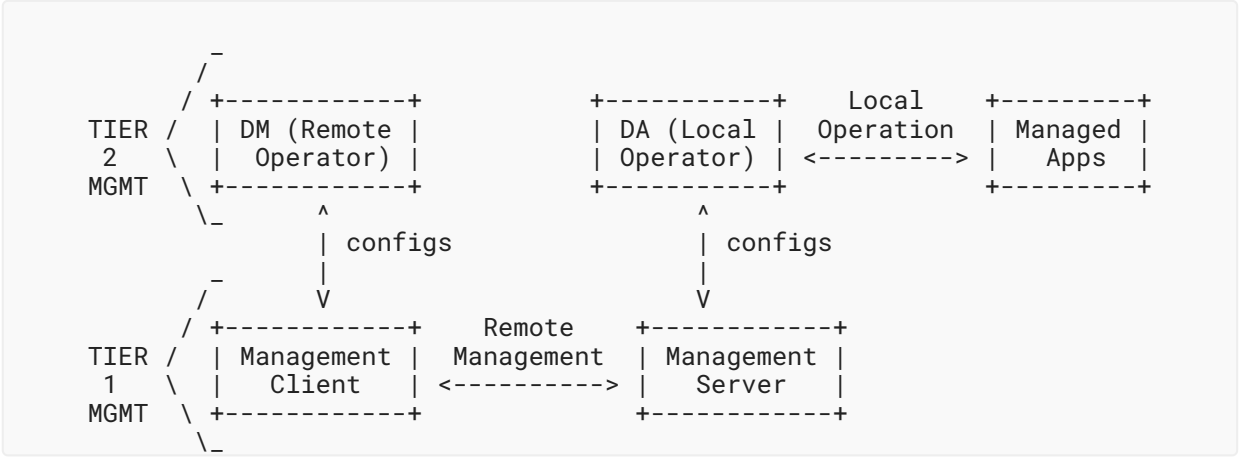

Figure 1: Two-Tiered Management Architecture

In this approach, the configurations produced by the DM are based on the DA features and associated data model. How those configurations are transported between the DM and the DA, and how results are communicated back from the DA to the DM, can be accomplished using whatever mechanism is most appropriate for the network and the device platforms -- for example, the use of a Network Configuration Protocol (NETCONF), RESTCONF, or Simple Network Management Protocol (SNMP) server on the managed device to provide configurations to a DA.

3.2.2. Remote and Local Manager Associations

In addition to disconnectivity, topological change can alter the associations amongst managed and managing devices. Different managing devices might be active in a network at different times or in different partitions. Managed devices might communicate with some, all, or none of these managing devices as a function of their own local configuration and policy.

> NOTE: These concepts relate to practices in conventional networks. For example, supporting multiple managing devices is similar to deploying multiple instances of a network service such as a DNS server or CA node. Selecting from a set of managing devices is similar to a sensor node's practice of electing cluster heads to act as privileged nodes for data storage and exfiltration.

Therefore, a network management architecture for challenged networks should:

1. Support a many-to-many association amongst managing and managed devices, and
2. Allow "control from" and "reporting to" managing devices to function independently of one another.

3.3. Management Special Cases

The following special cases illustrate some of the operational situations that can be encountered in the management of devices in a challenged network.

One-Way Management: A managed device can only be accessed via a unidirectional link or via a link whose duration is shorter than a single round-trip propagation time. Results of this management may come back at a different time, over a different path, and/or as observable from out-of-band changes to device behavior.

Summary Data: A managing device might only receive summary data regarding a managed device's state because a link or path is constrained by capacity or reliability.

Bulk Historical Reporting: A managing device receives a large volume of historical report data for a managed device. This can occur when a managed device rejoins a network or has temporary access to a high-capacity link (or path) between itself and the managing device.

Multiple Managers: A managed device tracks multiple managers in the network and communicates with them as a function of time, local state, or network topology. This scenario would also apply to challenged networks that interconnect two or more unchallenged networks such that managed and managing devices exist in different networks.

These special cases highlight the need for managed devices to operate without presupposing a dedicated connection to a single managing device. Managing devices in a challenged network might never expect a reply to a command, and communications from managed devices may be delivered much later than the events being reported.

4. Desirable Design Properties

This section describes those design properties that are desirable when defining a management architecture operating 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 DTNMA.

> NOTE: These properties may influence the design, construction, and adaptation of existing management tools for use in challenged networks. For example, the properties of the DTN architecture [RFC4838] resulted in the development of BPv7 [RFC9171] and BPSec [RFC9172]. Implementing the DTNMA model may result in the construction of new management data models, policy expressions, and/or protocols.

4.1. Dynamic Architectures

The DTNMA should be agnostic to the underlying physical topology, transport protocols, security solutions, and supporting infrastructure of a given network. Due to the likelihood of operating in a frequently partitioned environment, the topology of a network may change over time. Attempts to stabilize an architecture around individual nodes can result in a brittle management framework and the creation of congestion points during periods of connectivity.

The DTNMA should not prescribe any association between a DM and a DA other than those defined in this document. There should be no logical limitation on the number of DMs that can control a DA, the number of DMs that a DA should report to, or any requirement that a DM and DA relationship imply a pair.

> NOTE: Practical limitations on the relationships between and amongst DMs and DAs will exist as a function of the capabilities of networked devices. These limitations derive from processing and storage constraints, performance requirements, and other engineering factors. Implementors of managed and managing devices must account for these limitations in their device designs.

4.2. Hierarchically Modeled Information

The DTNMA should use data models to define the syntactic and semantic contracts for data exchange between a DA and a DM. A given model should have the ability to "inherit" the contents of other models to form hierarchical data relationships.

> NOTE: The term "data model" in this context refers to a schema that defines a contract between a DA and a DM regarding how information is represented and validated.

Many network management solutions use data models to specify the semantic and syntactic representation of data exchanged between managed and managing devices. The DTNMA is not different in this regard; information exchanged between DAs and DMs should conform to one or more predefined, normative data models.

A common best practice when defining a data model is to make it cohesive. A cohesive model is one that includes information related to a single purpose such as managing a single application or protocol. When applying this practice, it is not uncommon to develop a large number of small data models that, together, describe the information needed to manage a device.

Another best practice for data model development is the use of inclusion mechanisms to allow one data model to include information from another data model. This ability to include a data model avoids repeating information in different data models. When one data model includes information from another data model, there is an implied model hierarchy.

Data models in the DTNMA should allow for the construction of both cohesive models and hierarchically related models. These data models should be used to define all sources of information that can be retrieved, configured, or executed in the DTNMA. These actions would include supporting DA autonomy functions such as parameterization, filtering, and event-driven behaviors. These models will be used to both implement interoperable autonomy engines on DAs and define interoperable report parsing mechanisms on DMs.

> NOTE: While data model hierarchies can result in a more concise data model, arbitrarily complex nesting schemes can also result in very verbose encodings. Where possible, data identification schemes should be constructed that allow for both hierarchical data and highly compressible data identification.

4.3. Adaptive Push of Information

DAs in the DTNMA should determine when to push information to DMs as a function of their local state.

"Pull" management mechanisms require a managing device to send a query to a managed device and then wait for a response to that specific query. This practice implies some knowledge synchronization between entities (which may be as simple as knowing when a managed device might be powered). However, challenged networks cannot guarantee timely round-trip data exchange. For this reason, pull mechanisms should be avoided in the DTNMA.

"Push" mechanisms, in this context, indicate the ability of DAs to leverage local autonomy to determine when and what information should be sent to which DMs. The push is considered adaptive because a DA determines what information to push (and when) as an adaptation to changes to the DA's internal state. Once pushed, information might still be queued, pending connectivity of the DA to the network.

Even in cases where a round-trip exchange can occur, pull mechanisms increase the overall amount of traffic in the network and preclude the use of autonomy at managed devices. So, even when pull mechanisms are feasible, they should not be considered a pragmatic alternative to push mechanisms.

4.4. Efficient Data Encoding

Messages exchanged between a DA and a DM in the DTNMA should be defined in a way that allows for efficient on-the-wire encoding. DTNMA design decisions that result in smaller message sizes should be preferred over those that result in larger message sizes.

There is a relationship between message encoding and message processing time at a node. Messages with few or no encodings may simplify node processing, whereas more compact encodings may require additional activities to generate/parse encoded messages. Generally, compressing a message takes processing time at the sender and decompressing a message takes processing time at a receiver. Therefore, there is a design trade-off between minimizing message sizes and minimizing node processing.

There is a significant advantage to smaller DTNMA message sizes in a challenged network. Smaller messages require shorter periods of viable transmission for communication, they incur less retransmission cost, and they consume fewer resources when persistently stored en route in the network.

> NOTE: Naive approaches to minimizing message size through general-purpose compression algorithms do not produce minimal encodings. Data models can, and should, be designed for compact encoding from the beginning. Design strategies for compact encodings involve using structured data, hierarchical data models, and common substructures within data models. These strategies allow for compressibility beyond what would otherwise be achieved by computing large hash values over generalized data structures.

4.5. Universal, Unique Data Identification

Data elements within the DTNMA should be uniquely identifiable so that they can be individually manipulated. Further, these identifiers should be universal -- the identifier for a data element should be the same, regardless of role, implementation, or network instance.

Identification schemes that would be relative to a specific DA or specific system configuration might change over time and should be avoided. Relying on relative identification schemes would require resynchronizing relative state when nodes in a challenged network reconnect after periods of partition. This type of resynchronization should be avoided whenever possible.

> NOTE: Consider a common management technique for approximating an associative array lookup. If a managed device tracks the number of bytes passed by multiple named interfaces, then the number of bytes through a specific named interface ("int_foo") would be retrieved in the following way:
>
> 1. Query a list of ordered interface names from an agent.
> 2. Find the name that matches "int_foo", and infer the agent's index of "int_foo" from the ordered interface list. In this instance, assume that "int_foo" is the fourth interface in the list.
> 3. Query the agent (again) to now return the number of bytes passed through the fourth interface.
>
> Ignoring the inefficiency of two round-trip exchanges, this mechanism will fail if an agent implementation changes its index mapping between the first and second queries.

> The desired data being queried, "number of bytes through 'int_foo'", should be uniquely and universally identifiable and independent of how that data exists in any agent's custom implementation.

4.6. Runtime Data Definitions

The DTNMA allows for the addition of new data elements to a data model as part of the runtime operation of the management system. These definitions may represent custom data definitions that are applicable only for a particular device or network. Custom definitions should also be able to be removed from the system during runtime.

The goal of this approach is to dynamically add or remove data elements to the local runtime schemas as needed, such as the definition of new counters, new reports, or new rules.

The 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.

> NOTE: A DM could, for example, define a custom data report that includes only summary information about a specific operational event or as part of specific debugging. DAs could then produce this smaller report until it is no longer necessary, at which point the custom report could be removed from the management system.

Custom data elements should be calculated and used both as parameters for DA autonomy and for more efficient reporting to DMs. Defining new data elements allows for DAs to perform local data fusion, and defining new reporting templates allows for DMs to specify desired formats and generally save on link capacity, storage, and processing time.

4.7. Autonomous Operation

The management of applications by a DA should be achievable using only knowledge local to the DA because DAs might need to operate during times when they are disconnected from a DM.

DA autonomy may be used for simple automation of predefined tasks or to support semi-autonomous behavior in determining when to run tasks and how to configure or parameterize tasks when they are run.

Important features provided by the DA are listed below. These features work together to accomplish tasks. As such, there is commonality amongst their definitions and nature of their benefits.

Standalone Operation:

Preconfiguration allows DAs to operate without regular contact with other nodes in the network. Updates for configurations remain difficult in a challenged network, but this approach removes the requirement that a DM be in the loop during regular operations. Preconfiguring stimuli and responses on a DA during periods of connectivity allows DAs to self-manage during periods of disconnectivity.

