Sparkplug 3.0.0-SNAPSHOT

Sparkplug Specification

Eclipse Sparkplug Contributors

Version 3.0.0-SNAPSHOT, 2022-01-11

Table of Contents

Sparkplug
MQTT Topic &
Payload Definition

Revision Number	Date	Author	Description
1.0	5/26/16	Cirrus Link	Initial Release
2.1	12/10/16	Cirrus Link	Payload B Addition
2.2	10/11/19	Cirrus Link	Re-branding for Eclipse foundation added TM to Sparkplug
3.0	11/1/20	Eclipse Sparkplug Specification Project Team	Reorganized to be in adoc format and to include normative and non-normative statements

Chapter 1. Introduction

1.1. Rationale and Use Case

Eclipse Sparkplug™ provides an open and freely available specification for how Edge of Network Gateways (Sparkplug Edge Nodes) or native MQTT enabled end devices and Sparkplug Host Applications communicate bi-directionally within an MQTT Infrastructure. This document details the structure and implementation requirements for Sparkplug compliant MQTT Client implementations on both Edge Nodes and Host Applications.

It is recognized that MQTT is used across a wide spectrum of application solution use-cases, and an almost indefinable variation of network topologies. To that end the Sparkplug Specification strives to accomplish the three following goals.

1.1.1. Define an MQTT Topic Namespace

As noted many times in this document one of the many attractive features of MQTT is that is does not specify any required MQTT Topic Namespace within its implementation. This fact has meant that MQTT has taken a dominant position across a wide spectrum of IoT solutions. The intent of the Sparkplug Specification is to identify and document a Topic Namespace that is well thought out and optimized for the SCADA/IIoT solution sector. In addition, Sparkplug defines a Topic Namespace in such away that it provides semantics which allow for automatic discovery and bi-directional communication between MQTT clients in a system.

1.1.2. Define MQTT State Management

One of the unique aspects of MQTT is that it was originally designed for real time SCADA systems to help reduce data latency over bandwidth limited and often unreliable network infrastructures. In many implementations the full benefit of this “Continuous Session Awareness” is not well understood, or not even implemented. The intent of the Sparkplug Specification is to take full advantage of MQTT’s native Continuous Session Awareness capability as it applies to real time SCADA/IIoT solutions.

1.1.3. Define the MQTT Payload

Just as the MQTT Specification does not dictate any particular Topic Namespace, it also does not dictate any particular payload data encoding. The intent of the Sparkplug Specification is to define payload encoding mechanisms that remain true to the original, lightweight, bandwidth efficient, low latency features of MQTT while adding modern encoding schemes targeting the SCADA/IIoT solution space.

Sparkplug has defined an approach where the Topic Namespace can aid in the determination of the encoding scheme of any particular payload. Historically there have been two Sparkplug defined encoding schemes. The first one was the *Sparkplug A* and the second is *Sparkplug B*. Each of these uses a *first topic token identifier* so Sparkplug Edge Nodes can declare the payload encoding scheme they are using. These first topic tokens are:

```
spAv1.0
spBv1.0
```

Each token is divided up into three distinct components. These are:

- Sparkplug Identifier
  - Always *sp*
- Payload Encoding Scheme
  - Currently *A* or *B* but there could be future versions
- Payload Encoding Scheme Version
  - Currently v1.0 but denoted in the event that future versions are released

The original *Sparkplug A* encoding scheme was based on the Eclipse Kura™ open source Google Protocol Buffer definition. *Sparkplug B* was released shortly after the release of Sparkplug A and addressed a number of issues that were present in the A version of the payload encoding scheme. Due to lack of adoption and the fact that *Sparkplug B* was made available shortly after the release of *A*, the Sparkplug A definition has been omitted from this document and is no longer supported.

The *Sparkplug B* encoding scheme was created with a richer data model developed with the feedback of many system integrators and end user customers using MQTT. These additions included metric timestamp support, complex datatype support, metadata, and other improvements.

1.1.4. Background

MQTT was originally designed as a message transport for real-time SCADA systems. The MQTT Specification does not specify the Topic Namespace nor does it define the Payload representation of the data being published and/or subscribed to. In addition to this, since the original use-case for MQTT was targeting real-time SCADA, there are mechanisms defined to provide the state of an MQTT session such that SCADA/Control Human-Machine Interface (HMI) application can monitor the current state of any MQTT enabled device in the infrastructure. As with the Topic Namespace and Payload the way state information is implemented and managed within the MQTT infrastructure is not defined. All of this was intentional within the original MQTT Specification to provide maximum flexibility across any solution sector that might choose to use MQTT infrastructures.

But at some point, for MQTT based solutions to be interoperable within a given market sector, the Topic Namespace, Payload representation, and session state must be defined. The intent and purpose of the Sparkplug Specification is to define an MQTT Topic Namespace, payload, and session state management that can be applied generically to the overall IIoT market sector, but specifically meets the requirements of real-time SCADA/Control HMI solutions. Meeting the operational requirements for these systems will enable MQTT based infrastructures to provide more valuable real-time information to Line of Business and MES solution requirements as well.

The purpose of the Sparkplug Specification is to remain true to the original notion of keeping the Topic Namespace and message sizes to a minimum while still making the overall message transactions and session state management between MQTT enabled devices and MQTT SCADA/IIoT

applications simple, efficient, easy to understand, and implement.

1.2. Intellectual Property Rights

1.2.1. Eclipse Foundation Specification License

By using and/or copying this document, or the Eclipse Foundation document from which this statement is linked, you (the licensee) agree that you have read, understood, and will comply with the following terms and conditions:

Permission to copy, and distribute the contents of this document, or the Eclipse Foundation document from which this statement is linked, in any medium for any purpose and without fee or royalty is hereby granted, provided that you include the following on ALL copies of the document, or portions thereof, that you use:

- link or URL to the original Eclipse Foundation document.
- All existing copyright notices, or if one does not exist, a notice (hypertext is preferred, but a textual representation is permitted) of the form: "Copyright © [$date-of-document] "Eclipse Foundation, Inc. <<url to this license>> "

Inclusion of the full text of this NOTICE must be provided. We request that authorship attribution be provided in any software, documents, or other items or products that you create pursuant to the implementation of the contents of this document, or any portion thereof.

No right to create modifications or derivatives of Eclipse Foundation documents is granted pursuant to this license, except anyone may prepare and distribute derivative works and portions of this document in software that implements the specification, in supporting materials accompanying such software, and in documentation of such software, PROVIDED that all such works include the notice below. HOWEVER, the publication of derivative works of this document for use as a technical specification is expressly prohibited.

The notice is:

"Copyright (c) 2016-2021 Eclipse Foundation. This software or document includes material copied from or derived from the Sparkplug Specification: https://www.eclipse.org/tahu/spec/Sparkplug%20Topic%20Namespace%20and%20State%20ManagementV2.2-with%20appendix%20B%20format%20-%20Eclipse.pdf

1.2.2. Disclaimers

THIS DOCUMENT IS PROVIDED "AS IS," AND THE COPYRIGHT HOLDERS AND THE ECLIPSE FOUNDATION MAKE NO REPRESENTATIONS OR WARRANTIES, EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, NON-INFRINGEMENT, OR TITLE; THAT THE CONTENTS OF THE DOCUMENT ARE SUITABLE FOR ANY PURPOSE; NOR THAT THE IMPLEMENTATION OF SUCH CONTENTS WILL NOT INFRINGE ANY THIRD PARTY PATENTS, COPYRIGHTS, TRADEMARKS OR OTHER RIGHTS.

THE COPYRIGHT HOLDERS AND THE ECLIPSE FOUNDATION WILL NOT BE LIABLE FOR ANY DIRECT, INDIRECT, SPECIAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF ANY USE OF THE

DOCUMENT OR THE PERFORMANCE OR IMPLEMENTATION OF THE CONTENTS THEREOF.

The name and trademarks of the copyright holders or the Eclipse Foundation may NOT be used in advertising or publicity pertaining to this document or its contents without specific, written prior permission. Title to copyright in this document will at all times remain with copyright holders.

1.3. Organization of the Sparkplug Specification

This specification is split into the following chapters and appendices:

1.4. Terminology

1.4.1. Infrastructure Components

This section details the infrastructure components implemented.

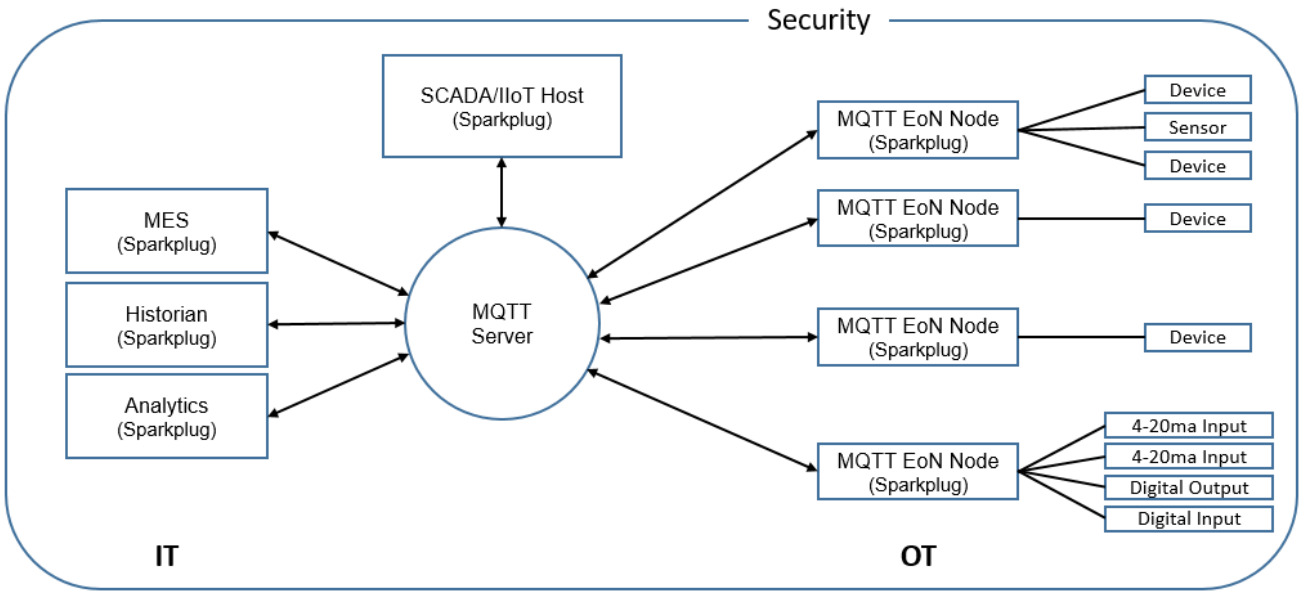

Figure 1 - MQTT SCADA Infrastructure

MQTT Server(s)

MQTT enabled infrastructure requires that one or more MQTT Servers are present in the infrastructure. An MQTT Server must be compatible with the requirements outlined in the Conformance Section. In addition, it must be sized to properly manage all MQTT message traffic.

One can implement the use (if required) of multiple MQTT servers for redundancy, high availability, and scalability within any given infrastructure.

Sparkplug Group

A *Sparkplug Group* is a logical or physical group of Edge Nodes that makes sense in the context of a distributed Sparkplug application. Groups can represent physical groups of Edge Nodes. For example, a Sparkplug Group could represent a set of Edge Nodes at a particular location, facility, or along a specific oil pipeline. Alternatively, a Sparkplug Group could represent group of similar types of Edge Nodes. For example, it could represent a particular set of like make and models of embedded gateways. The groups are meant to be defined by the system architects as appropriate for their particular application.

Sparkplug Edge Node

In the context of this specification, a Sparkplug Edge Node is any v3.1.1 or v5.0 compliant MQTT Client application that manages an MQTT Session and provides the physical and/or logical gateway functions required to participate in the Topic Namespace and Payload definitions described in this document. The Edge Node is responsible for any local protocol interface to existing devices (PLCs, RTUs, Flow Computers, Sensors, etc.) and/or any local discrete I/O, and/or any logical internal process variables (PVs).

Sparkplug Device

A Sparkplug Device represents a physical or logical device that makes sense in the context of a distributed Sparkplug application. Often times a Sparkplug Device will be a physical PLC, RTU, Flow

Computer, Sensor, etc. However, a Sparkplug device could also represent a logical grouping of data points as makes sense for the specific Sparkplug Application being developed. For example, it could represent a set of data points across multiple PLCs that make up a logical device that makes sense within the context of that application.

MQTT/Sparkplug Enabled Device

This represents any device, sensor, or hardware that directly connects to MQTT infrastructure using a compliant MQTT v3.1.1 or v5.0 connection with the payload and topic notation as outlined in this Sparkplug Specification. With MQTT/Sparkplug enabled directly in the device this could bypass the use of a Sparkplug Device in the infrastructure. In this case, the physical device or sensor is the Edge Node. It is up to the developer of the application to decide if the concept of a *Sparkplug Device* is to be used within their application.

Host Applications

A Host Application is defined as an application that consumes data from Sparkplug Edge Nodes. Depending on the nature of the Host Application it may consume Edge Node data and display it in a dashboard, it may historize the data in a database, or it may analyze the data in some way. SCADA/IIoT Hosts, MES, Historians, and Analytics applications are all examples of potential Sparkplug Host Applications. A Host Application may perform many different functions in handling the data. In addition, Host Applications may also send Sparkplug NCMD or DCMD messages to Edge Nodes.

A Sparkplug Edge Node may specify one Host Application as its *Primary Host Application*. This is handled by the Edge Node waiting to publish its NBIRTH and DBIRTH messages until the Host Application that the Edge Node has designated as its Primary Host application has come online. Sparkplug does not support the notion of multiple Primary Host Applications. This does not preclude any number of additional Host Applications participating in the infrastructure that are in either a pure monitoring mode, or in the role of a hot standby should the Edge Node's Primary Host Application go offline or become unavailable within the infrastructure.

Sparkplug Host Applications MUST publish STATE messages denoting their online and offline status.

Primary Host Application

A Primary Host Applications may be defined by an Edge Node. The Edge Node's behavior may change based on the status of its configured Primary Host. It is not required that an Edge Node must have a Primary Host configured but it may be useful in certain applications. This allows Edge Nodes to make decisions based on whether or not the Primary Host Application is online or not. For example, an Edge Node may store data at the edge until a Primary Host Application comes back online. When the Primary Host Application publishes a new STATE message denoting it is online, the Edge Node can resume publishing data and also flush any historical data that it may have stored while offline.

In a traditional SCADA system the SCADA Host would be the Primary Host Application. It is the most important consumer of data to keep operations running. With this same concept in mind, there can only be one Primary Host Application configured in an Edge Node as a result.

Sparkplug Identifiers

Sparkplug defines identifiers or IDs for different physical or logical components within the infrastructure. There are three primary IDs and one that is a composite ID. These are defined as the following.

- Group ID
  - **The Group ID MUST be UTF-8 string and used as part of the Sparkplug topics as defined in the Topics Section.**
  - **Because the Group ID is used in MQTT topic strings the Group ID MUST only contain characters allowed for MQTT topics per the MQTT Specification.**
  - Non-normative comment: The Group ID represents a general grouping of Edge Nodes that makes sense within the context of the Sparkplug application and use-case.
- Edge Node ID
  - **The Edge Node ID MUST be UTF-8 string and used as part of the Sparkplug topics as defined in the Topics Section.**
  - **Because the Edge Node ID is used in MQTT topic strings the Group ID MUST only contain characters allowed for MQTT topics per the MQTT Specification.**
  - Non-normative comment: The Edge Node ID represents a unique identifier for an Edge Node within the context of the Group ID under which it exists.
- Device ID
  - **The Device ID MUST be UTF-8 string and used as part of the Sparkplug topics as defined in the Topics Section.**
  - **Because the Device ID is used in MQTT topic strings the Group ID MUST only contain characters allowed for MQTT topics per the MQTT Specification.**
  - Non-normative comment: The Device ID represents a unique identifier for a Device within the context of the Edge Node ID under which it exists.
- Edge Node Descriptor (composite ID)
  - The Edge Node Descriptor is the combination of the Group ID and Edge Node ID.
  - **The Edge Node Descriptor MUST be unique within the context of all of other Edge Nodes within the Sparkplug infrastructure.** In other words, no two Edge Nodes within a Sparkplug environment can have the same Group ID and same Edge Node ID.
  - Non-normative comment: The Device ID represents a unique identifier for a Device within the context of the Edge Node ID under which it exists.

Sparkplug Metric

A Sparkplug Metric is the term used for a single *tag change event* in the Sparkplug Payload. It represents an event that occurred at the Edge Node or Device such as a value or quality of a data point changing. For example, it could represent the value of an analog or boolean changing at a Sparkplug Device. A Sparkplug Metric typically includes a name, value, and timestamp. Sparkplug Metrics are also used in NCMD and DCMD messages to send messages to Edge Nodes and Devices to change values at the Edge.

1.5. Normative References

A list of all normative statements made in the Sparkplug specification document can be found in Appendix B.

1.6. Security

Security is not directly addressed in the Sparkplug Specification with normative statements. However, security should be addressed appropriately in every Sparkplug system. MQTT clients, servers, authentication, authorization, network access, physical access, and all other aspects of security should be addressed based on how the system will be deployed and used. Because Sparkplug utilizes MQTT and TCP/IP, the security features and best practices of those protocols also applies to Sparkplug. The security practices related to TCP/IP and MQTT have changed throughout the years and likely will continue to do so. As a result, the Sparkplug Specification will defer to the underlying protocols and industry standards for best practices. However, some non-normative statements are included with regard to security in the Sparkplug Specification.

1.6.1. Authentication

There are several levels of security and access control configured within an MQTT infrastructure. From a pure MQTT client perspective, the client must provide a unique MQTT Client ID, and an optional MQTT username and password.

1.6.2. Authorization

Although access control is not mandated in the MQTT Specification for use in MQTT Server implementations, Access Control List (ACL) functionality is available in many MQTT Server implementations. The ACL of an MQTT Server implementation is used to specify which Topic Namespace any MQTT Client can subscribe to and publish on. For example, it may make sense to have an Edge Node's MQTT client only able to publish on topics associated with it's Group and Edge Node ID. This would make it difficult for an MQTT client to spoof another Edge Node whether it be malicious or a configuration setup error.

1.6.3. Encryption

The MQTT Specification does not specify any TCP/IP security scheme as it was envisaged during development of the MQTT Specification that TCP/IP security would (and did) change over time. Although this document will not specify any TCP/IP security schema it will provide examples on how to secure an MQTT infrastructure using TLS security.

1.7. Editing Convention

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119. RFC 2119: https://tools.ietf.org/html/rfc2119

All normative statements in this document are highlighted in **yellow text as shown here**.

1.8. Leveraging Standards and Open Source

In addition to leveraging MQTT v3.1.1 and MQTT v5.0 standards, the Sparkplug Specification leverages as much open source development tooling and data encoding as possible. Many different open source organizations, projects, and ideas were used in the development of the Sparkplug Specification. More information on these can be found in Appendix A

Chapter 2. Principles

2.1. Pub/Sub

This section discusses the simple topology shown in "Figure 2 – Simple MQTT Infrastructure" identifying how each of the components of the infrastructure interacts.

At the simplest level, there are only two components required as shown below. An MQTT client and an MQTT server are the primary two components. With proper credentials, any MQTT client can connect to the MQTT server without any notion of other MQTT client applications that are connected. The client can issue subscriptions to any MQTT messages that it might be interested in as well as start publishing any message containing data that it has. This is one of the principal notions of IIoT, that is the decoupling of intelligent devices from any direct connection to any one consumer application.

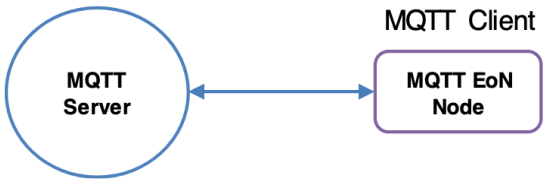

Figure 2 - Simple MQTT Infrastructure

2.2. Report by Exception

The Sparkplug Specification uses the concept of Report by Exception (RBE). Because Sparkplug utilizes the built in functions of MQTT to maintain session awareness, messages only need to be sent by an Edge Node when values at the edge change. In the initial BIRTH messages, all of the current metric values are published in the payload. Because the MQTT session is stateful, after the initial BIRTH messages are sent, new metric values only need to be published when the values change.

Sparkplug does not require that RBE be used in all cases. This is to account for special circumstances that may require periodic reporting. However, as a general rule periodic publishing should not be used.

Because of the stateful nature of Sparkplug sessions, data SHOULD NOT be published from Edge Nodes on a periodic basis and instead SHOULD be published using a RBE based approach.

2.3. Continuous Session Awareness

In any network architecture, network connection "State" is important. In SCADA/IIoT, connection State is extremely important. State is the session awareness of the MQTT Edge Node and the MQTT Server. The very reason that most SCADA Host systems in this market sector are still using legacy poll/response protocols to maintain a notion of the State of the connection between the SCADA application and the connected devices. *"I poll, I get a response, I know the State of all the I/O points, but now I must poll again because that State may have changed."*

Many implementations of solutions using MQTT treat it as a simple, stateless, pub/sub state machine. This is quite viable for IoT and some IIoT applications, however it is not taking advantage of the full capability of MQTT based infrastructures.

One of the primary applications for MQTT as it was originally designed was to provide reliable SCADA communications over VSAT networks. Due to propagation delay and cost, it was not feasible to use a poll/response protocol. Instead of a poll/response protocol where all the data was sent in response to every poll, MQTT was used to publish information from remote sites only when the data changed. This technique is sometimes called Report by Exception or RBE. But for RBE to work properly in real-time SCADA, the "state" of the end device needs to be always known. In other words, SCADA/IIoT host could only rely on RBE data arriving reliably if it could be assured of the state of the MQTT session.

The Eclipse Sparkplug™ specification defines the use of the MQTT "Will Message" feature to provide MQTT session state information to any other interested MQTT client in the infrastructure. The session state awareness is implemented around a set of defined BIRTH and DEATH topic namespace and payload definitions in conjunction with the MQTT connection "Keep Alive" timer.

2.4. Birth and Death Certificates

Birth and Death Certificates are used by both Edge Nodes and Host Applications. Death Certificates for both are always registered in the MQTT CONNECT packet as the MQTT Will Message. By using the MQTT Will message, the Death Certificates will be delivered to subscribers even if the MQTT client connection is lost ungracefully. For Edge Nodes, the Death Certificate uses the NDEATH Sparkplug verb in the topic. For Host Applications, the STATE/host_id topic is used. More information on Death certificates can be found in Edge Node Death Certificates and Host Application Death Certificates

Birth Certificates MUST be the first MQTT messages published by any Edge Node or any Host Application. They denote to any subscribing MQTT clients that they are now online. For Edge Nodes, the Birth Certificate uses the NBIRTH Sparkplug verb in the topic. For Host Applications, the STATE/host_id topic is used. More details and requirements on Birth certificates can be found in Edge Node Birth Certificates and Host Application Birth Certificates

2.5. Persistent vs Non-Persistent Connections for Edge Nodes

Persistent connections are intended to remain connected to the MQTT infrastructure at all times. They never send an MQTT DISCONNECT control packet during normal operation. This fact lets the Host Applications provide the real-time state of every persistent node in the infrastructure within the configured MQTT Keep Alive period using the BIRTH/DEATH mechanisms defined above.

But in some use cases, such as sending GPS coordinates for asset tracking or other IOT applications with periodic data from sensors, MQTT enabled devices do not need to remain connected to the MQTT infrastructure. In these use cases, all the Device needs to do is to issue an MQTT DISCONNECT control packet prior to going offline to leave the MQTT infrastructure "gracefully". In this case an MQTT device or associated DEATH certificate will most normally not be seen. System designers just

need to be aware that the metric in the Host Application will represent “Last Known Good” values with a timestamp of this data where the current state of the of the MQTT Device is not a real-time indication. The Host Application metric timestamp values can be used to determine when the values from this node were last updated.

Non-persistent MQTT Enabled Devices should still register a proper DEATH Certificate upon the establishment of an MQTT session. In this manner, the Host Application can still have a good representation of last known good process variable versus the fact that the MQTT session was terminated prior to the Edge Node being able to complete its transaction.

Regardless of a persistent or non-persistent connection, **Edge Node MQTT CONNECT packets MUST set the *Clean Session* flag to false.**

Chapter 3. Sparkplug Architecture and Infrastructure Components

This section details the infrastructure components implemented.

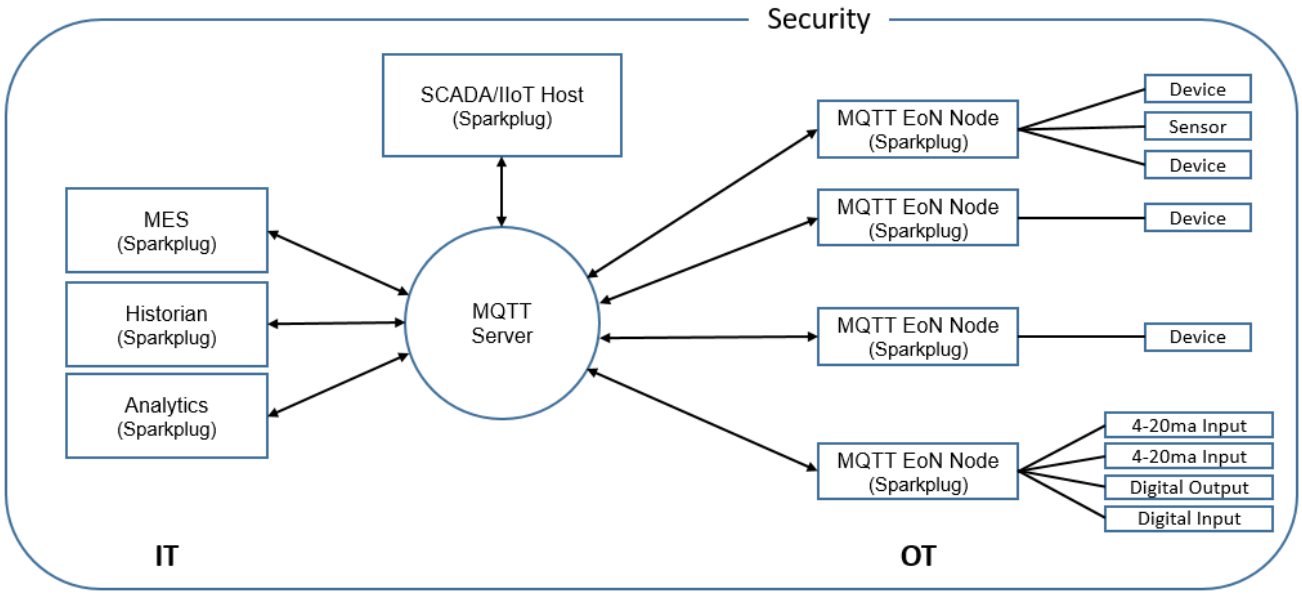

Figure 1 - MQTT SCADA Infrastructure

3.1. MQTT Server(s)

MQTT enabled infrastructure requires that one or more MQTT Servers are present in the infrastructure. The only requirement that the Eclipse Sparkplug™ specification places on the selection of an MQTT Server component in the architecture is it is required to be compliant with a subset of the MQTT specification. Specifically it must meet the requirements defined in the MQTT Server Conformance Section. The MQTT Server should also be sized to properly manage all MQTT message traffic.

One can implement the use (if required) of multiple MQTT servers for redundancy, high availability, and scalability within any given infrastructure.

3.2. MQTT Edge Node

In the context of this specification, an MQTT Edge Node is any MQTT v3.1.1 or v5.0 compliant MQTT Client application that manages an MQTT session and provides the physical and/or logical gateway functions required to participate in the topic namespace and payload definitions described in this document. The Edge Node is responsible for any local protocol interface to existing legacy devices (PLCs, RTUs, Flow Computers, Sensors, etc.) and/or any local discrete I/O, and/or any logical internal process variables(PVs).

3.3. Device / Sensor

The Device/Sensor represents any physical or logical device connected to the MQTT Edge Node providing any data, process variables or metrics.

3.4. MQTT Enabled Device (Sparkplug)

This represents any device, sensor, or hardware that directly connects to MQTT infrastructure using a compliant MQTT v3.1.1 or v5.0 connection with the payload and topic notation as outlined in this Sparkplug Specification. Note that it will be represented as an Edge Node in the Sparkplug topic.

3.5. Primary Host Application

A Primary Host Application is an MQTT client application that subscribes to MQTT Sparkplug Edge Node originated messages. It is deemed *primary* by the Edge Node. An Edge Node may be configured to modify its behavior based on one specific Sparklug Host Application being online or offline. This specific Host Application is referred to as the Edge Node's *Primary Host Application*.

The Primary Host Application is often also referred to as the SCADA Host or IIoT Host. In typical SCADA/IIoT infrastructure implementations, there will be only one Primary Host Application responsible for the monitoring and control of a given MQTT Edge Node. Sparkplug does support the notion of multiple Primary Host Applications for any one Edge Node. This does not preclude any number of additional Sparkplug Host Applications from participating in the infrastructure that are in either a pure monitoring mode, or in the role of a hot standby should the Primary Host Application go offline. In addition, there could be multiple Host Applications which are each the Primary Host Application for some subset of Edge Nodes connected to the MQTT infrastructure.

3.6. Sparkplug Host Application

A Sparkplug Host Application is any Sparkplug MQTT client that consumes the real-time Sparkplug messages or any other data being published with proper permission and security. **A Sparkplug Host Application MUST utilize the STATE messages to denote whether it is online or offline at any given point in time.**

Chapter 4. Topics and Messages

To get a working Message Oriented Middleware based SCADA system using MQTT, the first thing that must be defined is a topic namespace to work within. The beauty of MQTT is the fact that you can just come up with an arbitrary topic like “Portland/Temperature”, connect to an MQTT Server, and start publishing the temperature value. For this data to be useful to other MQTT Client applications that want to consume the temperature values, the Topic Namespace needs to be understood by everyone participating in the data exchange.

Every MQTT message published typically consists of a ***topic*** and ***payload*** components. These components are the overhead of an MQTT message as measured in bytes on the wire. The Eclipse Sparkplug™ Specification is designed to keep these components meaningful and easy to understand, but not to get so verbose as to negatively impact bandwidth/time sensitive data exchange.

4.1. Topic Namespace Elements

All MQTT clients using the Sparkplug specification MUST use the following topic namespace structure:

```
namespace/group_id/message_type/edge_node_id/[device_id]
```

4.1.1. namespace Element

The namespace element of the topic namespace is the root element that will define both the structure of the remaining namespace elements as well as the encoding used for the associated payload data. The Sparkplug specification defines two (2) namespaces. One is for Sparkplug payload definition A (now deprecated), and another one of for the Sparkplug payload definition B.

For the Sparkplug B version of the payload definition, the UTF-8 string constant for the namespace element MUST be:

```
spBv1.0
```

Note that for the remainder of this document, the version of the Sparkplug Payload definition does not affect the topic namespace or session state management as they will remain the same. There are separate definitions in this document for the encoding used for both the A and B versions of Sparkplug MQTT message payloads.

4.1.2. group_id Element

The Group ID element of the topic namespace provides for a logical grouping of Sparkplug Edge Nodes into the MQTT Server and back out to the consuming Sparkplug Host Applications.

The format of the Group ID MUST be a valid UTF-8 string with the exception of the reserved characters of + (plus), / (forward slash), and # (number sign). In most use cases to minimize

bandwidth, it should be descriptive but as small as possible. Examples of where the [group_id] might be used include Oil/Gas applications where Sparkplug Edge Nodes on a physical pipeline segment all have the same [group_id]. Plant floor applications may group Sparkplug Edge Nodes based on logical cell or manufacturing line requirements.

4.1.3. message_type Element

The message_type element of the topic namespace provides an indication as to how to handle the MQTT payload of the message. Note that the actual encoding of the payload will vary depending on the version of the Sparkplug implementation as indicated by the namespace element.

The following message_type elements are defined for the Sparkplug topic namespace:

- **NBIRTH** – Birth certificate for Sparkplug Edge Nodes
- **NDEATH** – Death certificate for Sparkplug Edge Nodes
- **DBIRTH** – Birth certificate for Devices
- **DDEATH** – Death certificate for Devices
- **NDATA** – Edge Node data message
- **DDATA** – Device data message
- **NCMD** – Edge Node command message
- **DCMD** – Device command message
- **STATE** – Sparkplug Host Application state message

The specification for each of these *message_type* elements are detailed later in this document.

4.1.4. edge_node_id Element

The edge_node_id element of the Sparkplug topic namespace uniquely identifies the Sparkplug Edge Node within the infrastructure. **The group_id combined with the *edge_node_id* element MUST be unique from any other group_id/edge_node_id assigned in the MQTT infrastructure. The format of the *edge_node_id* MUST be a valid UTF-8 string with the exception of the reserved characters of + (plus), / (forward slash), and # (number sign).** The topic element edge_node_id travels with every message published and should be as short as possible.

4.1.5. device_id Element

The device_id element of the Sparkplug topic namespace identifies a device attached (physically or logically) to the Sparkplug Edge Node. Note that the device_id is an optional element within the topic namespace as some messages will be either originating or destined to the edge_node_id and the device_id would not be required. **The format of the device_id MUST be a valid UTF-8 string except for the reserved characters of + (plus), / (forward slash), and # (number sign). The device_id MUST be unique from other devices being reported on by the same Edge Node. The device_id MAY be duplicated from Edge Node to other Edge Nodes.** The device_id element travels with every message published and should be as short as possible. **The device_id MUST be included with message_type elements DBIRTH, DDEATH, DDATA, and DCMD based topics The device_id MUST NOT be included with message_type elements NBIRTH, NDEATH, NDATA,**

NCMD, and STATE based topics

4.2. Message Types and Contents

Sparkplug defines the topic namespace for set of MQTT messages that are used to manage connection state as well as bidirectional metric information exchange that would apply to many typical real-time SCADA/IIoT, monitoring, and data collection system use cases. The defined message types are:

- **NBIRTH** – Birth certificate for Sparkplug Edge Nodes
- **NDEATH** – Death certificate for Sparkplug Edge Nodes
- **DBIRTH** – Birth certificate for Devices
- **DDEATH** – Death certificate for Devices
- **NDATA** – Node data message
- **DDATA** – Device data message
- **NCMD** – Node command message
- **DCMD** – Device command message
- **STATE** – Sparkplug Host Application state message

Using these defined messages Host Applications can:

- Discover all metadata and monitor state of all Edge Nodes and Devices connected to the MQTT infrastructure.
- Discover all metrics which include all diagnostics, properties, metadata, and current state values.
- Issue write/command messages to any Edge Node or Device metric.

This section defines the payload contents and how each of the associated messages types can be used.