Deterministic Behavior: Operational systems might need to act in a deterministic way, even in the absence of an operator in the loop. Deterministic behavior allows an out-of-contact DM to predict the state of a DA and to determine how a DA got into a particular state.

Engine-Based Behavior: Operational systems might not be able to deploy "mobile code" solutions [RFC4949] due to network bandwidth, memory or processor loading, or security concerns. Engine-based approaches provide configurable behavior without incurring these concerns.

Authorization and Accounting: The DTNMA does not require a specific underlying transport protocol, a specific network infrastructure, or specific network services. Therefore, mechanisms for authorization and accounting need to be present in a standard way at DAs and DMs to provide these functions if the underlying network does not. This is particularly true in cases where multiple DMs may be active concurrently in the network.

To understand the contributions of these features to a common type of behavior, consider the example of a managed device coming online with a set of preinstalled configurations. In this case, the device's standalone operation comes from the preconfiguration of its local autonomy engine. This engine-based behavior allows the system to behave in a deterministic way, and any new configurations will need to be authorized before being adopted.

Features such as deterministic processing and engine-based behavior are separate from (but do not preclude the use of) other Artificial Intelligence (AI) and Machine Learning (ML) approaches for device management.

5. Current Remote Management Approaches

Several remote management solutions have been developed for both local area networks and wide area networks. Their capabilities range from simple configuration and report generation to complex modeling of device settings, state, and behavior. All of these approaches are successful in the domains for which they have been built but are not all equally functional when deployed in a challenged network.

This section describes some of the well-known protocols for remote management and contrasts their purposes with the desirable properties of the DTNMA. The purpose of this comparison is to identify parts of existing approaches that can be adopted or adapted for use in challenged networks and where new capabilities should be created specifically for such environments.

5.1. SNMP and SMI Models

An early and widely used example of a remote management protocol is SNMP, which is currently at version 3 [RFC3410]. SNMP utilizes a request-response model to get and set data values within an arbitrarily deep object hierarchy. Objects are used to identify data such as host identifiers, link utilization metrics, error rates, and counters between application software on managing and managed devices [RFC3411]. Additionally, SNMP supports a model for unidirectional push messages, called event notifications, based on agent-defined triggering events.

SNMP relies on logical sessions with predictable round-trip latency to support its pull mechanism, but a single activity is likely to require many round-trip exchanges. Complex management can be achieved, but only through careful orchestration of real-time, end-to-end, managing-device-generated query-and-response logic.

There is existing work that uses the SNMP data model to support some low-fidelity agent-side processing; this work includes using "Distributed Management Expression MIB" [RFC2982] and "Definitions of Managed Objects for the Delegation of Management Scripts" [RFC3165]. However, agent autonomy is not an SNMP mechanism, so support for a local agent response to an initiating event is limited. In a challenged network where the delay between a managing device receiving an alert and sending a response can be significant, SNMP is insufficient for autonomous event handling.

5.1.1. The SMI Modeling Language

SNMP separates the representations for managed data models from messaging, sequencing, and encoding between managers and agents. Each data model is termed a "Management Information Base" (or "MIB") [RFC3418] and uses the Structure of Management Information (SMI) modeling language [RFC2578]. Additionally, the SMI itself is based on the ASN.1 syntax [ASN.1], which is used not just for SMI but for other, unrelated data structure specifications such as the Cryptographic Message Syntax (CMS) [RFC5652]. Separating data models from messaging and encoding is a best practice in remote management protocols and is also necessary for the DTNMA.

Each SNMP MIB is composed of managed object definitions, each of which is associated with a hierarchical Object Identifier (OID). Because of the arbitrarily deep nature of MIB object trees, the size of OIDs is not strictly bounded by the protocol (though it may be bounded by implementations).

5.1.2. SNMP and Transport

SNMPv2 [RFC3416] [RFC3417] and SNMPv3 [RFC3414] can operate over a variety of transports, including plaintext UDP/IP [RFC3417], SSH/TCP/IP [RFC5592], and DTLS/UDP/IP or TLS/TCP/IP [RFC6353].

SNMP uses an abstracted security model to provide authentication, integrity, and confidentiality. There are options for the User-based Security Model (USM) [RFC3414], which uses in-message security, and the Transport Security Model (TSM) [RFC5591], which relies on the transport to provide security functions and interfaces.

5.2. XML-Infoset-Based Protocols and YANG Data Models

Several network management protocols, including NETCONF [RFC6241], RESTCONF [RFC8040], and the Constrained Application Protocol (CoAP) Management Interface (CORECONF) [CORE-COMI], share the same XML Information Set [xml-infoset] for the information set's hierarchical managed information and XPath expressions [XPath] to identify nodes of that information model. Since they share the same information model and the same data manipulation operations, together they will be referred to as "*CONF" protocols. Each protocol, however, provides a different encoding of that information set and its related operation-specific data.

The YANG modeling language as defined in [RFC7950] is used to define the data model for these management protocols. Currently, YANG represents the IETF standard for defining managed information models.

5.2.1. The YANG Modeling Language

The YANG modeling language defines a syntax and modular semantics for organizing and accessing a device's configuration or operational information. YANG allows subdividing a full managed configuration into separate namespaces defined by separate YANG modules. Once a module is developed, it is used (directly or indirectly) on both the client and server to serve as a contract between the two. A YANG module can be complex, describing a deeply nested and interrelated set of data nodes, actions, and notifications.

Unlike the separation between ASN.1 syntax and module semantics from higher-level SMI data model semantics as discussed in Section 5.1.1, YANG defines both a text syntax and module semantics together with data model semantics.

The YANG modeling language provides flexibility in the organization of model objects to the model developer. YANG supports a broad range of data types as noted in [RFC6991]. YANG also supports the definition of parameterized Remote Procedure Calls (RPCs) and actions to be executed on managed devices as well as the definition of event notifications within the model.

Current *CONF notification logic allows a client to subscribe to the delivery of specific containers or data nodes defined in the model, on either a periodic or "on-change" basis [RFC8641]. These notification events can be filtered according to XPath or subtree filtering [XPath] [RFC6241] as described in Section 2.2 of [RFC8639].

The use of YANG for data modeling necessarily comes with some side effects, some of which are described here.

Text Naming:

Data nodes, RPCs, and notifications within a YANG data model are named by a namespace-qualified, text-based path of the module, submodule, container, and any data nodes such as lists, leaf-lists, or leaves, without any explicit hierarchical organization based on data or object type.

Existing efforts to make compressed names for YANG objects, such as the YANG Schema Item iDentifiers (SIDs) as discussed in Section 3.2 of [RFC9254], allow a node to be named by a globally unique integer value but are still relatively verbose (up to 8 bytes per item) and still must be translated into text form for things like instance identification (see below). Additionally, when representing a tree of named instances, the child elements can use differential encoding of SID integer values as "delta" integers. The mechanisms for assigning SIDs and the lifecycle of those SIDs are discussed in [RFC9595].

Text Values and Built-In Types: Because the original use of YANG with NETCONF was to model XML Information Sets, the values and built-in types are necessarily text based. JSON encoding of YANG data [RFC7951] allows for optimized representations of many built-in types; similarly, Concise Binary Object Representation (CBOR) encoding [RFC9254] allows for different optimized representations.

In particular, the YANG built-in types support a fixed range of decimal fractions (Section 9.3 of [RFC7950]) but purposefully do not support floating-point numbers. There are alternatives, such as the type `bandwidth-ieee-float32` [RFC8294] or using the "binary" type with one of the IEEE-754 encodings.

Deep Hierarchy: YANG allows for, and current YANG modules take advantage of, the ability to deeply nest a model hierarchy to represent complex combinations and compositions of data nodes. When a model uses a deep hierarchy of nodes, this necessarily means that the qualified paths to name those nodes and instances are longer than they would be in a flat namespace.

Instance Identification: The node instances in a YANG module necessarily use XPath expressions for identification. Some identification is constrained to be strictly within the YANG domain, such as "must", "when", "augment", or "deviation" statements. Other identification needs to be processed by a managed device -- for example, via the "instance-identifier" built-in type. This means that any implementation of a managed device must include XPath processing and related information model handling per Section 6.4 of [RFC7950] and its referenced documents.

Protocol Coupling: A significant amount of existing YANG tooling or modeling presumes the use of YANG data within a management protocol with specific operations available. For example, the access control model defined in [RFC8341] relies on those operations specific to the *CONF protocols for proper behavior.

The emergence of multiple NETCONF-derived protocols may make these presumptions less problematic in the future. Work to more consistently identify different types of YANG modules and their use has been undertaken to disambiguate how YANG modules should be treated [RFC8199].

Manager-Side Control: YANG RPCs and actions execute on a managed device and generate an expected, structured response. RPC execution is strictly limited to those issued by the manager. Commands are executed immediately and sequentially as they are received by the managed device, and there is no method to autonomously execute RPCs triggered by specific events or conditions.

The YANG modeling language continues to evolve as new features are needed by adopting management protocols.

5.2.2. NETCONF Protocol and Transport

NETCONF is a stateful, XML-encoding-based protocol that provides a syntax to retrieve, edit, copy, or delete any data nodes or exposed functionality on a server. It requires that underlying transport protocols support long-lived, reliable, low-latency, sequenced data delivery sessions. A bidirectional NETCONF session needs to be established before any data transfer (or notification) can occur.

The XML exchanged within NETCONF messages is structured according to YANG modules supported by the NETCONF agent, and the data nodes reside within one of possibly many datastores in accordance with the Network Management Datastore Architecture (NMDA) [RFC8342].

NETCONF transports are required to provide authentication, data integrity, confidentiality, and replay protection. Currently, NETCONF can operate over SSH/TCP/IP [RFC6242] or TLS/TCP/IP [RFC7589].

5.2.3. RESTCONF Protocol and Transport

RESTCONF is a stateless, JSON-encoding-based protocol that provides the same operations as NETCONF, using the same YANG modules for structure and the same NMDA datastores, but using RESTful exchanges over HTTP. It uses HTTP methods to express its allowed operations: GET, POST, PUT, PATCH, or DELETE data nodes within a datastore.

Although RESTCONF is a logically stateless protocol, it does rely on state within its transport protocol to achieve behaviors such as authentication and security sessions. Because RESTCONF uses the same data node semantics as NETCONF, a typical activity can involve the use of several sequential round trips of exchanges to first discover managed device state and then act upon it.

5.2.4. CORECONF Protocol and Transport

CORECONF is an emerging stateless protocol built atop CoAP [RFC7252] that defines a messaging construct developed to operate specifically on constrained devices and networks by limiting message size and fragmentation. CoAP also implements a request-response system and methods for GET, POST, PUT, and DELETE.

5.3. gRPC Network Management Interface (gNMI)

Another emerging, but not IETF-affiliated, management protocol is the gRPC Network Management Interface (gNMI) [gNMI], which is based on gRPC messaging and uses Google protobuf data modeling.

The same limitations as those listed above for RESTCONF apply to gNMI because of its reliance on synchronous HTTP exchanges and TLS for normal operations, as well as the likely deep nesting of data schemas. gNMI is capable of transporting JSON-encoded YANG-modeled data, but how to compose such data is not yet fully standardized.

5.3.1. The Protobuf Modeling Language

The data managed and exchanged via gNMI is encoded and modeled using Google protobuf, an encoding and modeling syntax not affiliated with the IETF (although an attempt has been made and abandoned [PROTOCOL-BUFFERS]).

Because the protobuf modeling syntax is a relatively low-level syntax (about the same as ASN.1 or CBOR), there are some efforts as part of the OpenConfig work [gNMI] to translate YANG modules into protobuf schemas (similar to translation to XML or JSON schemas for NETCONF and RESTCONF, respectively), but there is no required interoperability between management via gRPC or any of the *CONF protocols.