4.2.1. Edge Node

Birth Message (NBIRTH)

Topic (NBIRTH)

The Birth Certificate topic for an Sparkplug Edge Node is:

```
namespace/group_id/NBIRTH/edge_node_id
```

Payload (NBIRTH)

The Sparkplug Edge Node Birth Certificate payload contains everything required to build out a data structure for all metrics for this Edge Node. At the time any Host Application receives an NBIRTH,

the ONLINE state of this Edge Node should be set to TRUE along with the associated ONLINE Date/Time parameter. Note that the Edge Node Birth Certificate ONLY indicates the Edge Node itself is online and in an MQTT Session, but any devices that have previously published a DBIRTH will still have STALE metric quality until the Host Application receives the associated DBIRTH messages.

The NBIRTH message requires the following payload components.

- **NBIRTH messages MUST be published with MQTT QoS equal to 0 and retain equal to false.**
- **The NBIRTH MUST include a sequence number in the payload and it MUST have a value of 0.**
- **The NBIRTH MUST include a timestamp denoting the Date/Time the message was sent from the Edge Node.**
- **The NBIRTH MUST include every metric the Edge Node will ever report on.**
- **At a minimum each metric MUST include the following:**
  - The metric name
  - The metric datatype
  - The current value
- **If Template instances will be published by this Edge Node or any devices, all Template definitions MUST be published in the NBIRTH.**
- **A bdseq number as a metric MUST be included in the payload.**
- **This MUST match the bdseq number provided in the MQTT CONNECT packet's Will Message payload.** This allows Host Applications to correlate NBIRTHs to NDEATHs.
- **The bdseq number MUST start at zero and increment by one on every new MQTT CONNECT packet.**

The NBIRTH message MUST include the following metric:

- Metric name: 'Node Control/Rebirth'
  - Used by Host Application(s) to request a new NBIRTH and DBIRTH(s) from an Edge Node.
  - Datatype: boolean
  - Value: false

The NBIRTH message can also include additional Node Control payload components. These are used by a Sparkplug Host Application to control aspects of the Edge Node. The following are examples of Node Control metrics.

- Metric name: 'Node Control/Reboot'
  - Used by Host Application(s) to reboot an Edge Node.
- Metric name: 'Node Control/Next Server'
  - Used by Host Application(s) to request an Edge Node to walk to the next MQTT Server in its list in multi-MQTT Server environments.
- Metric name: 'Node Control/Scan Rate'

    - Used by Host Application(s) to modify a poll rate on an Edge Node.

The NBIRTH message can also include optional 'Properties' of an Edge Node. The following are examples of Property metrics.

- Metric name: 'Properties/Hardware Make'
    - Used to transmit the hardware manufacturer of the Edge Node
- Metric name: 'Properties/Hardware Model'
    - Used to transmit the hardware model of the Edge Node
- Metric name: 'Properties/OS'
    - Used to transmit the operating system of the Edge Node
- Metric name: 'Properties/OS Version'
    - Used to transmit the OS version of the Edge Node

Data Message (NDATA)

Once an Sparkplug Edge Node is online with a proper NBIRTH it is in a mode of quiescent Report by Exception (RBE) or time based reporting of metric information that changes. This enables the advantages of the native Continuous Session Awareness of MQTT to monitor the STATE of all connected Sparkplug Edge Nodes and to rely on Report by Exception (RBE) messages for metric state changes over the MQTT session connection. Time based reporting is not explicitly disallowed by the Sparkplug Specification but it is discouraged. Due to the session awareness provided by MQTT and Sparkplug it is not necessary to send the same data again on a periodic basis.

Topic (NDATA)

The Data Topic for an Sparkplug Edge Node is:

```
namespace/group_id/NDATA/edge_node_id
```

The payload of NDATA messages will contain any RBE or time based metric Edge Node values that need to be reported to any subscribing MQTT clients.

Payload (NDATA)

The NDATA message requires the following payload components.

- **NDATA messages MUST be published with MQTT QoS equal to 0 and retain equal to false.**
- **The NDATA MUST include a sequence number in the payload and it MUST have a value of one greater than the previous MQTT message from the Edge Node contained unless the previous MQTT message contained a value of 255. In this case the sequence number MUST be 0.**
- **The NDATA MUST include a timestamp denoting the Date/Time the message was sent from the Edge Node.**
- **The NDATA MUST include the Edge Node's metrics that have changed since the last**

NBIRTH or NDATA message.

Death Message (NDEATH)

The Death Certificate topic and payload described here are not “published” as an MQTT message by a client, but provided as parameters within the MQTT CONNECT control packet when this Sparkplug Edge Node first establishes the MQTT Client session.

Immediately upon reception of an Edge Node Death Certificate (NDEATH message) with a bdSeq number that matches the preceding bdSeq number in the NBIRTH, any Host Application subscribed to this Edge Node should set the data quality of all metrics to STALE and should note the timestamp when the NDEATH message was received.

Topic (NDEATH)

The Death Certificate topic for an Sparkplug Edge Node is:

```
namespace/group_id/NDEATH/edge_node_id
```

Payload (NDEATH)

- **The NDEATH message contains a very simple payload that MUST only include a single metric, the bdseq number, so that the NDEATH event can be associated with the NBIRTH.** Since this is typically published by the MQTT Server on behalf of the Edge Node, information about the current state of the Edge Node and its devices is not and cannot be known. As a result, **the NDEATH message MUST NOT include a sequence number.**

The MQTT payload typically associated with this topic can include a Birth/Death sequence number used to track and synchronize Birth and Death sequences across the MQTT infrastructure. Since this payload will be defined in advance, and held in the MQTT server and only delivered on the termination of an MQTT session, not a lot of additional diagnostic information can be pre-populated into the payload.

Command (NCMD)

Topic (NCMD)

The NCMD command topic provides the topic namespace used to send commands to any connected Edge Nodes. This means sending an updated metric value to an associated metric included in the NBIRTH metric list.

```
namespace/group_id/NCMD/edge_node_id
```

Payload (NCMD)

The NCMD message requires the following payload components.

- **NCMD messages MUST be published with MQTT QoS equal to 0 and retain equal to false.**

- **The NCMD MUST include a timestamp denoting the Date/Time the message was sent from the Host Application's MQTT client.**
- **The NCMD MUST include the metrics that need to be written to on the Edge Node.**

4.2.2. Device / Sensor

Birth Message (DBIRTH)

The Sparkplug Edge Node is responsible for the management of all attached physical and/or logical devices. Once the Edge Node has published its NBIRTH, any Sparkplug Host Application ensures that the metric structure has the Edge Node in an ONLINE state. But each physical and/or logical device connected to this node will still need to provide this DBIRTH before Host Applications create/update the metric structure (if this is the first time this device has been seen) and set any associated metrics in the application to a "**GOOD**" state.

The DBIRTH payload contains everything required to build out a data structure for all metrics for this device. The ONLINE state of this device should be set to TRUE along with the associated ONLINE Date/Time this message was received.

Topic (DBIRTH)

The topic namespace for a Birth Certificate for a device is:

```
namespace/group_id/DBIRTH/edge_node_id/device_id
```

Payload (DBIRTH)

The DBIRTH message requires the following payload components.

- **DBIRTH messages MUST be published with MQTT QoS equal to 0 and retain equal to false.**
- **The DBIRTH MUST include a sequence number in the payload and it MUST have a value of one greater than the previous MQTT message from the Edge Node contained unless the previous MQTT message contained a value of 255. In this case the sequence number MUST be 0.**
- **The DBIRTH MUST include a timestamp denoting the Date/Time the message was sent from the Edge Node.**
- **The DBIRTH MUST include every metric the Edge Node will ever report on.**
- **At a minimum each metric MUST include the following:**
  - The metric name
  - The metric datatype
  - The current value

The DBIRTH message can also include optional 'Device Control' payload components. These are used by a Host Application to control aspects of a device. The following are examples of Device Control metrics.

- Metric name: ‘Device Control/Reboot’
  - Used by Host Application(s) to reboot a device.
- Metric name: ‘Device Control/Rebirth’
  - Used by Host Application(s) to request a new DBIRTH from a device.
- Metric name: ‘Device Control/Scan rate’
  - Used by Host Application(s) to modify a poll rate on a device.

The DBIRTH message can also include optional ‘Properties’ of a device. The following are examples of Property metrics.

- Metric name: ‘Properties/Hardware Make’
  - Used to transmit the hardware manufacturer of the device
- Metric name: ‘Properties/Hardware Model’
  - Used to transmit the hardware model of the device
- Metric name: ‘Properties/FW’
  - Used to transmit the firmware version of the device

Data Message (DDATA)

Once a Sparkplug Edge Node and associated Devices are all online with proper Birth Certificates it is in a mode of quiescent Report by Exception (RBE) reporting of any metric that changes. This takes advantage of the native Continuous Session Awareness of MQTT to monitor the STATE of all connected devices and can rely on Report by Exception (RBE) messages for any metric value change over the MQTT session connection. Again, time based reporting can be used instead of RBE but is discouraged and typically unnecessary.

Topic (DDATA)

As defined above, the Data Topic for an MQTT device is:

```
namespace/group_id/DDATA/edge_node_id/device_id
```

The payload of DDATA messages can contain one or more metric values that need to be reported.

Payload (DDATA)

The DDATA message requires the following payload components.

- **DDATA messages MUST be published with MQTT QoS equal to 0 and retain equal to false.**
- **The DDATA MUST include a sequence number in the payload and it MUST have a value of one greater than the previous MQTT message from the Edge Node contained unless the previous MQTT message contained a value of 255. In this case the sequence number MUST be 0.**
- **The DDATA MUST include a timestamp denoting the Date/Time the message was sent from**

the Edge Node.

- **The DDATA MUST include the Device's metrics that have changed since the last DBIRTH or DDATA message.**

Death Message (DDEATH)

It is the responsibility of the Sparkplug Edge Node to indicate the real-time state of either physical legacy device using poll/response protocols and/or local logical devices. If the device becomes unavailable for any reason (no response, CRC error, etc.) it is the responsibility of the Edge Node to publish a DDEATH on behalf of the end device.

Immediately upon reception of a DDEATH, any Host Application subscribed to this device should set the data quality of all metrics for the Device to STALE and should note the timestamp when the DDEATH message was received.

Topic (DDEATH)

The Sparkplug topic namespace for a device Death Certificate is:

```
namespace/group_id/DDEATH/edge_node_id/device_id
```

Payload (DDEATH)

The DDEATH message requires the following payload components.

- **DDEATH messages MUST be published with MQTT QoS equal to 0 and retain equal to false.**
- **The DDEATH MUST include a sequence number in the payload and it MUST have a value of one greater than the previous MQTT message from the Edge Node contained unless the previous MQTT message contained a value of 255. In this case the sequence number MUST be 0.**

Command (DCMD)

The DCMD topic provides the topic namespace used to publish metrics to any connected device. This means sending a new metric value to an associated metric included in the DBIRTH metric list.

Topic DCMD)

```
namespace/group_id/DCMD/edge_node_id/device_id
```

Payload (DCMD)

The DCMD message requires the following payload components.

- **DCMD messages MUST be published with MQTT QoS equal to 0 and retain equal to false.**
- **The DCMD MUST include a timestamp denoting the Date/Time the message was sent from the Host Application's MQTT client.**

- **The DCMD MUST include the metrics that need to be written to on the Device.**

Sparkplug Host Application

Birth Certificate Message (STATE)

The first MQTT message a Host Application MUST publish is a Birth Certificate. The Host Application Death Certificate is registered within the establishment of the MQTT session and is published as a part of the native MQTT transport if the MQTT session terminates for any reason.

The Birth Certificate that is defined here is an MQTT application level message published by the Sparkplug Host Application MQTT Client applications.

Birth Certificate Topic (STATE)

The topic used for the Host Birth Certificate is identical to the topic used for the Death Certificate:

```
STATE/sparkplug_host_application_id
```

- **The Birth Certificate Payload MUST be the UTF-8 string "ONLINE"**
- **The MQTT Quality of Service (QoS) MUST be set to 1**
- **The MQTT retain flag for the Birth Certificate MUST be set to TRUE**

Birth Certificate Payload (STATE)

- **The STATE message from the Sparkplug Host Application Birth Certificate message MUST include a payload that is a UTF-8 string that is the following**

```
ONLINE
```

Sparkplug B payloads are not used for encoding in this payload. This allows Host Applications to work across Sparkplug payload types.

Death Certificate Message (STATE)

When the Sparkplug Host Application MQTT client establishes an MQTT session to the MQTT Server(s), the Death Certificate will be part of the Will Topic and Will Payload registered in the MQTT CONNECT packet.

Death Certificate Topic (STATE)

The **Will Topic** as defined above will be:

```
STATE/sparkplug_host_application_id
```

- **The Sparkplug Host Application MUST provide a Will message in the MQTT CONNECT packet**

- **The MQTT Will Payload MUST be the UTF-8 string “OFFLINE”**
- **The MQTT Will QoS MUST be set to 1**
- **The MQTT Will retain flag MUST be set to TRUE**

Death Certificate Payload (STATE)

- **The STATE messages from the Sparkplug Host Application Death Certificate message MUST include a payload that is a UTF-8 string that is the following:**

```
OFFLINE
```

Sparkplug B payloads are not used for encoding in this payload. This allows Host Applications to work across Sparkplug payload types.

Chapter 5. Operational Behavior

An MQTT based SCADA system is unique in that the Primary Host Application is not responsible for establishing and maintaining connections to the Edge Nodes as is the case in most existing legacy poll/response device protocols. With an MQTT based architecture, both the Host Applications as well as the Edge Nodes establish MQTT Sessions with one or more central MQTT Servers. This is the desired functionality as it provides the necessary decoupling from any one application and any given Edge Node/Device. Additional Sparkplug Host Application MQTT clients can connect and subscribe to any of the real time data without impacting the Primary Host Application.

Due to the nature of real time SCADA solutions, it is very important for the Primary Host Application and all connected Eclipse Sparkplug™ Edge Nodes to have the MQTT Session STATE information for each other. In order to accomplish this the Sparkplug Topic Namespace definitions for Birth/Death Certificates along with the defined payloads provide both state and context between the Primary Host Application and the associated Edge Nodes. In most use cases and solution scenarios there are two main reasons for this "designation" of a Primary Host Application:

1. Only the Primary Host Application should have the permission to issue commands to Edge Nodes.
2. In high availability and redundancy use cases where multiple MQTT Servers are used, Sparkplug Edge Nodes need to be aware of whether the Primary Host Application has network connectivity to each MQTT Server in the infrastructure. If the Primary Host Application STATE shows that an Edge Node is connected to an MQTT Server that the Primary Host Application is **NOT** connected to, then the Edge Node should connect to the next available MQTT Server where STATE for the Primary Host Application is ‘ONLINE’.

5.1. Host Application Session Establishment

The Sparkplug Host Application upon startup or reconnect will immediately try to create a Host MQTT Session with the configured MQTT Server infrastructure. Note that the establishment of an Host Application MQTT session is asynchronous of any other MQTT Client session. If Edge Nodes are already connected to the MQTT Server infrastructure, the Sparkplug Host Application will synchronize using the STATE MQTT topic. If associated Edge Nodes are not connected, the Sparkplug Host Application will synchronize with the Edge Nodes and their data streams when the Edge Nodes publish their Birth Certificates. Any Edge Node that has specified this Sparkplug Host Application as its Primary Host Application will will wait to publish their Birth Certificates until after they receive the STATE message denoting that the Primary Host application is online.

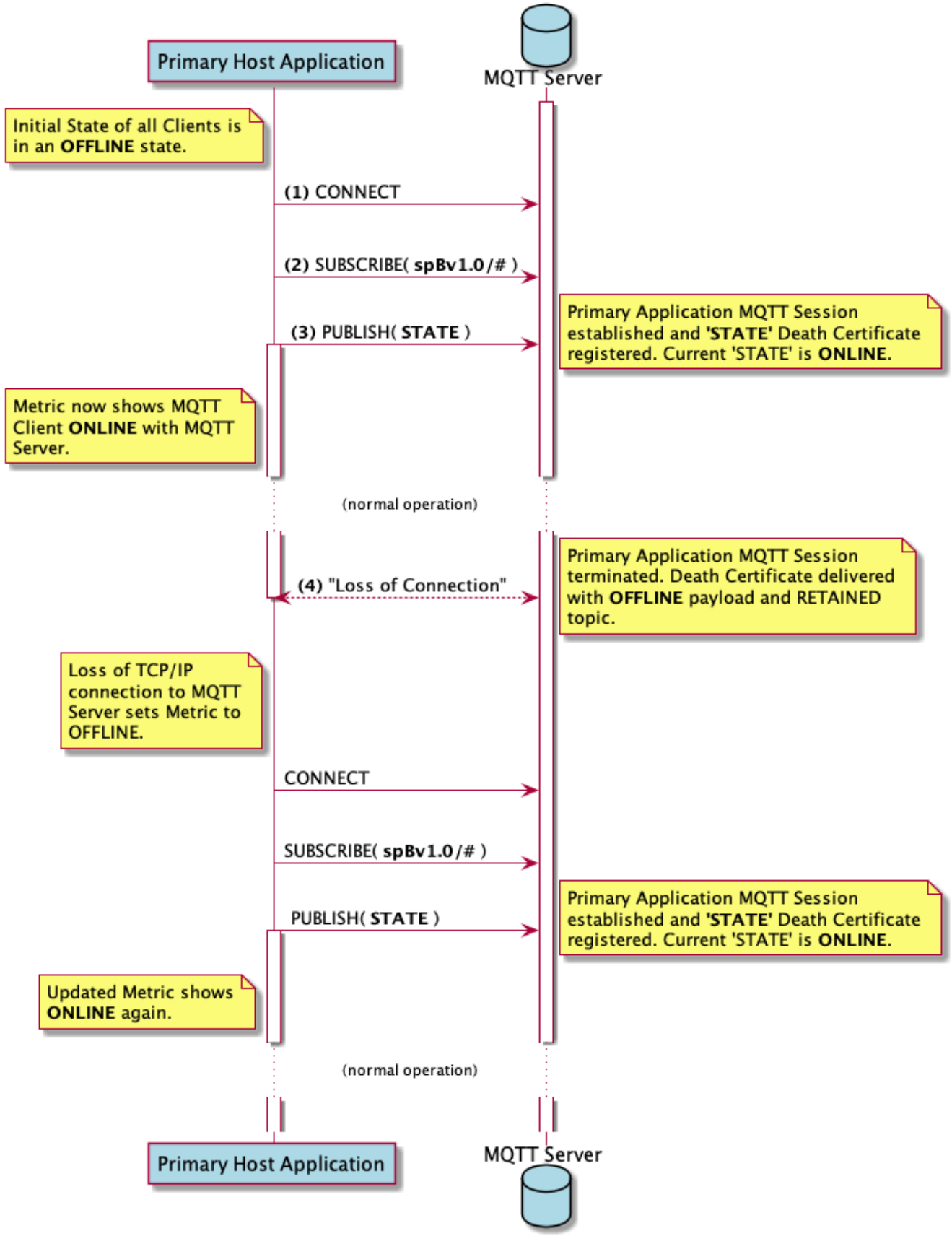

Figure 3 - Host Session Establishment

The session diagram in Figure 3 - Host Session Establishment shows a very simple topology with a single MQTT Server. The steps outlined in the session diagram are defined as follows:

1. Sparkplug Host Applications will try to create an MQTT Session using the MQTT CONNECT

Control Packet. A Death Certificate is constructed into the MQTT Will Topic and Will Payload of the of the CONNECT Control Packet with a Will QoS set to 1 and Will Retain flag set to true. **The CONNECT Control Packet for all Sparkplug Host Applications MUST set the MQTT *Clean Session* flag to true.** The MQTT CONNECT Control Packet is acknowledged as successful with a valid MQTT CONNACK Control Packet from the MQTT Server. From this point forward in time, the MQTT Server is ready to deliver a Host Death Certificate any time the Sparkplug Host Application MQTT Client loses connectivity to the MQTT Server.

2. With the MQTT Session established, Sparkplug Host Application MUST issue an MQTT subscription for the defined Sparkplug Topic Namespace. **The subscription on the Sparkplug Topic Namespace and the STATE topic MUST be done immediately after successfully establishing the MQTT session and before publishing its own STATE message.**

3. **Once an MQTT Session has been established, the Sparkplug Host Application subscriptions on the Sparkplug Topic Namespace have been established, and the STATE topic subscription has been been established, the Sparkplug Host Application MUST publish a new STATE message with a Payload of a UTF-8 string of *ONLINE*.** The Host Application is now ready to start receiving MQTT messages from any connected Edge Node within the infrastructure. At this point, the Host Application can update the MQTT Client metrics in the Host Application with a current state of ONLINE once each Edge Node publishes its Sparkplug NBIRTH and DBIRTH messages. Since the Sparkplug Host Application is also relying on the MQTT Session to the MQTT Server(s), the availability of MQTT Servers to the Host Application is also being monitored and reflected in the MQTT Client metrics in the Host Application.

4. If at any point in time Host Application loses connectivity with the defined MQTT Server(s), the ONLINE state of the Server is immediately reflected in the MQTT Client metrics in the Host Application. **All metric data associated with any Sparkplug Edge Node that was connected to that MQTT Server and known by the Host Application MUST be updated to a STALE data quality if the Host Application loses connection to the MQTT Server.**

5.2. Edge Node Session Establishment

- **Any Edge Node in the MQTT infrastructure MUST establish an MQTT Session prior to publishing NBIRTH and DBIRTH messages.**
- **Any Edge Node in the MQTT infrastructure MUST verify the Primary Host Application is ONLINE via the STATE topic if a Primary Host Application is configured for the Edge Node before publishing NBIRTH and DBIRTH messages.** Most implementations of a Sparkplug Edge Node for real time SCADA systems will try to maintain a persistent MQTT Session with the MQTT Server Infrastructure. But there are use cases where the MQTT Session does not need to be persistent. In either case, an Edge Node can try to establish an MQTT Session at any time and is completely asynchronous from any other MQTT Client in the infrastructure. The only exception to this rule is the use case where there are multiple MQTT Servers and a Primary Host Application. Note this does not refer to the use of the MQTT *Clean Session* flag. All types of MQTT clients (both Host and Edge Nodes) in a Sparkplug system MUST always set the *Clean Session* flag in the MQTT CONNECT packet to true.

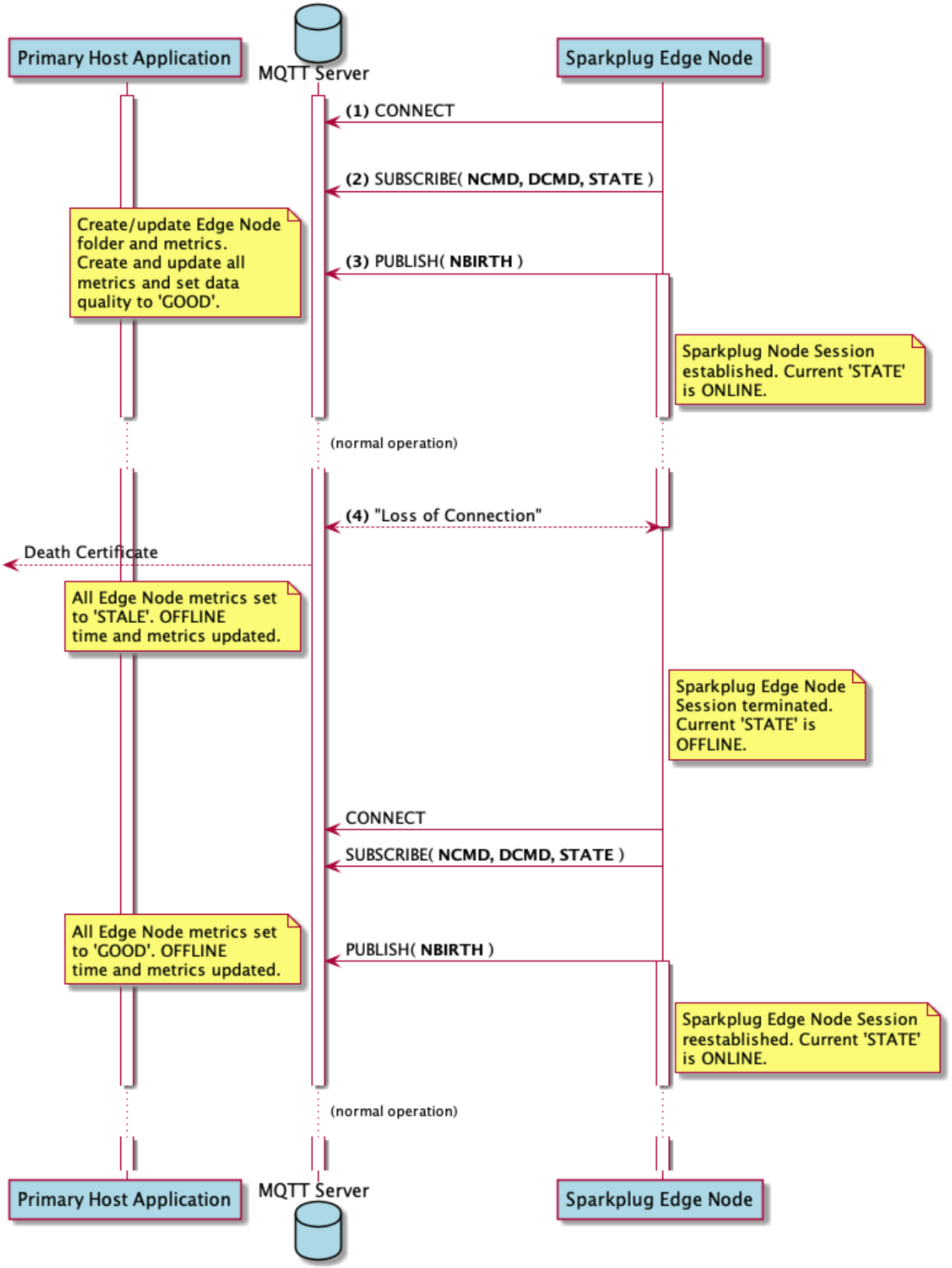

Figure 4 - Edge Node MQTT Session Establishment

The session diagram in Figure 4 - Edge Node MQTT Session Establishment shows a very simple topology with a single MQTT Server. The steps outlined in the session diagram are defined as follows:

1. The Edge Node MQTT Client will attempt to create an MQTT connection to the available MQTT Server(s) using the MQTT CONNECT Control Packet. The Death Certificate constructed into the Will Topic and Will Payload follows the format defined in section on NDEATH messages.
2. The subscription to NCMD level topics ensures that Edge Node targeted messages from the Primary Host Application are delivered. The subscription to DCMD ensures that device targeted messages from the Primary Host Application are delivered. In infrastructures with multiple MQTT Servers and a designated Primary Host Application, the subscription to STATE informs the Edge Node the current state of the Primary Host Application. At this point the Edge node has fully completed the steps required for establishing a valid MQTT Session with the Primary Host Application.
3. Once an MQTT Session has been established, the Edge Node MQTT client MUST publish an application level NBIRTH as defined here. At this point, the Primary Host Application will have all the information required to build out the Edge Node metric structure and show the Edge Node in an "ONLINE" state once it publishes its NBIRTH and DBIRTH messages.
4. If at any point in time the Edge Node MQTT Client loses connectivity to the defined MQTT Server(s), a Death Certificate (NDEATH) is issued by the MQTT Server on behalf of the Edge Node. Upon receipt of the Death Certificate with a bdSeq number metric that matches the preceding bdSeq number in the NBIRTH messages, the Primary Host Application should set the state of the Edge Node to ‘OFFLINE’ and update all metric timestamps related to this Edge Node. Any defined metrics will be set to a STALE data quality.
    a. The bdSeq number is used to correlate an NBIRTH with a NDEATH. Because the NDEATH is included in the MQTT CONNECT packet, its timestamp (if included) is not useful to Sparkplug Host Applications. Instead, a bdSeq number must be included as a metric in the payload of the NDEATH. The same bdSeq number metric value must also be included in the NBIRTH message published immediately after the MQTT CONNECT. This allows Host Applications to know that a NDEATH matches a specific NBIRTH message. This is required because timing with Will Messages may result in NDEATH messages arriving after a new/next NBIRTH message. The bdSeq number allows Host Applications to know when it must consider the Edge Node offline.

5.3. Edge Node Session Termination

Edge Nodes for various reasons may disconnect intentionally. When this is done, **an Edge Node MUST publish an NDEATH before terminating the connection. Immediately following the NDEATH publish, a DISCONNECT packet MUST be sent to the MQTT Server.** This allows the MQTT Server to be notified that the Edge Node is offline and as a result the MQTT Will Message of the Edge Node will not be delivered by the MQTT Server to subscribed MQTT clients.

When an Edge Node goes offline by sending its NDEATH, it is implied that all of the Edge Node’s associated Devices are also offline.

5.4. Device Session Establishment

The Sparkplug Specification is provided to get real time process variable information from existing and new end devices measuring, monitoring, and controlling a physical process into an MQTT infrastructure and the Host Application Industrial Internet of Things application platform. In the

context of this document an MQTT Device can represent anything from existing legacy poll/response driven PLCs, RTUs, HART Smart Transmitter, etc., to new generation automation and instrumentation devices that can implement a conformant MQTT client natively.

The preceding sections in this document detail how the Sparkplug Host Application interacts with the MQTT Server infrastructure and how that infrastructure interacts with the notion of a Sparkplug Edge Node. But to a large extent the technical requirements of those pieces of the infrastructure have already been provided. For most use cases in this market sector the primary focus will be on the implementation of the Sparkplug Specification between the native device and the Edge Node API's.

In order to expose and populate the metrics from any intelligent device, the following simple session diagram outlines the requirements:

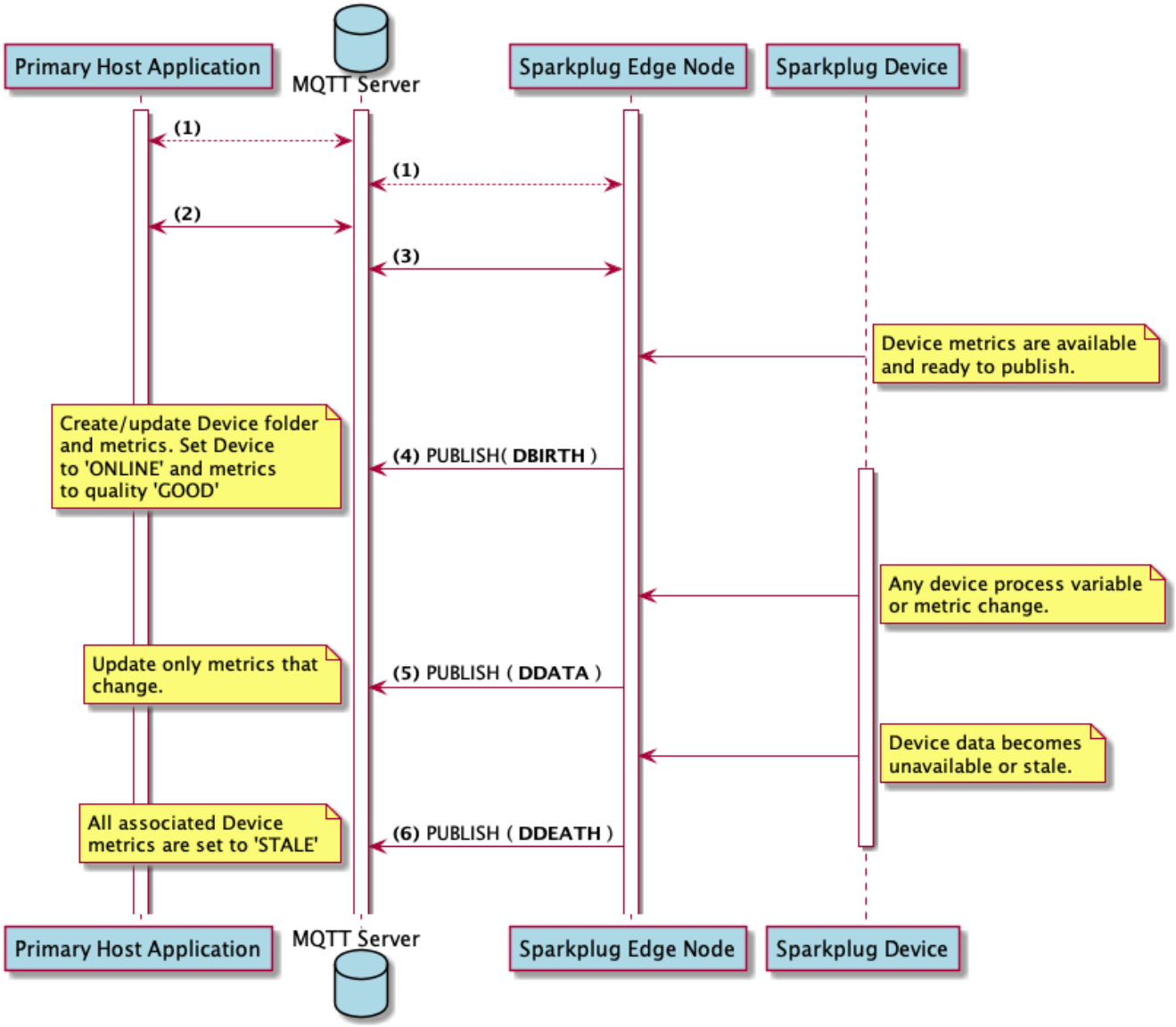

Figure 5 - MQTT Device Session Establishment

The session diagram in Figure 5 - MQTT Device Session Establishment shows a simple topology with all the Sparkplug elements in place i.e. Host Application, MQTT Server(s), Sparkplug Edge Node and this element, the device element. The steps outlined in the session diagram are defined as follows:

This flow diagram assumes that at least one MQTT Server is available and operational within the

infrastructure. Without at least a single MQTT Server the remainder of the infrastructure is unavailable.

1. Assuming MQTT Server is available.
2. Assuming the Primary Host Application established MQTT Session with the MQTT Server(s).
3. The Session Establishment of the associated Sparkplug Edge Node is described in Edge Node Session Establishment. This flow diagram assumes that the Edge Node session has already been established with the Primary Host Application. Depending on the target platform, the Edge Node may be a physical "Edge of Network" gateway device polling physical legacy devices via Modbus, AB, DNP3.0, HART, etc, an MQTT enabled sensor or device, or it might be a logical implementation of one of the Eclipse Tahu compatible implementations for prototype Edge Nodes running on the Raspberry PI platform. Regardless of the implementation, at some point the device interface will need to provide a state and associated metrics to publish to the MQTT infrastructure.
4. State #4 in the session diagram represents the state at which the Edge Node is ready to report all of its metric data to the MQTT Server(s) as defined in Sparkplug. It is the responsibility of the Edge node (logical or physical) to put this information in a form defined in DBIRTH messages. Upon receiving the DBIRTH message, the Primary Host Application can build out the proper metric structure and set the Sparkplug Device to *online*.
5. Following the Sparkplug Specification in Device Data Messages (DDATA), all subsequent metrics are published to the Primary Host Application on a Report by Exception (RBE) basis using the DDATA message format.
6. If at any time the Sparkplug Device cannot provide real time information, the Sparkplug Specification requires that an DDEATH be published. This will inform the Primary Host Application that all metric information associated with that Sparkplug Device be set to a STALE data quality.