5.3.2. gRPC Protocol and Transport

The message encoding and exchange for gNMI, as the name implies, is the gRPC protocol [gRPC]. gRPC exclusively uses HTTP/2 [RFC9113] for transport and relies on some aspects specific to HTTP/2 for its operations (such as HTTP trailer fields). While not mandated by gRPC, when used to transport gNMI data, TLS is required for transport security.

5.4. Intelligent Platform Management Interface (IPMI)

A lower-level remote management protocol, intended to be used to manage hardware devices and network appliances below the operating system (OS), is the Intelligent Platform Management Interface (IPMI), standardized in [IPMI]. The IPMI is focused on health monitoring, event logging, firmware management, and Serial over LAN (SOL) remote console access in a "pre-OS or OS-absent" host environment. The IPMI operates over a companion Remote Management Control Protocol (RMCP) for messaging, which itself can use UDP for transport.

Because the IPMI and RCMP are tailored to low-level and well-connected devices within a data center, with typical workflows requiring many messaging round trips or low-latency interactive sessions, they are not suitable for operation over a challenged network.

5.5. Autonomic Networking

The future of network operations requires more autonomous behavior, including self-configuration, self-management, self-healing, and self-optimization. One approach to support this is termed "Autonomic Networking" [RFC7575].

There is a large and growing set of work within the IETF focused on developing an Autonomic Networking Integrated Model and Approach (ANIMA). The ANIMA work has developed a comprehensive reference model for distributing autonomic functions across multiple nodes in an Autonomic Networking infrastructure [RFC8993].

This work, focused on learning the behavior of distributed systems to predict future events, is an emerging network management capability. This includes the development of signaling protocols such as the GeneRic Autonomic Signaling Protocol (GRASP) [RFC8990] and the Autonomic Control Plane (ACP) [RFC8368].

Both autonomic and challenged networks require similar degrees of autonomy. However, challenged networks cannot provide the complex coordination between nodes and distributed supporting infrastructure necessary for the frequent data exchanges for negotiation, learning, and bootstrapping associated with the above capabilities.

There is some emerging work in ANIMA as to how disconnected devices might join and leave the ACP over time. However, this work is addressing a different problem than that encountered by challenged networks.

5.6. Deep Space Autonomy

Outside of the terrestrial networking community, there are existing and established remote management systems used for deep space mission operations. Two examples of such systems are the New Horizons mission to Pluto [NEW-HORIZONS] and the Double Asteroid Redirection Test (DART) mission to the asteroid Dimorphos [DART].

The DTNMA has some heritage in the concepts of deep space autonomy, but each of those mission instantiations uses mission-specific data encoding, messaging, and transport as well as mission-specific (or heavily mission-tailored) modeling concepts and languages. Part of the goal of the DTNMA is to take the proven concepts from these missions and standardize a messaging syntax as well as a modular data modeling method.

6. Motivation for New Features

Management mechanisms that provide the complete set of DTNMA desirable properties do not currently exist. This is not surprising, since autonomous management in the context of a challenged networking environment is a new and emerging use case.

In particular, a management architecture is needed that integrates the following motivating features.

Open-Loop Control: Freedom from a request-response architecture, API, or other presumption of timely round-trip communications. This is particularly important when managing networks that are not built over an HTTP or TCP/TLS infrastructure.

Standard Autonomy Model: An autonomy model that allows for standard expressions of policy to guarantee deterministic behavior across devices and vendor implementations.

Compressible Model Structure: A data model that allows for very compact encodings by defining and exploiting common structures for data schemas.

Combining these new features with existing mechanisms for message data exchange (such as BP), data representations (such as CBOR), and data modeling languages (such as YANG) will form a pragmatic approach to defining challenged network management.

7. Reference Model

This section describes a reference model for analyzing network management concepts for challenged networks (generally) and those conforming to the DTN architecture (in particular). The goal of this section is to describe how DTNMA services provide DTNMA desirable properties.

7.1. Important Concepts

Like other network management architectures, the DTNMA draws a logical distinction between a managed device and a managing device. Managed devices use a DA to manage resident applications. Managing devices use a DM to both monitor and control DAs.

The terms "managing" and "managed" represent logical characteristics of a device and are not, themselves, mutually exclusive. For example, a managed device might, itself, also manage some other device in the network. Therefore, a device may support either or both of these characteristics.

The DTNMA differs from some other management architectures in three significant ways, all related to the need for a device to self-manage when disconnected from a managing device.

Pre-Shared Definitions: Managing and managed devices should operate using pre-shared data definitions and models. This implies that static definitions should be standardized whenever possible and that managing and managed devices may need to negotiate definitions during periods of connectivity.

Agent Self-Management: A managed device may find itself disconnected from its managing device. In many challenged networking scenarios, a managed device may spend the majority of its time without a regular connection to a managing device. In these cases, DAs manage themselves by applying pre-shared policies received from managing devices.

Command-Based Interface: Managing devices communicate with managed devices through a command-based interface. Instead of exchanging variables, objects, or documents, a managing device issues commands to be run by a managed device. These commands may create or update variables, change datastores, or impact the managed device in ways similar to other network management approaches. The use of commands is, in part, driven by the need for DAs to receive updates from both remote management devices and local autonomy. The use of Controls for the implementation of commands is discussed in more detail in Section 9.5.

7.2. Model Overview

A DTNMA reference model is provided in Figure 2 below. In this reference model, applications and services on a managing device communicate with a DM that uses pre-shared definitions to create a set of policy directives that can be sent to a managed device's DA via a command-based interface. The DA provides local monitoring and control (commanding) of the applications and services resident on the managed device. The DA also performs local data fusion as necessary to synthesize data products (such as reports) that can be sent back to the DM when appropriate.

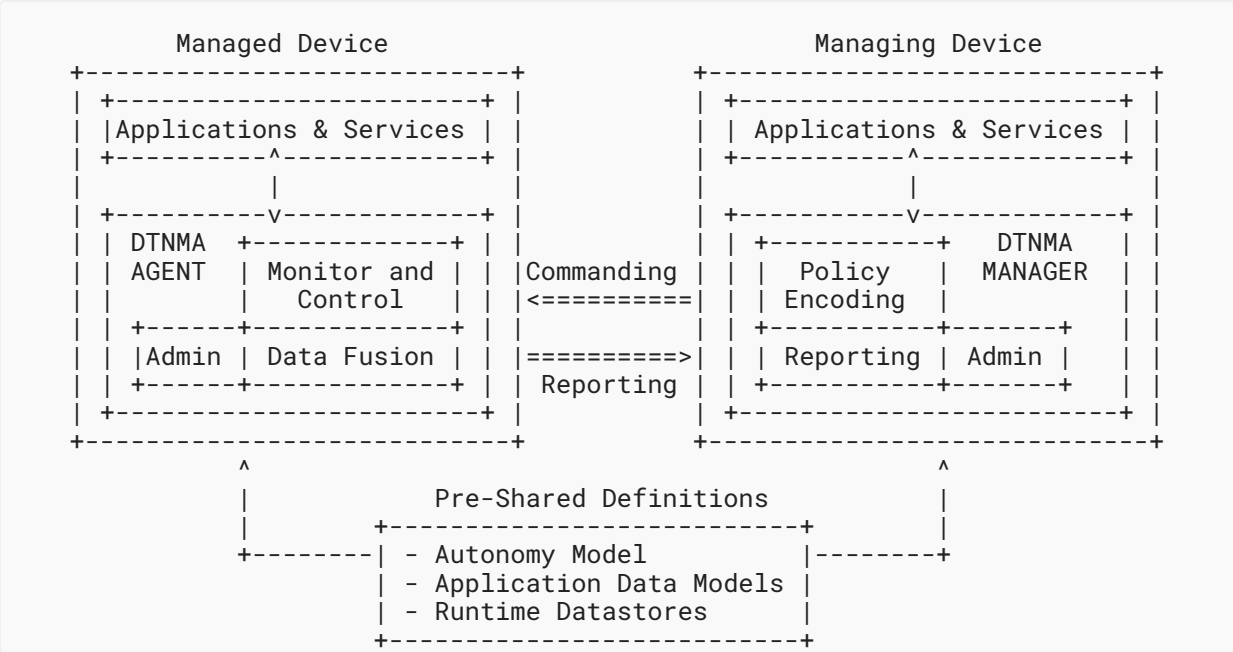

Figure 2: DTNMA Reference Model

This model preserves the familiar concept of "managers" resident on managing devices and "agents" resident on managed devices. However, the DTNMA model is unique in how the DM and DA operate. The DM is used to preconfigure DAs in the network with management policies. It is expected that the DAs, themselves, perform monitoring and control functions on their own. In this way, a properly configured DA may operate without a reliable connection back to a DM.

7.3. Functional Elements

The reference model illustrated in Figure 2 implies the existence of certain logical components whose roles and responsibilities are discussed in this section.

7.3.1. Managed Applications and Services

By definition, managed applications and services reside on a managed device. These software entities can be controlled through some interface by the DA, and their state can be sampled as part of periodic monitoring. It is presumed that the DA on the managed device has the proper data model, control interface, and permissions to alter the configuration and behavior of these software applications.

7.3.2. DTNMA Agent (DA)

A DA resides on a managed device. As is the case with other network management approaches, this agent is responsible for the monitoring and control of the applications local to that device. Unlike other network management approaches, the agent accomplishes this task without a regular connection to a DM.

The DA performs three major functions on a managed device: the monitoring and control of local applications, production of data analytics, and the administrative control of the agent itself.

7.3.2.1. Monitoring and Control

DAs monitor the status of applications running on their managed device and selectively control those applications as a function of that monitoring. The following components are used to perform monitoring and control on an agent.

Rule Database:
: Each DA maintains a database of policy expressions that form rules regarding the behavior of the managed device. Within this database, each rule regarding behavior is a tuple of a stimulus and a response. Within the DTNMA, these rules are the embodiment of policy expressions received from DMs and evaluated at regular intervals by the autonomy engine. The rule database is the collection of active rules known to the DA.

Autonomy Engine:
: The DA autonomy engine monitors the state of the managed device, looking for predefined stimuli and, when such stimuli are encountered, issuing a predefined response. To the extent that this function is driven by the rule database, this engine acts as a policy execution engine. This engine may also be directly configured by managers during periods of connectivity for actions separate from those in the rule database (such as enabling or disabling sets of rules). Once configured, the engine may function without other access to any managing device. This engine may also reconfigure itself as a function of policy.

Application Control Interfaces:
: DAs support control interfaces for all managed applications. Control interfaces are used to alter the configuration and behavior of an application. These interfaces may be custom for each application or as provided through a common framework, protocol, or OS.

7.3.2.2. Data Fusion

DAs generate new data elements as a function of the current state of the managed device and its applications. These new data products may take the form of individual data values or of new collections of data used for reporting. The logical components responsible for these behaviors are as follows.

Application Data Interfaces:
: DAs support mechanisms by which important state is retrieved from various applications resident on the managed device. These data interfaces may be custom for each application or as provided through a common framework, protocol, or OS.

Data Value Generators:
: DAs may support the generation of new data values as a function of other values collected from the managed device. These data generators may be configured with descriptions of data values, and the data values they generate may be included in the overall monitoring and reporting associated with the managed device.

Report Generators:
: DAs may, as appropriate, generate collections of data values and provide them to whatever local mechanism takes responsibility for their eventual transmission (or expiration and removal). Reports can be generated as a matter of policy or in response to the handling of critical events (such as errors) or other logging needs. The generation of a report is independent of whether there exists any connectivity between a DA and a DM.

7.3.2.3. Administration

DAs perform a variety of administrative services in support of their configuration, such as the following.