5.5. Device Session Termination

If a Sparkplug Edge Node loses connection with an attached Sparkplug Device, it MUST publish a DDEATH message on behalf of the device. This allows Sparkplug Host Applications to know that the Device is no longer connected and therefore the Edge Node is not able to report live/accurate data values. In turn, the Sparkplug Host Applications MUST mark the Device offline and denote that Device's tags as stale.

5.6. Sparkplug Host Applications

As noted above, there is the notion of a Sparkplug Host Application in the infrastructure that has the required permissions to send commands to Edge Nodes and Sparkplug Devices and the fact that all Edge Nodes need to know the Primary Host Application is connected to the same MQTT Server its connected to or it needs to walk to another one in the infrastructure. Both are common requirements of a mission critical SCADA system.

But unlike legacy SCADA system implementations, all real time process variable information being published thru the MQTT infrastructure is available to any number of additional MQTT Clients in the business that might be interested in subsets if not all of the real time data.

The only fundamental difference between a Primary Host Application MQTT Client and other Sparkplug Host Application MQTT Clients is that the Edge Nodes in the infrastructure know to make sure the Primary Host Application is online before publishing data.

5.7. Sparkplug Host Application Message Ordering

Sparkplug Host Applications are required to validate the order of messages arriving from Edge Nodes. This is done using the sequence number which is sent in every NBIRTH, DBIRTH, NDATA, and DDATA message that comes from an Edge Node. Because these MQTT messages are sent on different topics, it is possible based on MQTT Server implementations that these messages may arrive at the Sparkplug Host Application in a different order than they were sent from the Edge Node. This can be especially common when using clustered MQTT Servers. It is the responsibility of the Sparkplug Host Application to ensure that all messages arrive within a *Reorder Timeout*. In typical environments this timeout can be as little as a couple of seconds. In deployments with very slow networks or clustered MQTT servers it may need to be longer. In some environments, the MQTT Server may ensure in-order delivery of QoS0 MQTT messages even across topics. In these cases this timeout could be zero.

If a Sparkplug Host Applications receives messages from Edge Node with sequence numbers 1, 2, and 4. At the time the message with a sequence number of 4 arrives, a timer SHOULD be started within the Host Application. This is the start of the Reordering Timeout timer. A messages with sequence number 3 MUST arrive before the Reordering Timeout elapses. If a message with sequence number 3 does not arrive before the timeout, a Rebirth Request SHOULD be sent to the Edge Node. The ensures the session state is properly reestablished. If a message with a sequence number of 3 arrives before the Reorder Timeout occurs, the timer can be shutdown and normal operation of the Host Application can continue.

It is also important to note that depending on the Sparkplug Host Application's purpose, it may make sense to never process messages out of order. It also may make sense to not process a message that arrived out of sequence if its preceding messages didn't arrive before the Reorder Timeout. These choices are left to the Sparkplug Host Application developer.

- **Sparkplug Host Applications SHOULD provide a configurable *Reorder Timeout* parameter**
- **If a message arrives with a out of order sequence number, the Host Application SHOULD start a timer denoting the start of the Reorder Timeout window**
- **If the Reorder Timeout elapses and the missing message(s) have not been received, the Sparkplug Host Application SHOULD send an NCMD to the Edge Node with a *Node Control/Rebirth* request**
    - Non-normative comment: In most cases a *Primary Host Application* would send a Rebirth Request but a Non-Primary Host may not
- **If the missing messages that triggered the start of the Reorder Timeout timer arrive before the reordering timer elapses, the timer can be terminated and normal operation in the Host Application can continue**

5.8. Primary Host Application STATE in Multiple MQTT Server Topologies

For implementations with multiple MQTT Servers, there is one additional aspect that needs to be understood and managed properly. When multiple MQTT Servers are available there is the possibility of "stranding" an Edge Node if the Primary command/control of the Primary Host Application loses network connectivity to one of the MQTT Servers. In this instance the Edge Node would stay properly connected to the MQTT Server publishing information not knowing that Primary Host Application was not able to receive the messages. When using multiple MQTT Servers, the Primary Host Application instance must be configured to publish a STATE Birth Certificate and all Edge Nodes need to subscribe to this STATE message.

Regardless of the number of MQTT Servers in a Sparkplug Infrastructure, every time a Primary Host Application establishes a new MQTT Session with an MQTT Server, the STATE Birth Certificate defined in the STATE description section MUST be the first message that is published after a successful MQTT Session is established with each MQTT Server.

Sparkplug Edge Nodes in an infrastructure that provides multiple MQTT Servers can establish a session to any one of the MQTT Servers.

The Edge Nodes MUST not connected to more than one server at any point in time.

Upon establishing a session, the Edge Node should issue a subscription to the STATE message published by Primary Host Application. Since the STATE message is published with the RETAIN message flag set, MQTT will guarantee that the last STATE message is always available. The Edge Node should examine the payload of this message to ensure that it is a value of "ONLINE". If the value is "OFFLINE", this indicates the Primary Application has lost its MQTT Session to this particular MQTT Server.

If the Primary Host Application is OFFLINE as denoted via the STATE MQTT Message, the Edge Node MUST terminate its session with this MQTT Server and move to the next available MQTT Server that is available.

The Edge Node MUST also wait to publish its BIRTH sequence until an "ONLINE" STATE message is received by the Edge Node. This use of the STATE message in this manner ensures that any loss of connectivity to an MQTT Server to the Primary Host Application does not result in Edge Nodes being "stranded" on an MQTT server because of network issues. The following message flow diagram outlines how the STATE message is used when three (3) MQTT Servers are available in the infrastructure:

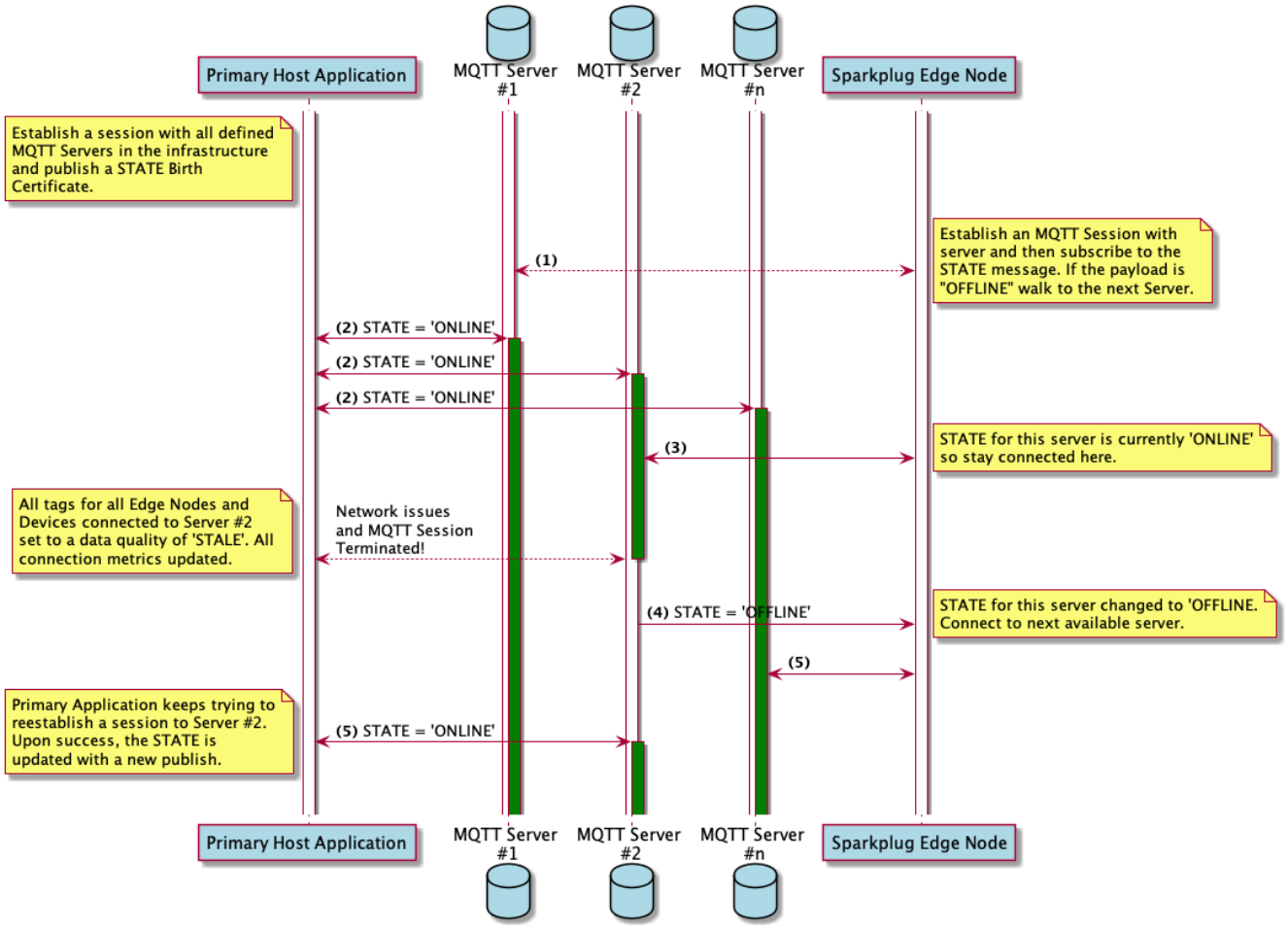

Figure 7 – Primary Host Application STATE flow diagram

1. When an Edge Node is configured with multiple available MQTT Servers in the infrastructure it should issue a subscription to the Primary Host Application STATE message. The Edge Nodes are free to establish an MQTT Session to any of the available servers over any available network at any time and examine the current STATE value. If the STATE message payload is ‘OFFLINE’ then the Edge Node should disconnect and walk to the next available server.
2. Upon startup, the configured Primary Host Application’s MQTT Client MUST include the Primary Host Application DEATH Certificate that indicates STATE is ‘OFFLINE’ with the message RETAIN flag set to true in the MQTT Will Message. Then the Primary Host Application BIRTH Certificate MUST be published with a STATE payload of ‘ONLINE’.
3. As the Edge Node walks its available MQTT Server list, it will establish an MQTT Session with a server that has a STATE message with a payload of ‘ONLINE’. The Edge Node can stay connected to this server if its MQTT Session stays intact and it does not receive the Primary Host Application DEATH Certificate.
4. Having a subscription registered to the MQTT Server on the STATE topic will result in any change to the current the Primary Host Application STATE being received immediately. In this case, a network disruption causes the Primary Host Application MQTT Session to server #2 to be terminated. This will cause the MQTT Server, on behalf of the now terminated the Primary Host Application MQTT Client to publish the Death Certificate to anyone that is currently subscribed to it. Upon receipt of the Primary Host Application Death Certificate this Edge Node will move to the next MQTT Server in its list.
5. The Edge Node connected to the next available MQTT Server and since the current STATE on

this server is ‘ONLINE’, it can stay connected. In the meantime, the network disruption between Primary Host Application and MQTT Server #2 has been corrected. The Primary Host Application has a new MQTT Session established to server #2 with an update Birth Certificate of ‘ONLINE’. Now MQTT Server #2 is ready to accept new Edge Node session requests.

5.9. Edge Node NDATA and NCMD Messages

We’ll start this section with a description of how metric information is published to the Primary Host Application from an Edge Node in the MQTT infrastructure. The definition of an Edge Node is generic in that it can represent both physical "Edge of Network Gateway" devices that are interfacing with existing legacy equipment and a logical MQTT endpoint for devices that natively implement the Sparkplug Specification. The NBIRTH Section defines the Edge Node Birth Certificate MQTT Payload and the fact that it can provide any number of metrics that will be exposed in the Primary Host Application. Some examples of these will be "read only" such as:

- Edge Node Manufacture ID
- Edge Node Device Type
- Edge Node Serial Number
- Edge Node Software Version Number
- Edge Node Configuration Change Count
- Edge Node Position (if GPS device is available)
- Edge Node Cellular RSSI value (if cellular is being used)
- Edge Node Power Supply voltage level
- Edge Node Temperature

Other metrics may be dynamic and "read/write" such as:

- Edge Node Rebirth command to republish all Edge Node and Device Birth Certificates
- Edge Node Next server command to move to next available MQTT Server
- Edge Node Reboot command to reboot the Edge Node
- Edge Node Primary Network (PRI_NETWORK) where 1 = Cellular, 2 = Ethernet

The important point to realize is that the metrics exposed in the Primary Host Application for use in the design of applications are completely determined by what metric information is published in the NBIRTH. This is entirely dependent on the application and use-case. Each specific Edge Node can best determine what data to expose, and how to expose it, and it will automatically appear in the Primary Host Application metric structure. Metrics can even be added dynamically at runtime and with a new NBIRTH and DBIRTH sequence of messages. These metrics will automatically be added to the Primary Host Application metric structure.

The other very important distinction to make here is that Edge Node NDATA and NCMD messages are decoupled from the Sparkplug Device level data and command messages of DDATA and DCMD. This decoupling in the Topic Namespace is important because it allows interaction from all MQTT Clients in the system (to the level of permission and application) with the Edge Nodes, but NOT to

the level of sending device commands. The Primary Host Application could provide a configuration parameter that would BLOCK output DDATA and DCMD messages but still allow NDATA and NCMD messages to flow. In this manner, multiple application systems can be connected to the same MQTT infrastructure, but only the ones with DCMD enabled can publish Device commands.

The following simple message flow diagram demonstrates the messages used to update a changing cellular RSSI value in the Primary Host Application and sending a command from the Primary Host Application to the Edge Node to use a different primary network path.

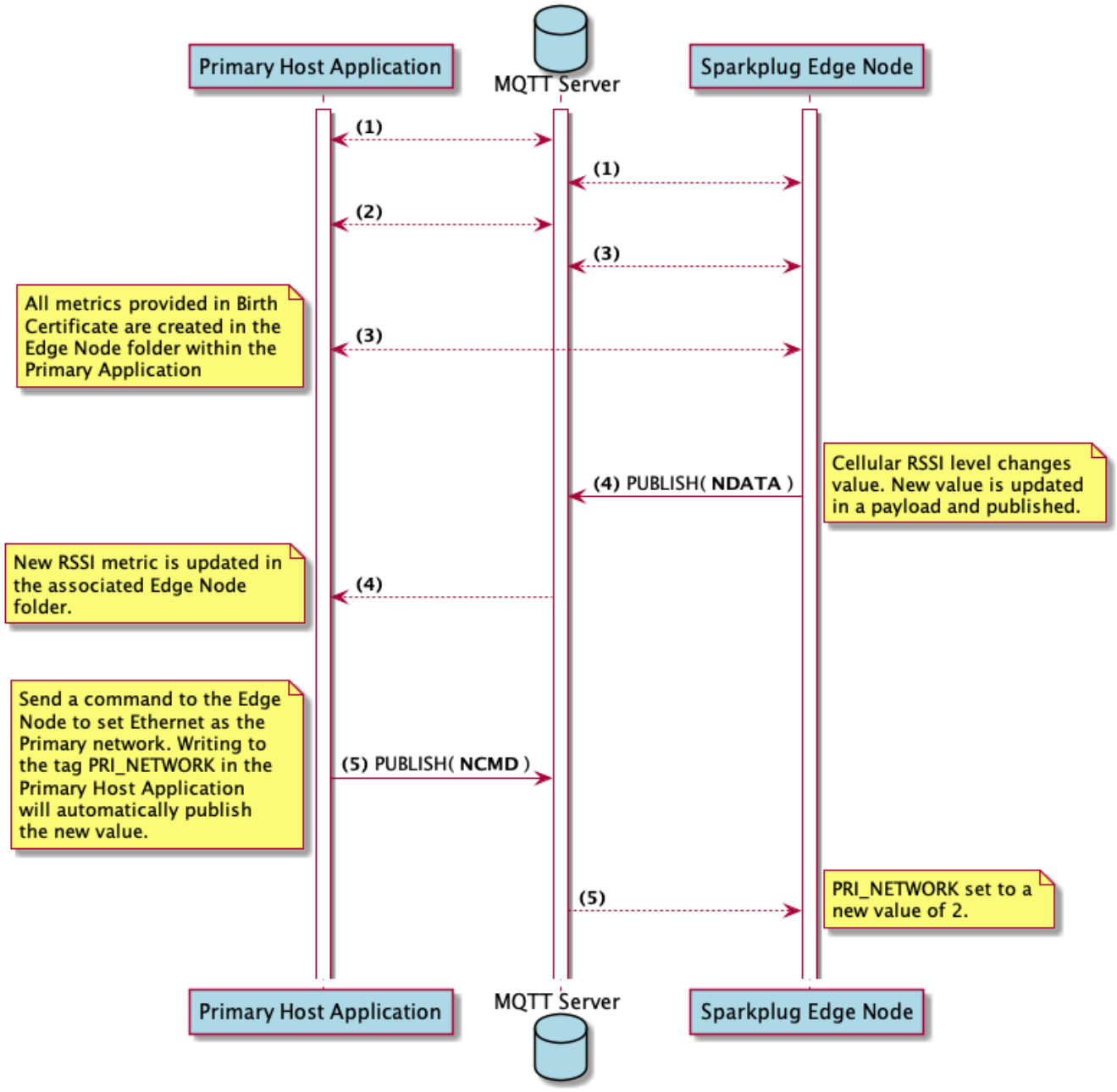

Figure 6 - Edge Node NDATA and NCMD Message Flow

1. Assuming MQTT Server is available.
2. Assuming the Primary Host Application established MQTT Session with the MQTT Server(s).
3. The Edge Node has an established MQTT Session and the NBIRTH has been published. Primary Host Application now has all defined metrics and their current value.
4. The Edge Node is monitoring its local cellular RSSI level. The level has changed and now the

Edge Node wants to publish the new value to the associated metric in Primary Host Application.

5. From an operational requirement, the Edge Node needs to be told to switch its primary network interface from cellular to Ethernet. From the Primary Host Application, the new metric value is published to the Edge Node using a NCMD Sparkplug message.

5.10. MQTT Enabled Device Session Establishment

When implementing Sparkplug directly on an I/O enabled Device, there are two options. The notion of a *Sparkplug Device* can be removed entirely. In this scenario the MQTT Client can publish *Edge Node level* messages (e.g. NBIRTH, NDEATH, NCMD, and NDATA) and never use the concept of *Device level* messages (e.g. DBIRTH, DDEATH, DCMD, and DDATA messages. All of the metrics can be published on the Edge Node level Sparkplug verbs and simply omit use of the Device level Sparkplug verbs. Because the Edge Node level verbs encapsulate the MQTT/Sparkplug Session, this is all that is required.

Alternatively, the implementation can use the concept of both Edge Node and Device Sparkplug verbs (NBIRTH, NDEATH, NDATA, NCMD, DBIRTH, DDEATH, DDATA, and DCMD) as any other Gateway based Edge Node would. From any consuming application this would look like any other Edge Node Gateway that may be managing one or more attached devices.

5.11. Sparkplug Host Application Session Establishment

Sparkplug Host Applications must follow the following rules when connecting to the MQTT Server.

- **The host_id MUST be unique to all other Sparkplug Host IDs in the infrastructure.**
- **When a Sparkplug Host Application sends its MQTT CONNECT packet, it MUST include a Will Message.**
- **The MQTT Will Message's topic MUST be of the form *STATE/host_id* where host_id is the unique identifier of the Sparkplug Host Application**
- **The MQTT Will Message's payload MUST be the ASCII String of *OFFLINE*.**
- **The MQTT Will Message's MQTT QoS MUST be 1 (at least once).**
- **The MQTT Will Message's retained flag MUST be set to true.**

Once the Sparkplug Host Application has successfully connected to the MQTT Server, it must publish a birth with the following rules.

- **The MQTT Client associated with the Sparkplug Host Application MUST send a birth message immediately after successfully connecting to the MQTT Server.**
- **The Host Application's Birth topic MUST be of the form *STATE/host_id* where host_id is the unique identifier of the Sparkplug Host Application**
- **The Host Application's Birth payload MUST be the ASCII String of *ONLINE*.**
- **The Host Application's Birth MQTT QoS MUST be 1 (at least once).**
- **The Host Application's Birth retained flag MUST be set to true.**

5.12. Sparkplug Host Application Session Termination

[tck-testable tck-id-operational-behavior-host-application-termination]#[yellow-background]*If the Sparkplug Host Application ever disconnects intentionally, it must publish a Death message with the following characteristics.

- **The Sparkplug Host Application's Death topic MUST be of the form *STATE/host_id* where host_id is the unique identifier of the Sparkplug Host Application.**
- **The Sparkplug Host Application's Death payload MUST be the ASCII String of *OFFLINE*.**
- **The Sparkplug Host Application's Death MQTT QoS MUST be 1 (at least once).**
- **The Sparkplug Host Application's Death retained flag MUST be set to true.**

In the case of intentionally disconnecting, an MQTT DISCONNECT packet MUST be sent immediately after the Death message is sent.

5.13. Data Publish

Publishing of data messages occurs from an Edge Node any time it is online as denoted by previously publishing its BIRTH messages within the same MQTT Session. A Sparkplug session begins with an MQTT CONNECT and then the NBIRTH message. A Sparkplug session ends with an NDEATH. Using the fact that MQTT uses TCP as the underlying protocol as well as facilities in Sparkplug to encapsulate a session, data messages are sent *by exception*. In other words, data only has to be sent when it changes. This is true as long as the session remains established and valid. The following set of rules defines how data messages should be sent.

Rules for Edge Node data (NBIRTH and NDATA) messages:

- **NBIRTH messages MUST include all metrics for the specified Edge Node that will ever be published for that Edge Node within the established Sparkplug session.**
- **NBIRTH messages MUST include current values for all metrics.**
- **NDATA messages MUST only be published when Edge Node level metrics change.**
  - In other words, metric values that have not changed within the same Sparkplug Session MUST not be resent until a new Sparkplug session is established.
- NDATA messages SHOULD be aggregated to include multiple metrics.
  - This is up to the application developer in terms of how many metrics should be aggregated in a single message, but it typically doesn't make sense to publish an MQTT message for every single metric change.
  - Multiple value changes for the same metric MAY be included in the same Sparkplug NDATA message as long as they have different timestamps.
- **For all metrics where is_historical=false, NBIRTH and NDATA messages MUST keep metric values in chronological order in the list of metrics in the payload.**

Rules for Device data (DBIRTH and DDATA) messages:

- **DBIRTH messages MUST include all metrics for the specified Device that will ever be**

published for that Device within the established Sparkplug session.

- **DBIRTH messages MUST include current values for all metrics.**
- **DDATA messages MUST only be published when Device level metrics change.**
  - In other words, metric values that have not changed within the same Sparkplug Session MUST not be resent until a new Sparkplug session is established.
- NDATA messages SHOULD be aggregated to include multiple metrics.
  - This is up to the application developer in terms of how many metrics should be aggregated in a single message, but it typically doesn't make sense to publish an MQTT message for every single metric change.
  - Multiple value changes for the same metric MAY be included in the same Sparkplug DDATA message as long as they have different timestamps.
- **For all metrics where is_historical=false, DBIRTH and DDATA messages MUST keep metric values in chronological order in the list of metrics in the payload.**

5.14. Commands

Commands are used in Sparkplug to allow Sparkplug Host Applications to send data to Sparkplug Edge Nodes. Examples include writing to outputs of Sparkplug Edge Nodes and Devices or to request Rebirths from Edge Nodes. Custom command endpoints can be declared in an NBIRTH or DBIRTH message by an Edge Node or Device that may support functionality such as rebooting an Edge Node or Device. This is up to the Sparkplug implementor to define what functionality can be exposed.

Security and access is an important aspect of commands. It may be the case that not all Sparkplug Host Applications should have the ability to send commands. This can be be controlled in multiple ways. ACLs (Access Control Lists) may be used to allow/disallow certain MQTT clients from publishing NCMD and DCMD messages. Security features in the Sparkplug Host Application itself could be used to allow/disallow certain users or applications from sending certain commands. Security features in the Sparkplug Edge Node application could be used to allow/disallow CMD messages to be honored. There are a number of ways in which this can be done and should be considered. However, implementation details are not covered in the Sparkplug Specification and is left to specific application designers to consider.

There are two types of command (CMD) verbs in Sparkplug. These are NCMD and DCMD messages which target Edge Nodes and Devices respectively.

There is one NCMD that is required to be implemented for all Sparkplug Edge Nodes and that is the *Node Control/Rebirth* command. This exists to allow a Sparkplug Host Application to reset its end-to-end session with a specific Edge Node. For example, say an Edge Node has been in an established Sparkplug session and is publishing DATA messages. Now say a new Sparkplug Host Application connects to the same MQTT Server that the Edge Node is connected to. On the next DATA message published by the Edge Node, the Host Application will receive it without ever having received the BIRTH message(s) associated with the Edge Node. As a result, it can send a *Rebirth Request* using the *Node Control/Refresh* metric to reset its understanding of that Edge Node and become aware of all metrics associated with it.

These are the rules around the *Node Control/Rebirth* metric.

- **An NBIRTH message MUST include a metric with a name of *Node Control/Rebirth*.**
- **The *Node Control/Rebirth* metric in the NBIRTH message MUST have a datatype of *Boolean*.**
- **The *Node Control/Rebirth* metric value in the NBIRTH message MUST have a value of false.**

A *Rebirth Request* consists of the following message from a Sparkplug Host Application with the following characteristics.

- **A Rebirth Request MUST use the NCMD Sparkplug verb.**
- **A Rebirth Request MUST include a metric with a name of *Node Control/Rebirth*.**
- **A Rebirth Request MUST include a metric value of true.**

Upon receipt of a Rebirth Request, the Edge Node must do the following.

- **When an Edge Node receives a Rebirth Request, it MUST immediately stop sending DATA messages.**
- **After an Edge Node stops sending DATA messages, it MUST send a complete BIRTH sequence including the NBIRTH and DBIRTH(s) if applicable.**
- **The NBIRTH MUST include the same bdSeq metric with the same value it had included in the Will Message of the previous MQTT CONNECT packet.**
    - Because a new MQTT Session is not being established, there is no reason to update the bdSeq number
- After the new BIRTH sequence is published, the Edge Node may continue sending DATA messages.

Another common use case for sending commands is to use them to *write* to outputs on Sparkplug Devices. Often these are PLCs or RTUs with writable outputs. NCMD and DCMD messages can be used for these writes. The general flow is for a Host Application to send a command message, the Edge Device receives the message and writes to the output using the native protocol. Then when the output changes value, it results in the Edge Node publishing a DATA message denoting the new value.

For Edge Node level commands, the following rules must be followed.

- **An Edge Node level command MUST use the NCMD Sparkplug verb.**
- **An NCMD message MUST include a metric name that was included in the associated NBIRTH message for the Edge Node.**
- **An NCMD message MUST include a compatible metric value for the metric name that it is writing to.**
    - In other words, if the metric has a datatype of a boolean the value must be true or false.

For Device level commands, the following rules must be followed.

- **A Device level command MUST use the DCMD Sparkplug verb.**

- **A DCMD message MUST include a metric name that was included in the associated DBIRTH message for the Device.**
- **A DCMD message MUST include a compatible metric value for the metric name that it is writing to.**
    - In other words, if the metric has a datatype of a boolean the value must be true or false.

Chapter 6. Payloads

6.1. Overview

The MQTT specification does not define any required data payload format. From an MQTT infrastructure standpoint, the payload is treated as an agnostic binary array of bytes that can be anything from no payload at all, to a maximum of 256MB. But for applications within a known solution space to work using MQTT the payload representation does need to be defined.

This section of the Eclipse Sparkplug™ specification defines how MQTT Sparkplug payloads are encoded and the data that is required. Sparkplug supports multiple payloads encoding definitions. There is an *A* payload format as well as a *B* payload format. As described in the Introduction Section Sparkplug A is deprecated and is not included in this document. This section will only cover the details of the Sparkplug *B* payload format.

The majority of devices connecting into next generation IIoT infrastructures are legacy equipment using poll/response protocols. This means we must take in account register based data from devices that talk protocols like Modbus. The existing legacy equipment needs to work in concert with emerging IIoT equipment that is able to leverage message transports like MQTT natively.

6.2. Google Protocol Buffers

"Protocol Buffers are a way of encoding structured data in an efficient yet extensible format."

Google Protocol Buffers, sometimes referred to as "Google Protobufs", provide the efficiency of packed binary data encoding while providing the structure required to make it easy to create, transmit, and parse register based process variables using a standard set of tools while enabling emerging IIoT requirements around richer metadata. Google Protocol Buffers development tools are available for:

- C
- C++
- C#
- Java
- Python
- GO
- JavaScript

Additional information on Google Protocol Buffers can be found at:

https://developers.google.com/protocol-buffers/

6.3. Sparkplug A MQTT Payload Definition

As described in the Introduction Section Sparkplug A is deprecated and is not included in this document.

6.4. Sparkplug B MQTT Payload Definition

The goal of the Sparkplug is to provide a specification that both OEM device manufactures and application developers can use to create rich and interoperable SCADA/IIoT solutions using MQTT as a base messaging technology. In Sparkplug B message payload definition, the goal was to create a simple and straightforward binary message encoding that could be used primarily for legacy register based process variables (Modbus register value for example).

The Sparkplug B MQTT payload format has come about based on the feedback from many system integrators and end users who wanted to be able to natively support a much richer data model within the MQTT infrastructures that they were designing and deploying. Using the feedback from the user community Sparkplug B provides support for:

- Complex data types using templates.
- Datasets.
- Richer metrics with the ability to add property metadata for each metric.
- Metric alias support to maintain rich metric naming while keeping bandwidth usage to a minimum.
- Historical data.
- File data.

Sparkplug B definition creates a bandwidth efficient data transport for real time device data. For WAN based SCADA/IIoT infrastructures this equates to lower latency data updates while minimizing the amount of traffic and therefore cellular and/or VSAT bandwidth required. In situations where bandwidth savings is not the primary concern, the efficient use enables higher throughput of more and interesting data eliminating sensor data that may have previously been left stranded in the field. It is also ideal for LAN based SCADA infrastructures equating to higher throughput of real time data to consumer applications without requiring extreme networking topologies and/or equipment.

There are many data encoding technologies available that can all be used in conjunction with MQTT. Sparkplug B selected an existing, open, and highly available encoding scheme that efficiently encodes register based process variables. The encoding technology selected for Sparkplug B is Google Protocol Buffers.

6.4.1. Google Protocol Buffer Schema

Using lessons learned on the feedback from the Sparkplug A implementation a new Google Protocol Buffer schema was developed that could be used to represent and encode the more complex data models being requested. The entire Google Protocol Buffers definition is below.

```
// * Copyright (c) 2015-2021 Cirrus Link Solutions and others
// *
// * This program and the accompanying materials are made available under the
// * terms of the Eclipse Public License 2.0 which is available at
// * http://www.eclipse.org/legal/epl-2.0.
// *
// * SPDX-License-Identifier: EPL-2.0
// *
// * Contributors:
// *   Cirrus Link Solutions - initial implementation

//
// To compile:
// cd client_libraries/java
// protoc --proto_path=../../ --java_out=src/main/java ../../sparkplug_b.proto
//

syntax = "proto2";

package org.eclipse.tahu.protobuf;

option java_package         = "org.eclipse.tahu.protobuf";
option java_outer_classname = "SparkplugBProto";

enum DataType {
    // Indexes of Data Types

    // Unknown placeholder for future expansion.
    Unknown         = 0;

    // Basic Types
    Int8            = 1;
    Int16           = 2;
    Int32           = 3;
    Int64           = 4;
    UInt8           = 5;
    UInt16          = 6;
    UInt32          = 7;
    UInt64          = 8;
    Float           = 9;
    Double          = 10;
    Boolean         = 11;
    String          = 12;
    DateTime        = 13;
    Text            = 14;

    // Additional Metric Types
    UUID            = 15;
    DataSet         = 16;
    Bytes           = 17;
    File            = 18;
    Template        = 19;

    // Additional PropertyValue Types
    PropertySet     = 20;
    PropertySetList = 21;

    // Array Types
    Int8Array = 22;
    Int16Array = 23;
    Int32Array = 24;
    Int64Array = 25;
    UInt8Array = 26;
    UInt16Array = 27;
    UInt32Array = 28;
    UInt64Array = 29;
    FloatArray = 30;
```

```
    DoubleArray = 31;
    BooleanArray = 32;
    StringArray = 33;
    DateTimeArray = 34;
}

message Payload {

    message Template {

        message Parameter {
            optional string name        = 1;
            optional uint32 type        = 2;

            oneof value {
                uint32 int_value        = 3;
                uint64 long_value       = 4;
                float  float_value      = 5;
                double double_value     = 6;
                bool   boolean_value    = 7;
                string string_value     = 8;
                ParameterValueExtension extension_value = 9;
            }

            message ParameterValueExtension {
                extensions              1 to max;
            }
        }

        optional string version         = 1;          // The version of the Template to prevent mismatches
        repeated Metric metrics         = 2;          // Each metric includes a name, datatype, and optionally a value
        repeated Parameter parameters   = 3;
        optional string template_ref    = 4;          // Reference to a template if this is extending a Template or an
instance - MUST exist if an instance
        optional bool is_definition     = 5;
        extensions                      6 to max;
    }

    message DataSet {

        message DataSetValue {

            oneof value {
                uint32 int_value                      = 1;
                uint64 long_value                     = 2;
                float  float_value                    = 3;
                double double_value                   = 4;
                bool   boolean_value                  = 5;
                string string_value                   = 6;
                DataSetValueExtension extension_value   = 7;
            }

            message DataSetValueExtension {
                extensions  1 to max;
            }
        }

        message Row {
            repeated DataSetValue elements  = 1;
            extensions                      2 to max;   // For third party extensions
        }

        optional uint64   num_of_columns    = 1;
        repeated string   columns           = 2;
        repeated uint32   types             = 3;
        repeated Row      rows              = 4;
        extensions                          5 to max;   // For third party extensions
```

```
    }

    message PropertyValue {

        optional uint32     type                    = 1;
        optional bool       is_null                 = 2;

        oneof value {
            uint32          int_value               = 3;
            uint64          long_value              = 4;
            float           float_value             = 5;
            double          double_value            = 6;
            bool            boolean_value           = 7;
            string          string_value            = 8;
            PropertySet     propertyset_value       = 9;
            PropertySetList propertysets_value      = 10;      // List of Property Values
            PropertyValueExtension extension_value  = 11;
        }

        message PropertyValueExtension {
            extensions                              1 to max;
        }
    }

    message PropertySet {
        repeated string        keys     = 1;         // Names of the properties
        repeated PropertyValue values   = 2;
        extensions                      3 to max;
    }

    message PropertySetList {
        repeated PropertySet propertyset = 1;
        extensions                       2 to max;
    }

    message MetaData {
        // Bytes specific metadata
        optional bool   is_multi_part   = 1;

        // General metadata
        optional string content_type    = 2;        // Content/Media type
        optional uint64 size            = 3;        // File size, String size, Multi-part size, etc
        optional uint64 seq             = 4;        // Sequence number for multi-part messages

        // File metadata
        optional string file_name       = 5;        // File name
        optional string file_type       = 6;        // File type (i.e. xml, json, txt, cpp, etc)
        optional string md5             = 7;        // md5 of data

        // Catchalls and future expansion
        optional string description     = 8;        // Could be anything such as json or xml of custom properties
        extensions                      9 to max;
    }

    message Metric {

        optional string   name          = 1;        // Metric name - should only be included on birth
        optional uint64   alias         = 2;        // Metric alias - tied to name on birth and included in all later
DATA messages
        optional uint64   timestamp     = 3;        // Timestamp associated with data acquisition time
        optional uint32   datatype      = 4;        // DataType of the metric/tag value
        optional bool     is_historical = 5;        // If this is historical data and should not update real time tag
        optional bool     is_transient  = 6;        // Tells consuming clients such as MQTT Engine to not store this as a
tag
        optional bool     is_null       = 7;        // If this is null - explicitly say so rather than using -1, false,
etc for some datatypes.
        optional MetaData metadata      = 8;        // Metadata for the payload
```

```
        optional PropertySet properties = 9;

        oneof value {
            uint32   int_value                      = 10;
            uint64   long_value                     = 11;
            float    float_value                    = 12;
            double   double_value                   = 13;
            bool     boolean_value                  = 14;
            string   string_value                   = 15;
            bytes    bytes_value                    = 16;       // Bytes, File
            DataSet  dataset_value                  = 17;
            Template template_value                 = 18;
            MetricValueExtension extension_value    = 19;
        }

        message MetricValueExtension {
            extensions  1 to max;
        }
    }

    optional uint64   timestamp     = 1;        // Timestamp at message sending time
    repeated Metric   metrics       = 2;        // Repeated forever - no limit in Google Protobufs
    optional uint64   seq           = 3;        // Sequence number
    optional string   uuid          = 4;        // UUID to track message type in terms of schema definitions
    optional bytes    body          = 5;        // To optionally bypass the whole definition above
    extensions                      6 to max;   // For third party extensions
}
```