Manager Mapping:
: The DTNMA allows for a many-to-many relationship amongst DAs and DMs. A single DM may configure multiple DAs, and a single DA may be configured by multiple DMs. Multiple managers may exist in a network for at least the following two reasons. First, different managers may exist to control different applications on a device. Second, multiple managers increase the likelihood of an agent encountering a manager when operating in a sparse or challenged environment.

  While multiple managers are needed for proper operation in a dynamically partitioned network, conflicting information from different managers can result. Implementations of the DTNMA should consider conflict resolution mechanisms. Such mechanisms might include analyzing managed content, time, agent location, or other relevant information to select one manager input over other manager inputs.

Data Verifiers:
: DAs might handle large amounts of data produced by various sources, to include data from local managed applications, remote managers, and self-calculated values. DAs should ensure, when possible, that externally generated data values have the proper syntax and semantic constraints (e.g., data type and ranges) and any required authorization.

Access Controllers:
: DAs support authorized access to the management of individual applications, to include the administrative management of the agent itself. This means that a manager may only set policy on the agent pursuant to verifying that the manager is authorized to do so.

7.3.3. Managing Applications and Services

Managing applications and services reside on a managing device and serve as both the source of DA policy statements and the target of DA reporting. They may operate with or without an operator in the loop.

Unlike management applications in unchallenged networks, these applications cannot exert closed-loop control over any managed device application. Instead, they exercise open-loop control by producing policies that can be configured and enforced on managed devices by DAs.

> NOTE: Closed-loop control in this context refers to the practice of waiting for a response from a managed device prior to issuing new commands to that device. These "loops" may be closed quickly (in milliseconds) or over much longer periods (hours, days, years). The alternative to closed-loop control is open-loop control, where the issuance of new commands is not dependent on receiving responses to previous commands. Additionally, there might not be a one-to-one mapping between commands and responses. A DA may, for example, produce a single response that represents the end state of applying multiple commands.

7.3.4. DTNMA Manager (DM)

A DM resides on a managing device. This manager provides an interface between various managing applications and services and the DAs that enforce their policies. In providing this interface, DMs translate between whatever innate interface exists to various managing applications and the autonomy models used to encode management policy.

The DM performs three major functions on a managing device: policy encoding, reporting, and administration.

7.3.4.1. Policy Encoding

DMs translate policy directives from managing applications and services into standardized policy expressions that can be recognized by DAs. The following logical components are used to perform this policy encoding.

Application Control Interfaces:
: DMs support control interfaces for managing applications. These control interfaces are used to receive desired policy statements from applications. These interfaces may be custom for each application or as provided through a common framework, protocol, or OS.

Policy Encoders:
: DAs implement a standardized autonomy model comprising standardized data elements. This allows the open-loop control structures provided by managing applications to be represented in a common language. Policy encoders perform this encoding function.

Policy Aggregators:
: DMs collect multiple encoded policies into messages that can be sent to DAs over the network. This implies the proper addressing of agents and the creation of messages that support store-and-forward operations. It is recommended that control messages be packaged using BP bundles when there may be intermittent connectivity between DMs and DAs.

7.3.4.2. Reporting

DMs receive reports on the status of managed devices during periods of connectivity with the DAs on those devices. The following logical components are needed to implement reporting capabilities on a DM.

Report Collectors:
: DMs receive reports from DAs in an asynchronous manner. This means that reports may be received out of chronological order and in ways that are difficult or impossible to associate with a specific policy from a managing application. DMs collect these reports and extract their data in support of subsequent data analytics.

Data Analyzers:
: DMs review sets of data reports from DAs with the purpose of extracting relevant data to communicate with managing applications. This may include simple data extraction or may include more complex processing such as data conversion, data fusion, and appropriate data analytics.

Application Data Interfaces:
: DMs support mechanisms by which data retrieved from DAs may be provided back to managing devices. These interfaces may be custom for each application or as provided through a common framework, protocol, or OS.

7.3.4.3. Administration

DMs in the DTNMA perform a variety of administrative services, such as the following.

Agent Mappings:
: The DTNMA allows DMs to communicate with multiple DAs. However, not every agent in a network is expected to support the same set of application data models or otherwise have the same set of managed applications running. For this reason, DMs determine individual DA capabilities to ensure that only appropriate Controls are sent to a DA.

Data Verifiers:
: DMs handle large amounts of data produced by various sources, to include data from managing applications and DAs. DMs should ensure, when possible, that data values received from DAs over a network have the proper syntax and semantic constraints (e.g., data type and ranges) and any required authorization.

Access Controllers:
: DMs should only send Controls to DAs when the manager is configured with appropriate access to both the agent and the applications being managed.

7.3.5. Pre-Shared Definitions

A consequence of operating in a challenged environment is the potential inability to negotiate information in real time. For this reason, the DTNMA requires that managed and managing devices operate using pre-shared definitions rather than relying on data definition negotiation.

The three types of pre-shared definitions in the DTNMA are the DA autonomy model, managed application data models, and any runtime data shared by managers and agents.

Autonomy Model:
: A DTNMA autonomy model represents the data elements and associated autonomy structures that define the behavior of the agent autonomy engine. A standardized autonomy model allows for individual implementations of DAs and DMs to interoperate. A standardized model also provides guidance to the design and implementation of both managed and managing applications.

Application Data Models:
: As with other network management architectures, the DTNMA presupposes that managed applications (and services) define their own data models. These data models include the data produced by, and Controls implemented by, the application. These models are expected to be static for individual applications and standardized for applications implementing standard protocols.

Runtime Datastores:
: Runtime datastores, by definition, include data that is defined at runtime. As such, the data is not pre-shared prior to the deployment of DMs and DAs. Pre-sharing in this context means that DMs and DAs are able to define and synchronize data elements prior to their operational use in the system. This synchronization happens during periods of connectivity between DMs and DAs.

8. Desired Services

This section describes the services provided by DTNMA components on both managing and managed devices. Most of the services discussed in this section attempt to provide continuous operation of a managed device through periods of no connectivity with a managing device.

8.1. Local Monitoring and Control

DTNMA monitoring is associated with some DA autonomy engine. The term "monitoring" implies regular access to information such that state changes may be acted upon within some response time period.

Predicate autonomy on a managed device should collect state associated with the device at regular intervals and evaluate that collected state for any changes that require a preventative or corrective action. Similarly, this monitoring may cause the device to generate one or more reports destined to a managing device.

Like monitoring, DTNMA control results in actions by the agent to change the state or behavior of the managed device. All control in the DTNMA is local control. In cases where there exists a timely connection to a DM, received Controls are still evaluated and run locally as part of local autonomy. In this case, the autonomy stimulus is the receipt of the Control, and the response is to immediately run the Control. In this way, there is never a dependency on a session or other stateful exchange with any remote entity.

8.2. Local Data Fusion

DTNMA fusion services produce new data products from existing state on the managed device. These fusion products can be anything from simple summations of sampled counters to complex calculations of behavior over time.

Fusion is an important service in the DTNMA because fusion products are part of the overall state of a managed device. Complete knowledge of this overall state is important for the management of the device, and the predicates of rules on a DA may refer to fused data.

In situ data fusion is an important function, as it allows for the construction of intermediate summary data, the reduction of stored and transmitted raw data, and possibly fewer predicates in rule definitions; this type of data fusion insulates the data source from conclusions drawn from that data.

The DTNMA requires fusion to occur on the managed device itself. If the network is partitioned such that no connection to a managing device is available, then fusion needs to happen locally. Similarly, connections to a managing device might not remain active long enough for round-trip data exchange or may not have the bandwidth to send all sampled data.

> NOTE: The DTNMA does not restrict the storage and transmission of raw (pre-fused) data. Such raw data can be useful for debugging managed devices, understanding complex interactions and underlying conditions, and tuning for better performance and/or better outcomes.

8.3. Remote Configuration

DTNMA configuration services update the local configuration of a managed device with the intent of impacting the behavior and capabilities of that device.

The DTNMA configuration service is unique in that the selection of managed device configurations occurs as a function of the state of the device. This implies that management proxies on the device store multiple configuration functions that can be applied as needed without consultation from a managing device.

This approach differs from other management concepts of selecting from multiple datastores. DTNMA configuration functions can target individual data elements and can calculate new values from local device state.

When detecting stimuli, the agent autonomy engine supports a mechanism for evaluating whether application monitoring data or runtime data values are recent enough to indicate a change of state. In cases where data has not been updated recently, it may be considered stale and therefore not used to reliably indicate that some stimulus has occurred.

8.4. Remote Reporting

DTNMA reporting services collect information known to the managed device and prepare it for eventual transmission to one or more managing devices. The contents of these reports, and the frequency at which they are generated, occur as a function of the state of the managed device, independent of the managing device.

Once generated, it is expected that reports might be queued, pending a connection back to a managing device. Therefore, reports need to be differentiable as a function of the time they were generated.

> NOTE: When reports are queued pending transmission, the overall storage capacity at the queuing device needs to be considered. There may be cases where queued reports can be considered expired because they have been either queued for too long or replaced by a newer report. When a report is considered expired, it may be considered for removal and, thus, never transmitted. This consideration is expected to be part of the implementation of the queuing device and not the responsibility of the reporting function within the DTNMA.

When reports are sent to a managing device over a challenged network, they may arrive out of order due to taking different paths through the network or being delayed due to retransmissions. A managing device should not infer meaning from the order in which reports are received.

Reports may or may not be associated with a specific Control. Some reports may be annotated with the Control that caused the report to be generated. Sometimes, a single report will represent the end state of applying multiple Controls.

8.5. Authorization

Both local and remote services provided by the DTNMA affect the behavior of multiple applications on a managed device and may interface with multiple managing devices.

Authorization services enforce the potentially complex mapping of other DTNMA services amongst managed and managing devices in the network. For example, fine-grained access control can determine which managing devices receive which reports, and what Controls can be used to alter which managed applications.

This is particularly beneficial in networks that deal with either multiple administrative entities or overlay networks that cross administrative boundaries. Allowlists, blocklists, key-based infrastructures, or other schemes may be used for this purpose.

9. Logical Autonomy Model

An important characteristic of the DTNMA is the shift in the role of a managing device. One way to describe the behavior of the agent autonomy engine is to describe the characteristics of the autonomy model it implements.

This section describes a logical autonomy model in terms of the abstract data elements that would comprise the model. Defining abstract data elements allows for an unambiguous discussion of the behavior of an autonomy model without mandating a particular design, encoding, or transport associated with that model.

9.1. Overview

A managing autonomy capability on a potentially disconnected device needs to behave in both an expressive and deterministic way. Expressivity allows for the model to be configured for a wide range of future situations. Determinism allows for the forensic reconstruction of device behavior as part of debugging or recovery efforts. It also is necessary to ensure predictable behavior.

> NOTE: The use of predicate logic and a stimulus-response system does not conflict with the use of higher-level autonomous functions or the incorporation of Machine Learning (ML). Specifically, the DTNMA deterministic autonomy model can coexist with other autonomous functions managing applications and network services.
>
> An example of such coexistence is the use of the DTNMA model to ensure that a device stays within safe operating parameters while a less deterministic ML model directs other behaviors for the device.

The DTNMA autonomy model is a rule-based model in which individual rules associate a pre-identified stimulus with a preconfigured response to that stimulus.

Stimuli are identified using one or more predicate logic expressions that examine aspects of the state of the managed device. Responses are implemented by running one or more procedures on the managed device.