6.4.2. Payload Metric Naming Convention

For the remainder of this document JSON will be used to represent components of a Sparkplug B payload. It is important to note that the payload is a binary encoding and is not actually JSON. However, JSON representation is used in this document to represent the payloads in a way that is easy to read. For example, a simple Sparkplug B payload with a single metric can be represented in JSON as follows:

```
{
        "timestamp": <timestamp>,
        "metrics": [{
                "name": <metric_name>,
                "alias": <alias>,
                "timestamp": <timestamp>,
                "dataType": <datatype>,
                "value": <value>
        }],
        "seq": <sequence_number>
}
```

A simple Sparkplug B payload with values would be represented as follows:

```
{
        "timestamp": 1486144502122,
        "metrics": [{
                "name": "My Metric",
                "alias": 1,
                "timestamp": 1479123452194,
                "dataType": "String",
                "value": "Test"
        }],
        "seq": 2
}
```

Note that the 'name' of a metric may be hierarchical to build out proper folder structures for applications consuming the metric values. For example, in an application where an Edge Node in connected to several devices or data sources, the 'name' could represent discrete folder structures of:

'Folder 1/Folder 2/Metric Name'

Using this convention in conjunction with the **group_id**, **edge_node_id** and **device_id** already defined in the Topic Namespace, consuming applications can organize metrics in the same hierarchical fashion:

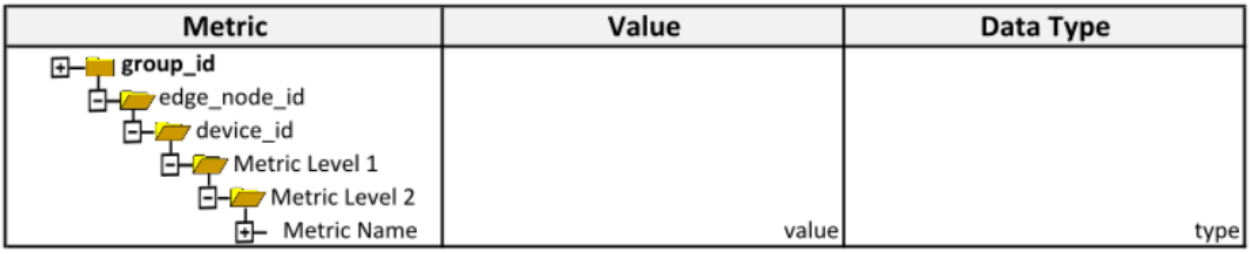

Figure 8 – Payload Metric Folder Structure

6.4.3. Sparkplug B v1.0 Payload Components

The Sparkplug specification Topics Section defines the Topic Namespace that Sparkplug uses to publish and subscribe between Edge Nodes and Host Applications within the MQTT infrastructure. Using that Topic Namespace, this section of the specification defines the actual payload contents of each message type in Sparkplug B v1.0.

6.4.4. Payload Component Definitions

Sparkplug B consists of a series of one or more metrics with metadata surrounding those metrics. The following definitions explain the components that make up a payload.

6.4.5. Payload

A Sparkplug B payload is the top-level component that is encoded and used in an MQTT message. It contains some basic information such as a timestamp and a sequence number as well as an array of

metrics which contain key/value pairs of data. A Sparkplug B payload includes the following components.

- **payload**
  - *timestamp*
    - This is the timestamp in the form of an unsigned 64-bit integer representing the number of milliseconds since epoch (Jan 1, 1970). **This timestamp MUST be in UTC.** This timestamp represents the time at which the message was published.
  - *metrics*
    - This is an array of metrics representing key/value/datatype values. Metrics are further defined here.
  - *seq*
    - This is the sequence number which is an unsigned 64-bit integer. **A sequence number MUST be included in the payload of every Sparkplug MQTT message except NDEATH messages. A NBIRTH message MUST always contain a sequence number of zero. All subsequent messages MUST contain a sequence number that is continually increasing by one in each message until a value of 255 is reached. At that point, the sequence number of the following message MUST be zero.**
  - *uuid*
    - This is a field which can be used to represent a schema or some other specific form of the message. Example usage would be to supply a UUID which represents an encoding mechanism of the optional array of bytes associated with a payload.
  - *body*
    - This is an array of bytes which can be used for any custom binary encoded data.

6.4.6. Metric

A Sparkplug B metric is a core component of data in the payload. It represents a key, value, timestamp, and datatype along with metadata used to describe the information it contains. These also represent *tags* in classic SCADA systems. It includes the following components.

- **name**
  - This is the friendly name of a metric. It should be represented as a forward-slash delimited UTF-8 string. The slashes in the string represent folders of the metric to represent hierarchical data structures. For example, 'outputs/A' would be a metric with a unique identifier of 'A' in the 'outputs' folder. There is no limit to the number of folders. However, across the infrastructure of MQTT publishers a defined folder should always remain a folder.
  - **The name MUST be included with every metric unless aliases are being used.**
  - All UTF-8 characters are allowed in the metric name. However, special characters including but not limited to the following are discouraged: . , \ @ # $ % ^ & * ( ) [ ] { } | ! ` ~ : ; ' " < > ?. This is because many Sparkplug Host Applications may have issues handling them.
- **alias**

    - This is an unsigned 64-bit integer representing an optional alias for a Sparkplug B payload. Aliases are optional and not required. **If aliases are used, the following rules apply.**
        - **If supplied in an NBIRTH or DBIRTH it MUST be a unique number across this Edge Node's entire set of metrics.**
            - Non-normative comment: no two metrics for the same Edge Node can have the same alias. Upon being defined in the NBIRTH or DBIRTH, subsequent messages can supply only the alias instead of the metric friendly name to reduce overall message size.
        - **NBIRTH and DBIRTH messages MUST include both a metric name and alias.**
        - **NDATA, DDATA, NCMD, and DCMD messages MUST only include an alias and the metric name MUST be excluded.**
- **timestamp**
    - This is the timestamp in the form of an unsigned 64-bit integer representing the number of milliseconds since epoch (Jan 1, 1970).
    - **The timestamp MUST be included with every metric in all NBIRTH, DBIRTH, NDATA, and DDATA messages.**
    - **The timestamp MAY be included with metrics in NCMD and DCMD messages.**
    - **This timestamp MUST be in UTC.**
        - Non-normative comment: This timestamp represents the time at which the value of a metric was captured.
- **datatype**
    - **This MUST be an unsigned 32-bit integer representing the datatype.**
    - **This value MUST be one of the enumerated values as shown in the valid Sparkplug Data Types.**
    - **This MUST be included with each metric definitions in NBIRTH and DBIRTH messages.**
    - **This SHOULD NOT be included with metric definitions in NDATA, NCMD, DDATA, and DCMD messages.**
- **is_historical**
    - This is a Boolean flag which denotes whether this metric represents a historical value. In some cases, it may be desirable to send metrics after they were acquired from a device or Edge Node. This can be done for batching, store and forward, or sending local backup data during network communication loses. This flag denotes that the message should not be considered a real time/current value.
- **is_transient**
    - This is a Boolean flag which denotes whether this metric should be considered transient. Transient metrics can be considered those that are of interest to a host application(s) but should not be stored in a historian.
- **is_null**
    - This is a Boolean flag which denotes whether this metric has a null value. This is Sparkplug B's mechanism of explicitly denoting a metric's value is actually null.

- **metadata**
  - This is a MetaData object associated with the metric for dealing with more complex datatypes. This is covered in the metadata section.
- **properties**
  - This is a PropertySet object associated with the metric for including custom key/value pairs of metadata associated with a metric. This is covered in the property set section.
- **value**
  - The value of a metric utilizes the 'oneof' mechanism of Google Protocol Buffers. The value supplied with a metric MUST be one of the following types. Note if the metrics is_null flag is set to true the value can be omitted altogether.
    - *uint32*
      - Defined here: https://developers.google.com/protocol-buffers/docs/proto#scalar
    - *uint64*
      - Defined here: https://developers.google.com/protocol-buffers/docs/proto#scalar
    - *float*
      - Defined here: https://developers.google.com/protocol-buffers/docs/proto#scalar
    - *double*
      - Defined here: https://developers.google.com/protocol-buffers/docs/proto#scalar
    - *bool*
      - Defined here: https://developers.google.com/protocol-buffers/docs/proto#scalar
    - *string*
      - Defined here: https://developers.google.com/protocol-buffers/docs/proto#scalar
    - *bytes*
      - Defined here: https://developers.google.com/protocol-buffers/docs/proto#scalar
    - *DataSet*
      - Defined here.
    - *Template*
      - Defined here.

6.4.7. MetaData

A Sparkplug B MetaData object is used to describe different types of binary data. These are optional and includes the following components.

- **is_multi_part**
  - A Boolean representing whether this metric contains part of a multi-part message. Breaking up large quantities of data can be useful for keeping the flow of MQTT messages flowing through the system. Because MQTT ensures in-order delivery of QoS 0 messages on the same topic, very large messages can result in messages being blocked while delivery of large

messages takes place.

- **content_type**
  - This is a UTF-8 string which represents the content type of a given metric value if applicable.
- **size**
  - This is an unsigned 64-bit integer representing the size of the metric value. This is useful when metric values such as files are sent. This field can be used for the file size.
- **seq**
  - If this is a multipart metric, this is an unsigned 64-bit integer representing the sequence number of this part of a multipart metric.
- **file_name**
  - If this is a file metric, this is a UTF-8 string representing the filename of the file.
- **file_type**
  - If this is a file metric, this is a UTF-8 string representing the type of the file.
- **md5**
  - If this is a byte array or file metric that can have a md5sum, this field can be used as a UTF-8 string to represent it.
- **description**
  - This is a freeform field with a UTF-8 string to represent any other pertinent metadata for this metric. It can contain JSON, XML, text, or anything else that can be understood by both the publisher and the subscriber.

6.4.8. PropertySet

A Sparkplug B PropertySet object is used with a metric to add custom properties to the object. The PropertySet is a map expressed as two arrays of equal size, one containing the keys and one containing the values. It includes the following components.

- **keys**
  - This is an array of UTF-8 strings representing the names of the properties in this PropertySet. **The array of keys in a PropertySet MUST contain the same number of values included in the array of PropertyValue objects.**
- **values**
  - This is an array of PropertyValue objects representing the values of the properties in the PropertySet. **The array of values in a PropertySet MUST contain the same number of items that are in the keys array.**

6.4.9. PropertyValue

A Sparkplug B PropertyValue object is used to encode the value and datatype of the value of a property in a PropertySet. It includes the following components.

- **type**

    - **This MUST be an unsigned 32-bit integer representing the datatype. This value MUST be one of the enumerated values as shown in the Sparkplug Basic Data Types or the Sparkplug Property Value Data Types. This MUST be included in Property Value Definitions in NBIRTH and DBIRTH messages.**
- **is_null**
    - This is a Boolean flag which denotes whether this property has a null value. This is Sparkplug B's mechanism of explicitly denoting a property's value is actually null.
- **value**
    - The value of a property utilizes the 'oneof' mechanism of Google Protocol Buffers. The value supplied with a metric MUST be one of the following types. Note if the metrics is_null flag is set to true the value can be omitted altogether.
        - *uint32*
            - Defined here: https://developers.google.com/protocol-buffers/docs/proto#scalar
        - *uint64*
            - Defined here: https://developers.google.com/protocol-buffers/docs/proto#scalar
        - *float*
            - Defined here: https://developers.google.com/protocol-buffers/docs/proto#scalar
        - *double*
            - Defined here: https://developers.google.com/protocol-buffers/docs/proto#scalar
        - *bool*
            - Defined here: https://developers.google.com/protocol-buffers/docs/proto#scalar
        - *string*
            - Defined here: https://developers.google.com/protocol-buffers/docs/proto#scalar
        - *PropertySet*
            - Defined here.
        - *PropertySetList*
            - Defined here.

Quality Codes

There is one specific property key in Sparkplug called *Quality*. This defines the quality of the value associated with the metric. This property is optional and is only required if the quality of the metric is not GOOD.

There are three possible quality code values. These are defined below with their associated meanings.

- **0**
    - BAD
- **192**

    - GOOD
- **500**
    - STALE

The *type* of the Property Value MUST be a value of 3 which represents a Signed 32-bit Integer.

The *value* of the Property Value MUST be an int_value and be one of the valid quality codes of 0, 192, or 500.

6.4.10. PropertySetList

A Sparkplug B PropertySetList object is an array of PropertySet objects. It includes the following components.

- **propertyset**
    - This is an array of PropetrySet objects.

6.4.11. DataSet

A Sparkplug B DataSet object is used to encode matrices of data. It includes the following components.

- **num_of_columns**
    - **This MUST be an unsigned 64-bit integer representing the number of columns in this DataSet.**
- **columns**
    - This is an array of strings representing the column headers of this DataSet. **The size of the array MUST have the same number of elements that the types array contains.**
- **types**
    - **This MUST be an array of unsigned 32 bit integers representing the datatypes of the columns. The array of types MUST have the same number of elements that the columns array contains. The values in the types array MUST be a unsigned 32-bit integer representing the datatype. This values in the types array MUST be one of the enumerated values as shown in the Sparkplug Basic Data Types. This MUST be included in DataSet Definitions in NBIRTH and DBIRTH messages.**
- **rows**
    - This is an array of DataSet.Row objects. It contains the data that makes up the data rows of this DataSet.

6.4.12. DataSet.Row

A Sparkplug B DataSet.Row object represents a row of data in a DataSet. It includes the following components.

- **elements**

- This is an array of DataSet.DataSetValue objects. It represents the data contained within a row of a DataSet.

6.4.13. DataSet.DataSetValue

- **value**
  - The value of a DataSet.DataSetValue utilizes the 'oneof' mechanism of Google Protocol Buffers. **The value supplied MUST be one of the following types: *uint32*, *uint64*, *float*, *double*, *bool*, or *string*.** More information on these types can be found below.
    - *uint32*
      - Defined here: https://developers.google.com/protocol-buffers/docs/proto#scalar
    - *uint64*
      - Defined here: https://developers.google.com/protocol-buffers/docs/proto#scalar
    - *float*
      - Defined here: https://developers.google.com/protocol-buffers/docs/proto#scalar
    - *double*
      - Defined here: https://developers.google.com/protocol-buffers/docs/proto#scalar
    - *bool*
      - Defined here: https://developers.google.com/protocol-buffers/docs/proto#scalar
    - *string*
      - Defined here: https://developers.google.com/protocol-buffers/docs/proto#scalar

6.4.14. Template

A Sparkplug B Template is used for encoding complex datatypes in a payload. It is a type of metric and can be used to create custom datatype definitions and instances. These are also sometimes referred to as *User Defined Types* or UDTs. There are two types of Templates.

- **Template Definition**
  - This is the definition of a Sparkplug Template. **A Template Definition MUST have is_definitiion set to true. A Template Definition MUST omit the template_ref field.**
- **Template Instance**
  - This is an instance of a Sparkplug Template. **A Template Instance MUST have is_definitiion set to false. A Template Instance MUST have template_ref set to the type of template definition it is.** In other words, it must be set to the name of the metric that represents the template definition.

A Sparkplug Template includes the following components.

- **version**
  - This is an optional field and can be included in a Template Definition or Template Instance. **If included, the version MUST be a UTF-8 string representing the version of the**

Template.