In its simplest form, a stimulus is a single predicate expression of a condition that examines some aspect of the state of the managed device. When the condition is met, a predetermined response is applied. This behavior can be captured using the construct:

```
IF <condition 1> THEN <response 1>
```

In more complex forms, a stimulus may include both a common condition shared by multiple rules and a specific condition for each individual rule. If the common condition is not met, the evaluation of the specific condition of each rule sharing the common condition can be skipped. In this way, the total number of predicate evaluations can be reduced. This behavior can be captured using the construct:

```
IF <common condition> THEN
  IF <specific condition 1> THEN <response 1>
  IF <specific condition 2> THEN <response 2>
  IF <specific condition 3> THEN <response 3>
```

> NOTE: The DTNMA model remains a stimulus-response system, regardless of whether a common condition is part of the stimulus. However, it is recommended that implementations incorporate a common condition because of the efficiency provided by such a bulk evaluation.
>
> NOTE: One use of a stimulus "common condition" is to associate the condition with an onboard event such as the expiring of a timer or the changing of a monitored value.

The DTNMA does not prescribe when to evaluate rule stimuli. Implementations may choose to evaluate rule stimuli at periodic intervals (such as 1 Hz or 100 Hz). When stimuli include onboard events, implementations may choose to perform an immediate evaluation at the time of the event rather than waiting for a periodic evaluation.

The flow of data into and out of the agent autonomy engine is illustrated in Figure 3.

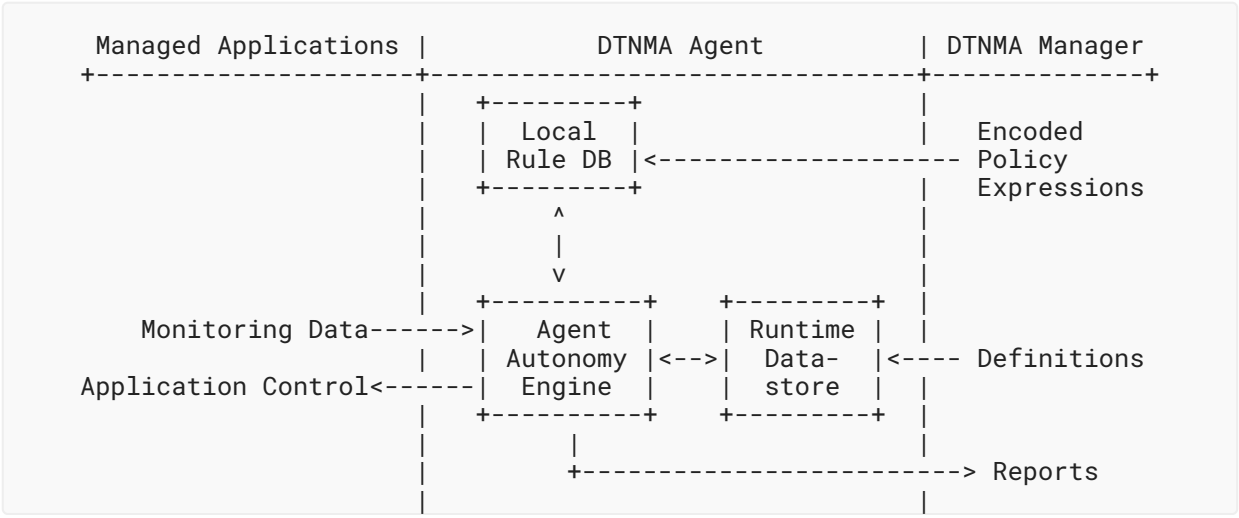

Figure 3: DTNMA Autonomy Model

In the model shown in Figure 3, the autonomy engine stores the combination of stimulus conditions and associated responses as a set of "rules" in a rule database. This database is updated through the execution of the autonomy engine and as configured from policy statements received by DMs.

Stimuli are detected by examining the state of applications as reported through application monitoring interfaces and through any locally derived data. Local data is calculated in accordance with definitions also provided by DMs as part of the runtime datastore.

Responses to stimuli may include updates to the rule database, updates to the runtime datastore, Controls sent to applications, and the generation of reports.

9.2. Model Characteristics

There are several practical challenges to the implementation of a distributed rule-based system. Large numbers of rules may be difficult to understand, deconflict, and debug. Rules whose conditions are given by fused or other dynamic data may require data logging and reporting for deterministic offline analysis. Rule differences across managed devices may lead to oscillating effects. This section identifies those characteristics of an autonomy model that might help implementations mitigate some of these challenges.

There are a number of ways to represent data values, and many data modeling languages exist for this purpose. When considering how to model data in the context of the DTNMA autonomy model, there are some modeling features that should be present to enable functionality. There are also some modeling features that should be prevented to avoid ambiguity.

Conventional network management approaches favor flexibility in their data models. The DTNMA stresses deterministic behavior that supports forensic analysis of agent activities "after the fact". As such, the following statements should be true of all data representations relating to DTNMA autonomy.

Strong Typing: The predicates and expressions that comprise the autonomy services in the DTNMA should require strict data typing. This avoids errors associated with implicit data conversions and helps detect misconfigurations.

Acyclic Dependency: Many dependencies exist in an autonomy model, particularly when combining individual expressions or results to create complex behaviors. Implementations that conform to the DTNMA need to prevent circular dependencies.

Fresh Data: Autonomy models operating on data values presume that their data inputs represent the actionable state of the managed device. If a data value has failed to be refreshed within a time period, autonomy might incorrectly infer an operational state. Regardless of whether a data value has changed, DTNMA implementations should provide some indicator of whether the data value is "fresh", i.e., meaning that it still represents the current state of the device.

Pervasive Parameterization: Where possible, autonomy model objects should support parameterization to allow for flexibility in the specification. Parameterization allows for the definition of fewer unique model objects and also can support the substitution of local device state when exercising device control or data reporting.

Configurable Cardinality: The number of data values that can be supported in a given implementation is finite. For devices operating in challenged environments, the number of supported objects may be far fewer than the number of objects that can be supported by devices in well-resourced environments. DTNMA implementations should define limits to the number of supported objects that can be active in a system at one time, as a function of the resources available to the implementation.

Control-Based Updates: The agent autonomy engine changes the state of the managed device by running Controls on the device. This is different from approaches where the behavior of a managed device is influenced by updating configuration values, such as in a table or datastore. Altering behavior via one or more Controls allows checking all preconditions before making changes as well as providing more granularity in the way in which the device is updated. Where necessary, Controls can be defined to perform bulk updates of configuration data so as not to lose that update modality. One important update precondition is that the system is not performing an action that would prevent the update (such as currently applying a competing update).

9.3. Data Value Representation

The expressive representation of simple data values is fundamental to the successful construction and evaluation of predicates in the DTNMA autonomy model. When defining such values, there are useful distinctions regarding how values are identified and whether values are generated in a way that is internal or external to the autonomy model.

A DTNMA data value should combine a base type (e.g., integer, real, string) representation with relevant semantic information. Base types are used for proper storage and encoding. Semantic information allows for additional typing, constraint definitions, and mnemonic naming. This expanded definition of data values allows for better predicate construction, better evaluation, and early type checking.

Data values may further be annotated based on whether their value is the result of a DA calculation or the result of some external process on the managed device. For example, operators may wish to know which values can be updated by actions on the DA versus which values (such as sensor readings) cannot be reliably changed because they are calculated in a way that is external to the DA.

9.4. Data Reporting

The DTNMA autonomy model should, as required, report on the state of its managed device (to include the state of the model itself). This reporting should be done as a function of the changing state of the managed device, independent of the connection to any managing device. Queuing reports allows for later forensic analysis of device behavior; this feature is a desirable property of DTNMA management.

DTNMA data reporting consists of the production of some data report instance conforming to a data report schema. The use of schemas allows a report instance to identify the schema to which it conforms instead of carrying the structure in the report itself. This approach can significantly reduce the size of generated reports.

The DTNMA data reporting concept is intentionally distinct from the concept of exchanging datastores across a network. It is envisioned that a DA might generate a data report instance of a data report schema at regular intervals or in response to local events. In this model, many report schemas may be defined to capture unique, relevant combinations of known data values rather than sending bulk datastores off-platform for analysis.

> NOTE: It is not required that data report schemas be tabular in nature. Individual implementations might define tabular schemas for table-like data and other report schemas for more heterogeneous reporting.

9.5. Command Execution

The agent autonomy engine requires that managed devices issue commands on themselves as if they were otherwise being controlled by a managing device. The DTNMA implements commanding through the use of Controls and macros.

Controls represent parameterized, predefined procedures run by the DA either as directed by the DM or as part of a rule response from the DA autonomy engine. Macros represent ordered sequences of Controls.

Controls are conceptually similar to RPCs in that they represent parameterized functions run on the managed device. However, they are conceptually dissimilar to RPCs in that they do not have a concept of a return code because they operate over an asynchronous transport. The concept of a return code in an RPC implies a synchronous relationship between the caller of the procedure and the procedure being called, which might not be possible within the DTNMA.

The success or failure of a Control may be handled locally by the agent autonomy engine. Local error handling is particularly important in this architecture, given the potential for long periods of disconnectivity between a DA and a DM. The failure of one or more Controls is part of the state of the DA and can be used to trigger rules within the DA autonomy engine.

The impact of a Control is externally observable via the generation and eventual examination of data reports produced by the managed device.

The failure of certain Controls might leave a managed device in an undesirable state. Therefore, it is important that there be consideration for Control-specific recovery mechanisms (such as a rollback or safing mechanism). When a Control that is part of a macro (such as in an autonomy response) fails, there may be a need to implement a safe state for the managed device based on the nature of the failure.

> NOTE: The use of the term "Control" in the DTNMA is derived in part from the concept of Command and Control (C2), where control implies the operational instructions undertaken to implement (or maintain) a commanded objective. The DA autonomy engine implements controls on a managed device to allow it to fulfill some commanded objective known by a (possibly disconnected) managing device.
>
> For example, a device might be commanded to maintain a safe internal thermal environment. Actions taken by a DA to manage heaters, louvers, and other temperature-affecting components are controls taken in service of that commanded objective.

9.6. Predicate Autonomy Rules

As discussed in Section 9.1, the DTNMA rule-based stimulus-response system associates stimulus detection with a predetermined response. Rules may be categorized based on whether (1) their stimuli include generic statements of managed device state or (2) they are optimized to only consider the passage of time on the device.

State-based rules are those whose stimulus is based on the evaluated state of the managed device. Time-based rules are a unique subset of state-based rules whose stimulus is given only by a time-based event. Implementations might create different structures and evaluation mechanisms for these two different types of rules to achieve more efficient processing on a platform.

10. Use Cases

Using the autonomy model defined in Section 9, this section describes flows through sample configurations conforming to the DTNMA. These use cases illustrate remote configuration, local monitoring and control, support for multiple DMs, and data fusion.

10.1. Notation

The use cases presented in this section are documented with a shorthand notation to describe the types of data sent between managers and agents. This notation, outlined in Table 1, leverages the definitions of the autonomy model components defined in Section 9.

Term	Definition	Example
EDD#	Externally Defined Data -- a data value defined in a way that is external to the DA.	EDD1, EDD2
V#	Variable -- a data value defined in a way that is internal to the DA.	V1 = EDD1 + 7
EXPR	Predicate expression -- used to define a rule stimulus.	V1 > 5
ID	DTNMA Object Identifier.	V1, EDD2
ACL#	Enumerated Access Control List.	ACL1
DEF(ACL, ID, EXPR)	Define "ID" from expression. Allow DMs in ACL to see this ID.	DEF(ACL1, V1, EDD1 + EDD2)
PROD(P, ID)	Produce "ID" according to predicate P. P may be a time period (1 second, or 1s) or an expression (EDD1 > 10).	PROD(1s, EDD1)
RPT(ID)	A report instance containing data named "ID".	RPT(EDD1)

Table 1: Terminology

These notations do not imply any implementation approach. They only provide a succinct syntax for expressing the data flows in the use case diagrams in the remainder of this section.

10.2. Serialized Management

This nominal configuration shows a single DM interacting with multiple DAs. The control flow for this scenario is outlined in Figure 4.

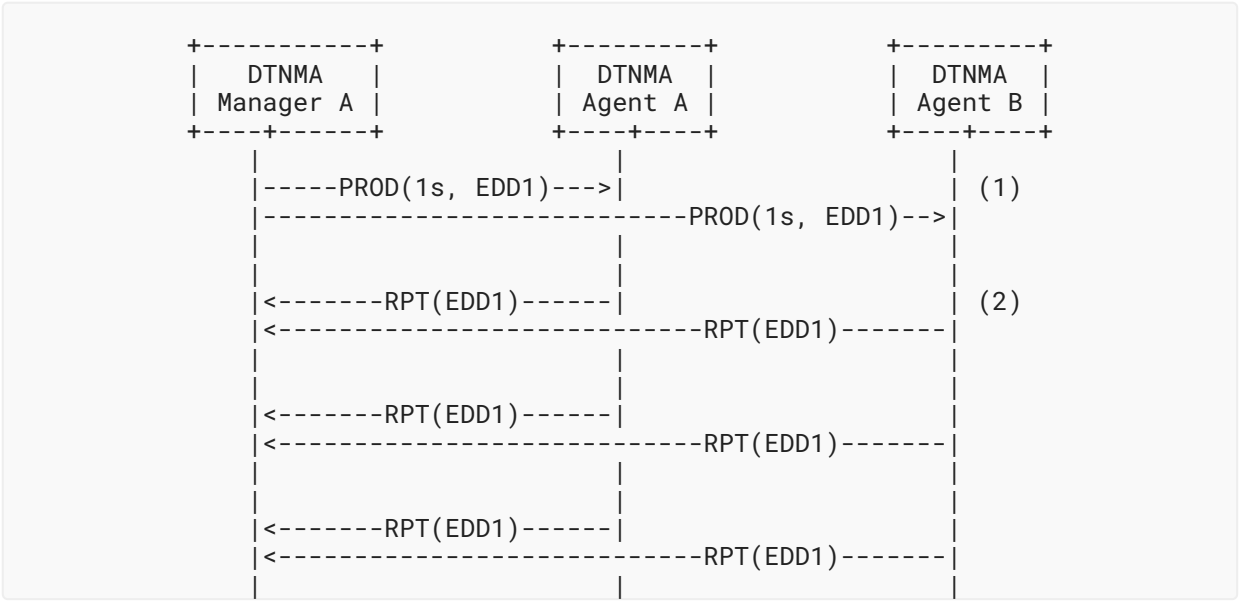

Figure 4: Serialized Management Control Flow

In a serialized management scenario, a single DM interacts with multiple DAs.

In this figure, DM A sends a policy to DAs A and B to report the value of an EDD (EDD1) every second (step 1). Each DA receives this policy and configures their respective autonomy engines for this production. Thereafter (step 2), each DA produces a report containing data element EDD1; each such report is then sent back to the DM.

This behavior continues without any additional communications from the DM.

10.3. Intermittent Connectivity

Building on the nominal configuration discussed in Section 10.2, this scenario shows a challenged network in which connectivity between DA B and the DM is temporarily lost. The control flow for this case is outlined in Figure 5.

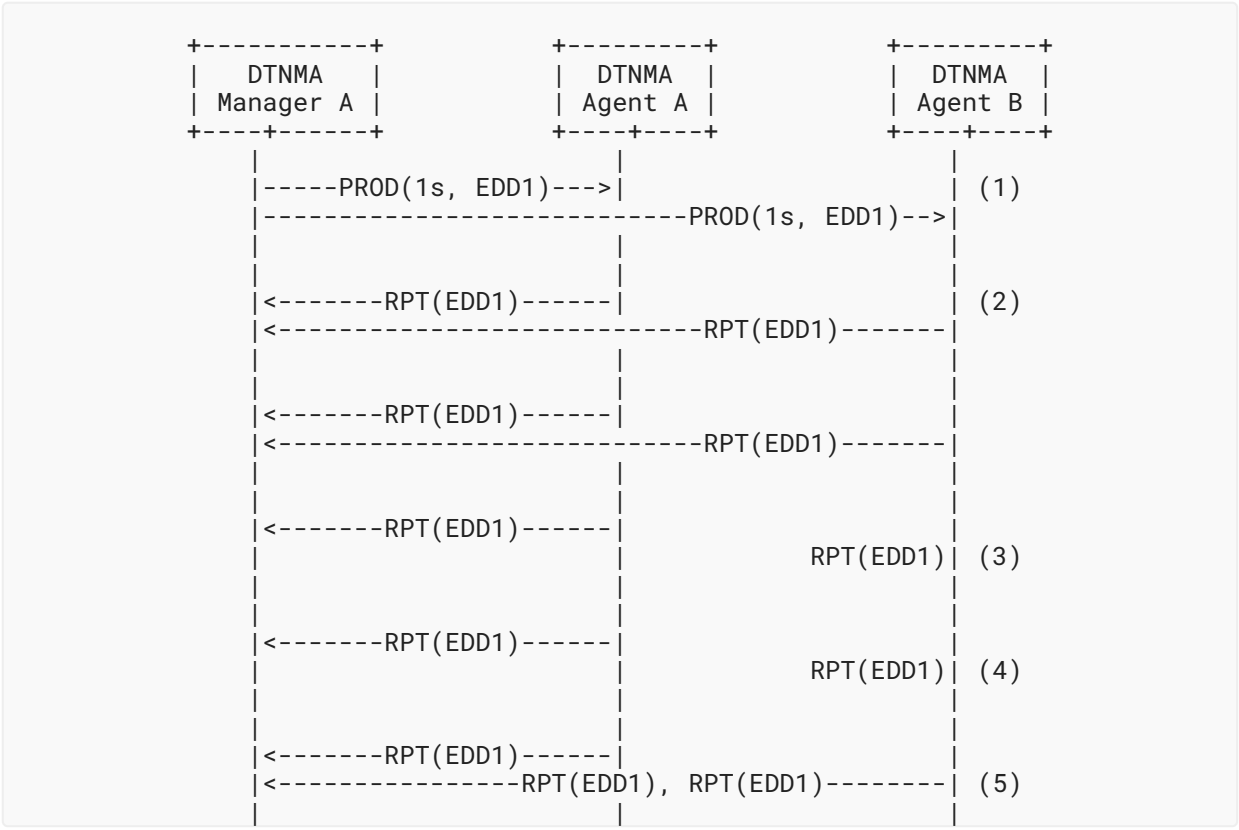

Figure 5: Challenged Management Control Flow

In a challenged network, DAs store reports, pending a transmit opportunity.

In this figure, DM A sends a policy to DAs A and B to produce an EDD (EDD1) every second (step 1). Each DA receives this policy and configures their respective autonomy engines for this production. Produced reports are transmitted when there is connectivity between the DA and DM (step 2).

At some point, DA B loses the ability to transmit in the network (steps 3 and 4). During this time period, DA B continues to produce reports, but they are queued for transmission. This queuing might be done by the DA itself or by a supporting transport such as BP. Eventually (and before the next scheduled production of EDD1), DA B is able to transmit in the network again (step 5), and all queued reports are sent at that time. DA A maintains connectivity with the DM during steps 3-5 and continues to send reports as they are generated.

10.4. Open-Loop Reporting

This scenario illustrates the DTNMA open-loop control paradigm, where DAs manage themselves in accordance with policies provided by DMs and provide reports to DMs based on these policies.

The control flow shown in Figure 6 includes an example of data fusion, where multiple policies configured by a DM result in a single report from a DA.

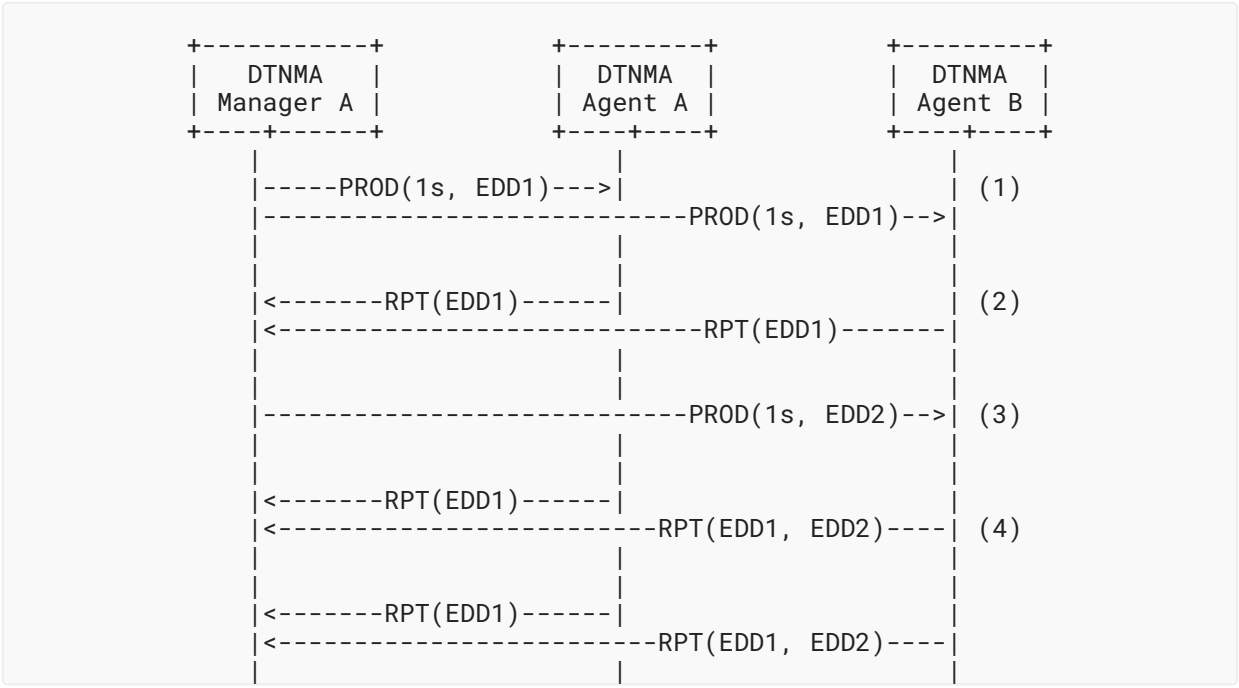

Figure 6: Consolidated Management Control Flow

A many-to-one mapping between management policy and device state reporting is supported by the DTNMA.

In this figure, DM A sends a policy statement in the form of a rule to DAs A and B, which instructs the DAs to produce a report for EDD1 every second (step 1). Each DA receives this policy, which is stored in its respective rule database, and configures its autonomy engine. Reports are transmitted by each DA when produced (step 2).

At a later time, DM A sends an additional policy to DA B, requesting the production of a report for EDD2 every second (step 3). This policy is added to DA B's rule database.

Following this policy update, DA A will continue to produce EDD1, and DA B will produce both EDD1 and EDD2 (step 4). However, DA B may provide these values to the DM in a single report rather than as two independent reports. In this way, there is no direct mapping between the consolidated reports sent by DA B (step 4) and the two different policies sent to DA B that caused that report to be generated (steps 1 and 3).

10.5. Multiple Administrative Domains

The managed applications on a DA may be controlled by different administrative entities in a network. The DTNMA allows DAs to communicate with multiple DMs in the network, such as in cases where there is one DM per administrative domain.

Whenever a DM sends a policy expression to a DA, that policy expression may be associated with authorization information. One method of representing this is an ACL.

The use of an ACL in this use case does not imply that the DTNMA requires ACLs to annotate policy expressions. ACLs and their representation in this context are for example purposes only.

The ability of one DM to access the results of policy expressions configured by some other DM will be limited to the authorization annotations of those policy expressions.

An example of multi-manager authorization is illustrated in Figure 7.

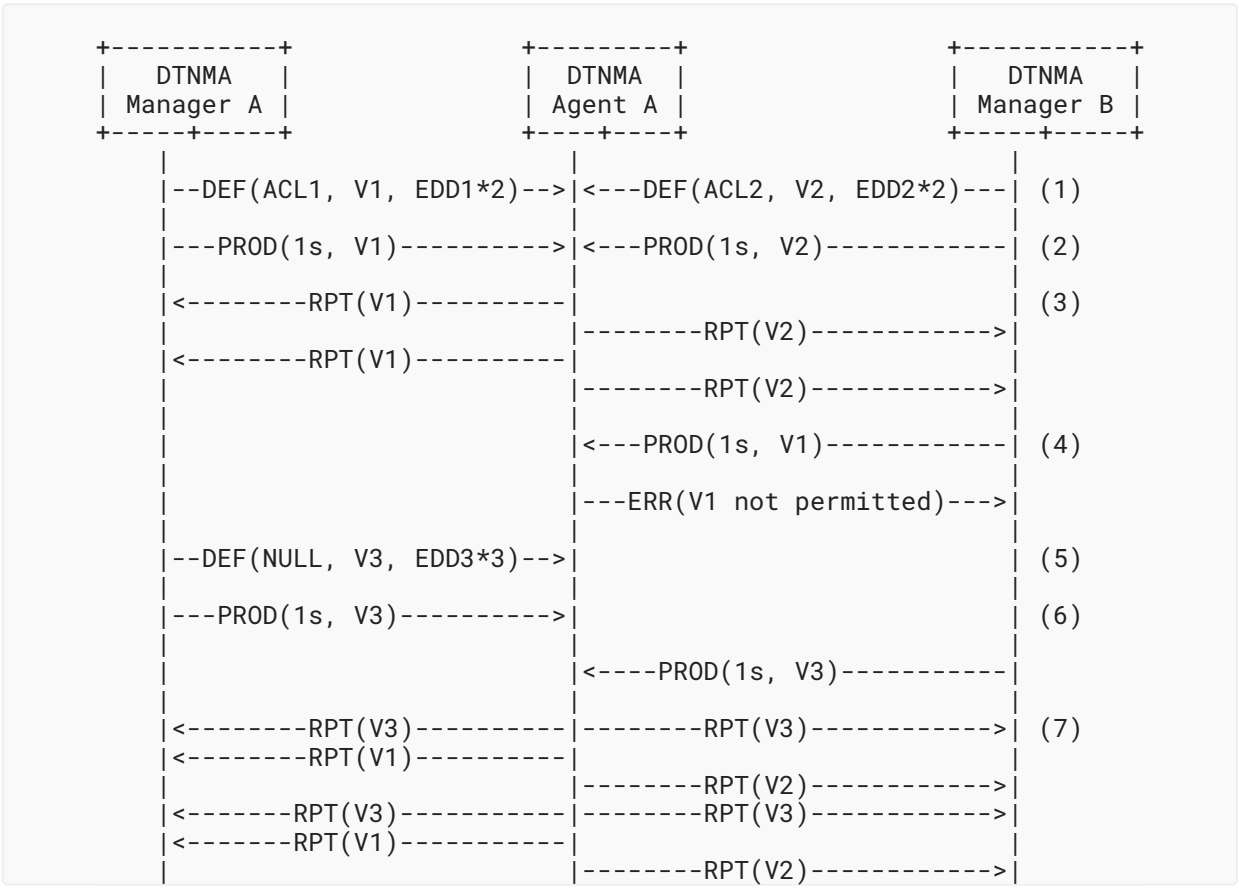

Figure 7: Multiplexed Management Control Flow

Multiple DMs may interface with a single DA, particularly in complex networks.

In this figure, both DM A and DM B send policies to DA A (step 1). DM A defines a variable (V1) whose value is given by the mathematical expression (EDD1 * 2) and is associated with an ACL (ACL1) that restricts access to V1 to DM A only. Similarly, DM B defines a variable (V2) whose value is given by the mathematical expression (EDD2 * 2) and is associated with an ACL (ACL2) that restricts access to V2 to DM B only.

Both DM A and DM B also send policies to DA A to report on the values of their variables at 1-second intervals (step 2). Since DM A can access V1 and DM B can access V2, there is no authorization issue with these policies, and they are both accepted by the autonomy engine on DA A. DA A produces reports as expected, sending them to their respective managers (step 3).

Later (step 4), DM B attempts to configure DA A to also report to it the value of V1. Since DM B does not have authorization to view this variable, DA A does not include this in the configuration of its autonomy engine; instead, some indication of a permission error is included in any regular reporting back to DM B.

DM A also sends a policy to DA A (step 5) that defines a variable (V3) whose value is given by the mathematical expression (EDD3 * 3) and is not associated with an ACL, indicating that any DM can access V3. In this instance, both DM A and DM B can then send policies to DA A to report the value of V3 (step 6). Since there is no authorization restriction on V3, these policies are accepted by the autonomy engine on DA A, and reports are sent to both DM A and DM B over time (step 7).

10.6. Cascading Management

There are times when a single network device may serve as both a DM for other DAs in the network and, itself, as a device managed by someone else. This may be the case on nodes serving as gateways or proxies. The DTNMA accommodates this case by allowing a single device to run both a DA and a DM.

An example of this configuration is illustrated in Figure 8.

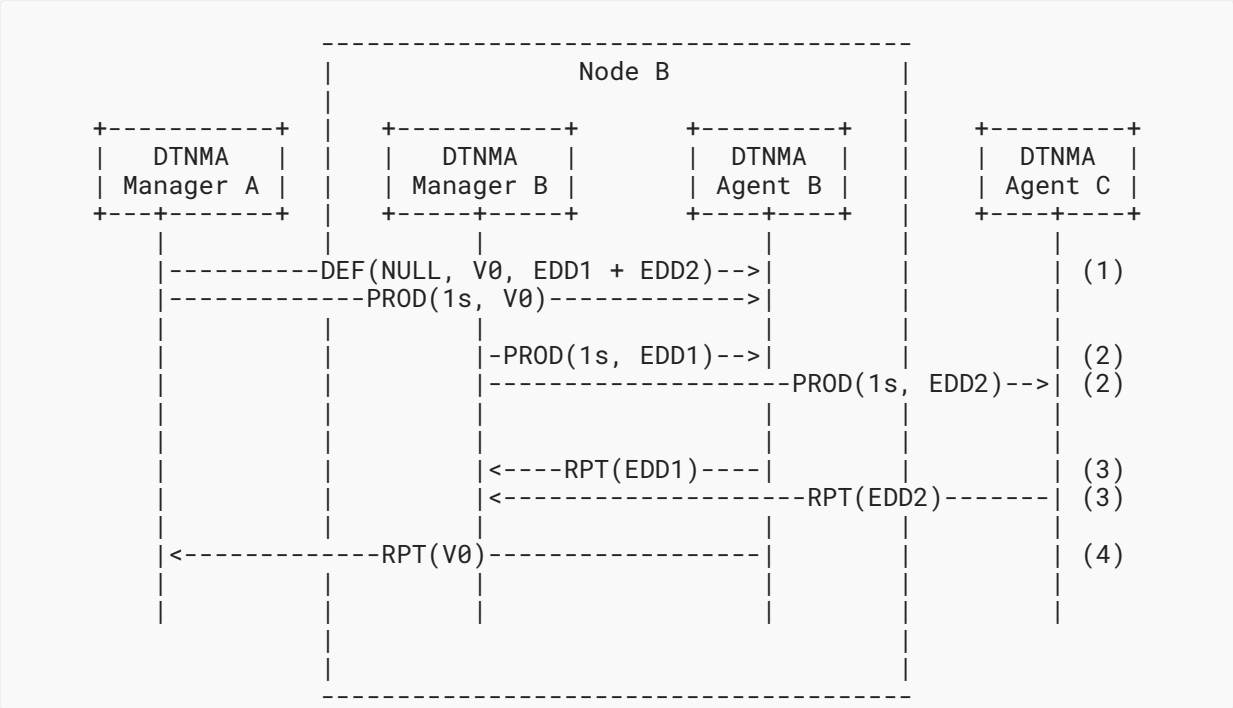

Figure 8: Cascading Management Control Flow

A device can operate as both a DM and a DA.

In this example, we presume that DA B is able to sample a given EDD (EDD1) and that DA C is able to sample a different EDD (EDD2). Node B houses DM B (which controls DA C) and DA B (which is controlled by DM A). DM A must periodically receive some new value that is calculated as a function of both EDD1 and EDD2.

First, DM A sends a policy to DA B to define a variable (V0) whose value is given by the mathematical expression (EDD1 + EDD2) without a restricting ACL. Further, DM A sends a policy to DA B to report on the value of V0 every second (step 1).

DA B needs the ability to monitor both EDD1 and EDD2 to produce V0. DA B is able to sample EDD1, so DM B sends a policy to DA B to report on the value of EDD1. However, the only way to receive EDD2 values is to have them reported back to Node B by DA C and included in the Node B runtime datastores. Therefore, DM B also sends a policy to DA C to report on the value of EDD2 (step 2).

DA B receives the policy in its autonomy engine and produces reports on the value of EDD2 every second. Similarly, DA C receives the policy in its autonomy engine and produces reports on the value of EDD2 every second (step 3).

DA B may locally sample EDD1 and EDD2 and uses that to compute values of V0 and report on those values at regular intervals to DM A (step 4).

While a trivial example, the mechanism of associating fusion with the DA function rather than the DM function scales with fusion complexity. Within the DTNMA, DAs and DMs are not required to be separate software implementations. There may be a single software application running on Node B implementing both DM B and DA B roles.

11. IANA Considerations

This document has no IANA actions.

12. Security Considerations

Security within a DTNMA exists in at least the following two layers: security in the data model and security in the messaging and encoding of the data model.

Data model security refers to the validity and accessibility of data elements. For example, a data element might be available to certain DAs or DMs in a system, whereas the same data element may be hidden from other DAs or DMs. Both verification and authorization mechanisms at DAs and DMs are important to achieve this type of security.

> NOTE: One way to provide finer-grained application security is through the use of ACLs that would be defined as part of the configuration of DAs and DMs. It is expected that many common data model tools provide mechanisms for the definition of ACLs and best practices for their operational use.

The exchange of information between and amongst DAs and DMs in the DTNMA is expected to be accomplished through some secured messaging transport.

13. Informative References

[ASN.1] ITU-T, "Information technology - Abstract Syntax Notation One (ASN.1): Specification of basic notation", ITU-T Recommendation X.680, ISO/IEC 8824-1:2021, February 2021, <https://www.itu.int/rec/T-REC-X.680>.

[CORE-COMI] Veillette, M., Ed., van der Stok, P., Ed., Pelov, A., Ed., Bierman, A., and C. Bormann, Ed., "CoAP Management Interface (CORECONF)", Work in Progress, Internet-Draft, draft-ietf-core-comi-18, 23 July 2024, <https://datatracker.ietf.org/doc/html/draft-ietf-core-comi-18>.

[DART] Tropf, B. T., Haque, M., Behrooz, N., and C. Krupiarz, "The DART Autonomy System", DOI 10.1109/SMC-IT56444.2023.00020, August 2023, <https://ieeexplore.ieee.org/abstract/document/10207457>.

[gNMI] Borman, P., Hines, M., Lebsack, C., Morrow, C., Shaikh, A., Shakir, R., Li, W., and D. Loher, "gRPC Network Management Interface (gNMI)", Version 10.0, May 2023, <https://www.openconfig.net/docs/gnmi/gnmi-specification/>.

[gRPC] gRPC Authors, "gRPC Documentation", 2024, <https://grpc.io/docs/>.

[IPMI] Intel, Hewlett-Packard, NEC, and Dell, "Intelligent Platform Management Interface Specification, Second Generation", Version 2.0, October 2013, <https://www.intel.la/content/dam/www/public/us/en/documents/specification-updates/ipmi-intelligent-platform-mgt-interface-spec-2nd-gen-v2-0-spec-update.pdf>.

[NEW-HORIZONS] Moore, R. C., "Autonomous safeing and fault protection for the New Horizons mission to Pluto", Acta Astronautica, Volume 61, Issues 1-6, June-August 2007, Pages 398-405, DOI 10.1016/j.actaastro.2007.01.009, August 2007, <https://www.sciencedirect.com/science/article/pii/S0094576507000604>.

[PROTOCOL-BUFFERS] Stuart, S. and R. Fernando, "Encoding rules and MIME type for Protocol Buffers", Work in Progress, Internet-Draft, draft-rfernando-protocol-buffers-00, 8 October 2012, <https://datatracker.ietf.org/doc/html/draft-rfernando-protocol-buffers-00>.

[RFC2578] McCloghrie, K., Ed., Perkins, D., Ed., and J. Schoenwaelder, Ed., "Structure of Management Information Version 2 (SMIv2)", STD 58, RFC 2578, DOI 10.17487/RFC2578, April 1999, <https://www.rfc-editor.org/info/rfc2578>.

[RFC2982] Kavasseri, R., Ed., "Distributed Management Expression MIB", RFC 2982, DOI 10.17487/RFC2982, October 2000, <https://www.rfc-editor.org/info/rfc2982>.

[RFC3165] Levi, D. and J. Schoenwaelder, "Definitions of Managed Objects for the Delegation of Management Scripts", RFC 3165, DOI 10.17487/RFC3165, August 2001, <https://www.rfc-editor.org/info/rfc3165>.

[RFC3410] Case, J., Mundy, R., Partain, D., and B. Stewart, "Introduction and Applicability Statements for Internet-Standard Management Framework", RFC 3410, DOI 10.17487/RFC3410, December 2002, <https://www.rfc-editor.org/info/rfc3410>.

[RFC3411] Harrington, D., Presuhn, R., and B. Wijnen, "An Architecture for Describing Simple Network Management Protocol (SNMP) Management Frameworks", STD 62, RFC 3411, DOI 10.17487/RFC3411, December 2002, <https://www.rfc-editor.org/info/rfc3411>.

[RFC3414] Blumenthal, U. and B. Wijnen, "User-based Security Model (USM) for version 3 of the Simple Network Management Protocol (SNMPv3)", STD 62, RFC 3414, DOI 10.17487/RFC3414, December 2002, <https://www.rfc-editor.org/info/rfc3414>.

[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, <https://www.rfc-editor.org/info/rfc3416>.

[RFC3417] Presuhn, R., Ed., "Transport Mappings for the Simple Network Management Protocol (SNMP)", STD 62, RFC 3417, DOI 10.17487/RFC3417, December 2002, <https://www.rfc-editor.org/info/rfc3417>.

[RFC3418] Presuhn, R., Ed., "Management Information Base (MIB) for the Simple Network Management Protocol (SNMP)", STD 62, RFC 3418, DOI 10.17487/RFC3418, December 2002, <https://www.rfc-editor.org/info/rfc3418>.

[RFC4838] Cerf, V., Burleigh, S., Hooke, A., Torgerson, L., Durst, R., Scott, K., Fall, K., and H. Weiss, "Delay-Tolerant Networking Architecture", RFC 4838, DOI 10.17487/RFC4838, April 2007, <https://www.rfc-editor.org/info/rfc4838>.

[RFC4949] Shirey, R., "Internet Security Glossary, Version 2", FYI 36, RFC 4949, DOI 10.17487/RFC4949, August 2007, <https://www.rfc-editor.org/info/rfc4949>.

[RFC5591] Harrington, D. and W. Hardaker, "Transport Security Model for the Simple Network Management Protocol (SNMP)", STD 78, RFC 5591, DOI 10.17487/RFC5591, June 2009, <https://www.rfc-editor.org/info/rfc5591>.

[RFC5592] Harrington, D., Salowey, J., and W. Hardaker, "Secure Shell Transport Model for the Simple Network Management Protocol (SNMP)", RFC 5592, DOI 10.17487/RFC5592, June 2009, <https://www.rfc-editor.org/info/rfc5592>.

[RFC5652] Housley, R., "Cryptographic Message Syntax (CMS)", STD 70, RFC 5652, DOI 10.17487/RFC5652, September 2009, <https://www.rfc-editor.org/info/rfc5652>.

[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, <https://www.rfc-editor.org/info/rfc6241>.

[RFC6242] Wasserman, M., "Using the NETCONF Protocol over Secure Shell (SSH)", RFC 6242, DOI 10.17487/RFC6242, June 2011, <https://www.rfc-editor.org/info/rfc6242>.

[RFC6353] Hardaker, W., "Transport Layer Security (TLS) Transport Model for the Simple Network Management Protocol (SNMP)", STD 78, RFC 6353, DOI 10.17487/RFC6353, July 2011, <https://www.rfc-editor.org/info/rfc6353>.

[RFC6991] Schoenwaelder, J., Ed., "Common YANG Data Types", RFC 6991, DOI 10.17487/RFC6991, July 2013, <https://www.rfc-editor.org/info/rfc6991>.

[RFC7228] Bormann, C., Ersue, M., and A. Keranen, "Terminology for Constrained-Node Networks", RFC 7228, DOI 10.17487/RFC7228, May 2014, <https://www.rfc-editor.org/info/rfc7228>.

[RFC7252] Shelby, Z., Hartke, K., and C. Bormann, "The Constrained Application Protocol (CoAP)", RFC 7252, DOI 10.17487/RFC7252, June 2014, <https://www.rfc-editor.org/info/rfc7252>.

[RFC7575] Behringer, M., Pritikin, M., Bjarnason, S., Clemm, A., Carpenter, B., Jiang, S., and L. Ciavaglia, "Autonomic Networking: Definitions and Design Goals", RFC 7575, DOI 10.17487/RFC7575, June 2015, <https://www.rfc-editor.org/info/rfc7575>.

[RFC7589] Badra, M., Luchuk, A., and J. Schoenwaelder, "Using the NETCONF Protocol over Transport Layer Security (TLS) with Mutual X.509 Authentication", RFC 7589, DOI 10.17487/RFC7589, June 2015, <https://www.rfc-editor.org/info/rfc7589>.

[RFC7950] Bjorklund, M., Ed., "The YANG 1.1 Data Modeling Language", RFC 7950, DOI 10.17487/RFC7950, August 2016, <https://www.rfc-editor.org/info/rfc7950>.

[RFC7951] Lhotka, L., "JSON Encoding of Data Modeled with YANG", RFC 7951, DOI 10.17487/RFC7951, August 2016, <https://www.rfc-editor.org/info/rfc7951>.

[RFC8040] Bierman, A., Bjorklund, M., and K. Watsen, "RESTCONF Protocol", RFC 8040, DOI 10.17487/RFC8040, January 2017, <https://www.rfc-editor.org/info/rfc8040>.

[RFC8199] Bogdanovic, D., Claise, B., and C. Moberg, "YANG Module Classification", RFC 8199, DOI 10.17487/RFC8199, July 2017, <https://www.rfc-editor.org/info/rfc8199>.

[RFC8294] Liu, X., Qu, Y., Lindem, A., Hopps, C., and L. Berger, "Common YANG Data Types for the Routing Area", RFC 8294, DOI 10.17487/RFC8294, December 2017, <https://www.rfc-editor.org/info/rfc8294>.

[RFC8341] Bierman, A. and M. Bjorklund, "Network Configuration Access Control Model", STD 91, RFC 8341, DOI 10.17487/RFC8341, March 2018, <https://www.rfc-editor.org/info/rfc8341>.

[RFC8342] Bjorklund, M., Schoenwaelder, J., Shafer, P., Watsen, K., and R. Wilton, "Network Management Datastore Architecture (NMDA)", RFC 8342, DOI 10.17487/RFC8342, March 2018, <https://www.rfc-editor.org/info/rfc8342>.

[RFC8368] Eckert, T., Ed. and M. Behringer, "Using an Autonomic Control Plane for Stable Connectivity of Network Operations, Administration, and Maintenance (OAM)", RFC 8368, DOI 10.17487/RFC8368, May 2018, <https://www.rfc-editor.org/info/rfc8368>.

[RFC8639] Voit, E., Clemm, A., Gonzalez Prieto, A., Nilsen-Nygaard, E., and A. Tripathy, "Subscription to YANG Notifications", RFC 8639, DOI 10.17487/RFC8639, September 2019, <https://www.rfc-editor.org/info/rfc8639>.

[RFC8641] Clemm, A. and E. Voit, "Subscription to YANG Notifications for Datastore Updates", RFC 8641, DOI 10.17487/RFC8641, September 2019, <https://www.rfc-editor.org/info/rfc8641>.

[RFC8990] Bormann, C., Carpenter, B., Ed., and B. Liu, Ed., "GeneRic Autonomic Signaling Protocol (GRASP)", RFC 8990, DOI 10.17487/RFC8990, May 2021, <https://www.rfc-editor.org/info/rfc8990>.

[RFC8993] Behringer, M., Ed., Carpenter, B., Eckert, T., Ciavaglia, L., and J. Nobre, "A Reference Model for Autonomic Networking", RFC 8993, DOI 10.17487/RFC8993, May 2021, <https://www.rfc-editor.org/info/rfc8993>.

[RFC9113] Thomson, M., Ed. and C. Benfield, Ed., "HTTP/2", RFC 9113, DOI 10.17487/RFC9113, June 2022, <https://www.rfc-editor.org/info/rfc9113>.

[RFC9171] Burleigh, S., Fall, K., and E. Birrane, III, "Bundle Protocol Version 7", RFC 9171, DOI 10.17487/RFC9171, January 2022, <https://www.rfc-editor.org/info/rfc9171>.

[RFC9172] Birrane, III, E. and K. McKeever, "Bundle Protocol Security (BPSec)", RFC 9172, DOI 10.17487/RFC9172, January 2022, <https://www.rfc-editor.org/info/rfc9172>.

[RFC9254] Veillette, M., Ed., Petrov, I., Ed., Pelov, A., Bormann, C., and M. Richardson, "Encoding of Data Modeled with YANG in the Concise Binary Object Representation (CBOR)", RFC 9254, DOI 10.17487/RFC9254, July 2022, <https://www.rfc-editor.org/info/rfc9254>.

[RFC9595] Veillette, M., Ed., Pelov, A., Ed., Petrov, I., Ed., Bormann, C., and M. Richardson, "YANG Schema Item iDentifier (YANG SID)", RFC 9595, DOI 10.17487/RFC9595, July 2024, <https://www.rfc-editor.org/info/rfc9595>.

[xml-infoset] Cowan, J., Ed. and R. Tobin, Ed., "XML Information Set (Second Edition)", W3C Recommendation REC-xml-infoset-20040204, February 2004, <https://www.w3.org/TR/2004/REC-xml-infoset-20040204/>.

[XPath] Clark, J., Ed. and S. DeRose, Ed., "XML Path Language (XPath) Version 1.0", W3C Recommendation REC-xpath-19991116, November 1999, <https://www.w3.org/TR/1999/REC-xpath-19991116>.

Acknowledgements

Brian Sipos of the Johns Hopkins University Applied Physics Laboratory (JHU/APL) provided excellent technical review of the DTNMA concepts presented in this document and additional information related to existing network management techniques.

Authors' Addresses

Edward J. Birrane, III
The Johns Hopkins University Applied Physics Laboratory
Email: Edward.Birrane@jhuapl.edu

Sarah E. Heiner
The Johns Hopkins University Applied Physics Laboratory
Email: Sarah.Heiner@jhuapl.edu

Emery Annis
The Johns Hopkins University Applied Physics Laboratory
Email: Emery.Annis@jhuapl.edu