- **metrics**
  - This is an array of metrics representing the members of the Template. These can be primitive datatypes or other Templates as required.
- **parameters**
  - This is an option field and is an array of Parameter objects representing parameters associated with the Template.
- **template_ref**
  - **This MUST be omitted if this is a Template Definition. This MUST be a UTF-8 string representing a reference to a Template Definition name if this is a Template Instance.**
- **is_definition**
  - This is a Boolean representing whether this is a Template definition or a Template instance. **This MUST be included in every Template Definition and Template Instance. This MUST be set to true if this is a Template Definition. This MUST be set to false if this is a Template Instance.**

6.4.15. Template.Parameter

A Sparkplug B Template.Parameter is a metadata field for a Template. This can be used to represent parameters that are common across a Template Definition but the values are unique to the Template instances. It includes the following components.

- **name**
  - **This MUST be included in every Template Parameter definition. This MUST be a UTF-8 string representing the name of the Template parameter.**
- **type**
  - **This MUST be an unsigned 32-bit integer representing the datatype. This value MUST be one of the enumerated values as shown in the Sparkplug Basic Data Types. This MUST be included in Template Parameter Definitions in NBIRTH and DBIRTH messages.**
- **value**
  - The value of a template parameter utilizes the 'oneof' mechanism of Google Protocol Buffers. **The value supplied MUST be one of the following types: *uint32*, *uint64*, *float*, *double*, *bool*, or *string*.** For a template definition, this is the default value of the parameter. For a template instance, this is the value unique to that instance. More information on these types can be found below.
    - *uint32*
      - Defined here: https://developers.google.com/protocol-buffers/docs/proto#scalar
    - *uint64*
      - Defined here: https://developers.google.com/protocol-buffers/docs/proto#scalar
    - *float*
      - Defined here: https://developers.google.com/protocol-buffers/docs/proto#scalar

- *double*
  - Defined here: https://developers.google.com/protocol-buffers/docs/proto#scalar
- *bool*
  - Defined here: https://developers.google.com/protocol-buffers/docs/proto#scalar
- *string*
  - Defined here: https://developers.google.com/protocol-buffers/docs/proto#scalar

6.4.16. Data Types

Sparkplug defines the valid data types used for various Sparkplug constucts including Metric datatypes Property Value types, DataSet types, and Template Parameter types. Datatypes are represented as an enum in Google Protobufs as shown below.

```
enum DataType {
    // Indexes of Data Types

    // Unknown placeholder for future expansion.
    Unknown         = 0;

    // Basic Types
    Int8            = 1;
    Int16           = 2;
    Int32           = 3;
    Int64           = 4;
    UInt8           = 5;
    UInt16          = 6;
    UInt32          = 7;
    UInt64          = 8;
    Float           = 9;
    Double          = 10;
    Boolean         = 11;
    String          = 12;
    DateTime        = 13;
    Text            = 14;

    // Additional Metric Types
    UUID            = 15;
    DataSet         = 16;
    Bytes           = 17;
    File            = 18;
    Template        = 19;

    // Additional PropertyValue Types
    PropertySet     = 20;
    PropertySetList = 21;

    // Array Types
    Int8Array = 22;
    Int16Array = 23;
    Int32Array = 24;
    Int64Array = 25;
    UInt8Array = 26;
    UInt16Array = 27;
    UInt32Array = 28;
    UInt64Array = 29;
    FloatArray = 30;
    DoubleArray = 31;
    BooleanArray = 32;
    StringArray = 33;
    DateTimeArray = 34;
}
```

6.4.17. Datatype Details

- **Basic Types**
  - *Unknown*
    - Sparkplug enum value: 0
  - *Int8*
    - Signed 8-bit integer
    - Google Protocol Buffer Type: uint32
    - Sparkplug enum value: 1
  - *Int16*
    - Signed 16-bit integer
    - Google Protocol Buffer Type: uint32
    - Sparkplug enum value: 2
  - *Int32*
    - Signed 32-bit integer
    - Google Protocol Buffer Type: uint32
    - Sparkplug enum value: 3
  - *Int64*
    - Signed 64-bit integer
    - Google Protocol Buffer Type: uint64
    - Sparkplug enum value: 4
  - *UInt8*
    - Unsigned 8-bit integer
    - Google Protocol Buffer Type: uint32
    - Sparkplug enum value: 5
  - *UInt16*
    - Unsigned 16-bit integer
    - Google Protocol Buffer Type: uint32
    - Sparkplug enum value: 6
  - *UInt32*
    - Unsigned 32-bit integer
    - Google Protocol Buffer Type: uint32
    - Sparkplug enum value: 7
  - *UInt64*
    - Unsigned 64-bit integer

        - Google Protocol Buffer Type: uint64
        - Sparkplug enum value: 8
    - *Float*
        - 32-bit floating point number
        - Google Protocol Buffer Type: float
        - Sparkplug enum value: 9
    - *Double*
        - 64-bit floating point number
        - Google Protocol Buffer Type: double
        - Sparkplug enum value: 10
    - *Boolean*
        - Boolean value
        - Google Protocol Buffer Type: bool
        - Sparkplug enum value: 11
    - *String*
        - String value (UTF-8)
        - Google Protocol Buffer Type: string
        - Sparkplug enum value: 12
- *DateTime*
    - Date time value as uint64 value representing milliseconds since epoch (Jan 1, 1970)
    - Google Protocol Buffer Type: uint64
    - Sparkplug enum value: 13
- *Text*
    - String value (UTF-8)
    - Google Protocol Buffer Type: string
    - Sparkplug enum value: 14
- **Additional Types**
    - *UUID*
        - UUID value as a UTF-8 string
        - Google Protocol Buffer Type: string
        - Sparkplug enum value: 15
    - *DataSet*
        - DataSet as defined here
        - Google Protocol Buffer Type: none – defined in Sparkplug

        - Sparkplug enum value: 16
    - *Bytes*
        - Array of bytes
        - Google Protocol Buffer Type: bytes
        - Sparkplug enum value: 17
    - *File*
        - Array of bytes representing a file
        - Google Protocol Buffer Type: bytes
        - Sparkplug enum value: 18
    - *Template*
        - Template as defined here
        - Google Protocol Buffer Type: none – defined in Sparkplug
        - Sparkplug enum value: 19
- **Additional PropertyValue Types**
    - *PropertySet*
        - PropertySet as defined here
        - Google Protocol Buffer Type: none – defined in Sparkplug
        - Sparkplug enum value: 20
    - *PropertySetList*
        - PropertySetList as defined here
        - Google Protocol Buffer Type: none – defined in Sparkplug
        - Sparkplug enum value: 21
- **Array Types**
    - *Int8Array*
        - Int8Array as an array of packed little endian int8 bytes
        - Google Protocol Buffer Type: bytes
        - Sparkplug enum value: 22
    - *Int16Array*
        - Int16Array as an array of packed little endian int16 bytes
        - Google Protocol Buffer Type: bytes
        - Sparkplug enum value: 23
    - *Int32Array*
        - Int8Array as an array of packed little endian int32 bytes
        - Google Protocol Buffer Type: bytes

        - Sparkplug enum value: 24
    - *Int64Array*
        - Int8Array as an array of packed little endian int64 bytes
        - Google Protocol Buffer Type: bytes
        - Sparkplug enum value: 25
    - *UInt8Array*
        - UInt8Array as an array of packed little endian uint8 bytes
        - Google Protocol Buffer Type: bytes
        - Sparkplug enum value: 26
    - *UInt16Array*
        - UInt16Array as an array of packed little endian uint16 bytes
        - Google Protocol Buffer Type: bytes
        - Sparkplug enum value: 27
    - *UInt32Array*
        - UInt32Array as an array of packed little endian uint32 bytes
        - Google Protocol Buffer Type: bytes
        - Sparkplug enum value: 28
    - *UInt64Array*
        - UInt64Array as an array of packed little endian uint64 bytes
        - Google Protocol Buffer Type: bytes
        - Sparkplug enum value: 29
    - *FloatArray*
        - FloatArray as an array of packed little endian 32-bit float bytes
        - Google Protocol Buffer Type: bytes
        - Sparkplug enum value: 30
    - *DoubleArray*
        - DoubleArray as an array of packed little endian 64-bit float bytes
        - Google Protocol Buffer Type: bytes
        - Sparkplug enum value: 31
    - *BooleanArray*
        - BooleanArray as an array of packed little endian boolean/bit bytes
        - Google Protocol Buffer Type: bytes
        - Sparkplug enum value: 32
    - *StringArray*
        - StringArray as an array of packed little endian bytes

        - Google Protocol Buffer Type: bytes
        - Sparkplug enum value: 33
    - *DateTimeArray*
        - DateTimeArray as an array of packed little endian bytes where each date/time value is an 8-byte value representing the number of milliseconds since epoch in UTC
        - Google Protocol Buffer Type: bytes
        - Sparkplug enum value: 34

6.4.18. Payload Representation on Host Applications

Sparkplug B payloads in conjunction with the Sparkplug topic namespace result in hierarchical data structures that can be represented in folder structures with metrics which are often called tags.

6.4.19. NBIRTH

The NBIRTH is responsible for informing host applications of all of the information about the Edge Node. This includes every metric it will publish data for in the future.

- **NBIRTH messages MUST include a payload timestamp that denotes the time at which the message was published.**
- **Every Edge Node Descriptor in any Sparkplug infrastructure MUST be unique in the system.** These are used like addresses and need to be unique as a result.
- **Every NBIRTH message MUST include a sequence number and it MUST have a value of 0.**
- **Every NBIRTH message MUST include a bdSeq number metric.**
- **Every NBIRTH message in a new Sparkplug MQTT session SHOULD include a bdSeq number value that is one greater than the previous NBIRTH's bdSeq number. This value MUST never exceed 255. If in the previous NBIRTH a value of 255 was sent, the next NBIRTH MUST have a value of 0.**
- **Every NBIRTH MUST include a metric with the name *Node Control/Rebirth* and have a boolean value of false.** This is used by Host Applications to force an Edge Node to send a new birth sequence (NBIRTH and DBIRTH messages) if errors are detected by the Host Application in the data stream.
- **NBIRTH messages MUST be published with the MQTT QoS set to 0.**
- **NBIRTH messages MUST be published with the MQTT retain flag set to false.**

The following is a representation of a simple NBIRTH message on the topic:

```
spBv1.0/Sparkplug B Devices/NBIRTH/Raspberry Pi
```

In the topic above the following information is known based on the Sparkplug topic definition:

- The 'Group ID' is: Sparkplug B Devices

- The ‘Edge Node ID’ is: Raspberry Pi
- The *Edge Node Descriptor* is the combination of the Group ID and Edge Node ID.
- This is an NBIRTH message based on the *NBIRTH* Sparkplug Verb

Consider the following Sparkplug B payload in the NBIRTH message shown above:


```
{
        "timestamp": 1486144502122,
        "metrics": [{
                "name": "bdSeq",
                "timestamp": 1486144502122,
                "dataType": "Uint64",
                "value": 0
        }, {
                "name": "Node Control/Reboot",
                "timestamp": 1486144502122,
                "dataType": "Boolean",
                "value": false
        }, {
                "name": "Node Control/Rebirth",
                "timestamp": 1486144502122,
                "dataType": "Boolean",
                "value": false
        }, {
                "name": "Node Control/Next Server",
                "timestamp": 1486144502122,
                "dataType": "Boolean",
                "value": false
        }, {
                "name": "Node Control/Scan Rate",
                "timestamp": 1486144502122,
                "dataType": "Int64",
                "value": 3000
        }, {
                "name": "Properties/Hardware Make",
                "timestamp": 1486144502122,
                "dataType": "String",
                "value": "Raspberry Pi"
        }, {
                "name": "Properties/Hardware Model",
                "timestamp": 1486144502122,
                "dataType": "String",
                "value": "Pi 3 Model B"
        }, {
                "name": "Properties/OS",
                "timestamp": 1486144502122,
                "dataType": "String",
                "value": "Raspbian"
        }, {
                "name": "Properties/OS Version",
```

```
                "timestamp": 1486144502122,
                "dataType": "String",
                "value": "Jessie with PIXEL/11.01.2017"
        }, {
                "name": "Supply Voltage",
                "timestamp": 1486144502122,
                "dataType": "Float",
                "value": 12.1
        }],
        "seq": 0
}
```

This would result in a structure as follows on the Host Application.

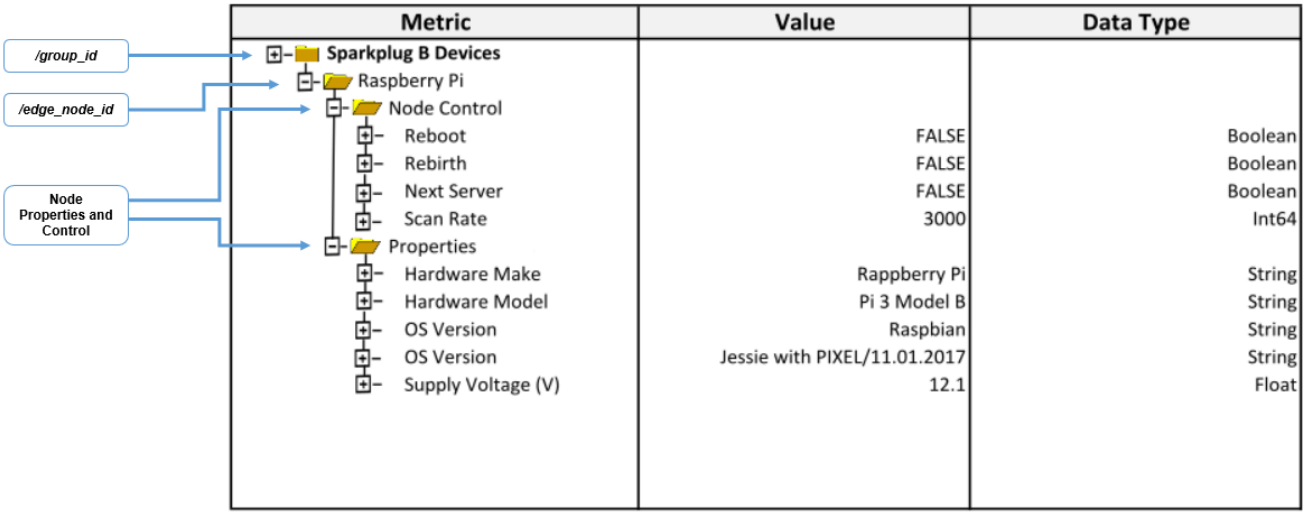

Figure 9 – Sparkplug B Metric Structure 1

6.4.20. DBIRTH

The DBIRTH is responsible for informing the Host Application of all of the information about the device. This includes every metric it will publish data for in the future.

- **DBIRTH messages MUST include a payload timestamp that denotes the time at which the message was published.**
- **Every DBIRTH message MUST include a sequence number.**
- **Every DBIRTH message MUST include a sequence number value that is one greater than the previous sequence number sent by the Edge Node. This value MUST never exceed 255. If in the previous sequence number sent by the Edge Node was 255, the next sequence number sent MUST have a value of 0.**
- **All DBIRTH messages sent by and Edge Node MUST be sent immediately after the NBIRTH and before any NDATA or DDATA messages are published by the Edge Node.**
- **DBIRTH messages MUST be published with the MQTT QoS set to 0.**
- **DBIRTH messages MUST be published with the MQTT retain flag set to false.**

The following is a representation of a simple DBIRTH message on the topic:

```
spBv1.0/Sparkplug B Devices/DBIRTH/Raspberry Pi/Pibrella
```

In the topic above the following information is known based on the Sparkplug topic definition:

- The ‘Group ID’ is: Sparkplug B Devices
- The ‘Edge Node ID’ is: Raspberry Pi
- The ‘Device ID’ is: Pibrella
- This is an DBIRTH message based on the *DBIRTH* Sparkplug Verb

Consider the following Sparkplug B payload in the DBIRTH message shown above:

```
{
        "timestamp": 1486144502122,
        "metrics": [{
                "name": "Inputs/A",
                "timestamp": 1486144502122,
                "dataType": "Boolean",
                "value": false
        }, {
                "name": "Inputs/B",
                "timestamp": 1486144502122,
                "dataType": "Boolean",
                "value": false
        }, {
                "name": "Inputs/C",
                "timestamp": 1486144502122,
                "dataType": "Boolean",
                "value": false
        }, {
                "name": "Inputs/D",
                "timestamp": 1486144502122,
                "dataType": "Boolean",
                "value": false
        }, {
                "name": "Inputs/Button",
                "timestamp": 1486144502122,
                "dataType": "Boolean",
                "value": false
        }, {
                "name": "Outputs/E",
                "timestamp": 1486144502122,
                "dataType": "Boolean",
                "value": false
        }, {
                "name": "Outputs/F",
                "timestamp": 1486144502122,
```

```
                "dataType": "Boolean",
                "value": false
        }, {
                "name": "Outputs/G",
                "timestamp": 1486144502122,
                "dataType": "Boolean",
                "value": false
        }, {
                "name": "Outputs/H",
                "timestamp": 1486144502122,
                "dataType": "Boolean",
                "value": false
        }, {
                "name": "Outputs/LEDs/Green",
                "timestamp": 1486144502122,
                "dataType": "Boolean",
                "value": false
        }, {
                "name": "Outputs/LEDs/Red",
                "timestamp": 1486144502122,
                "dataType": "Boolean",
                "value": false
        }, {
                "name": "Outputs/LEDs/Yellow",
                "timestamp": 1486144502122,
                "dataType": "Boolean",
                "value": false
        }, {
                "name": "Outputs/Buzzer",
                "timestamp": 1486144502122,
                "dataType": "Boolean",
                "value": false
        }, {
                "name": "Properties/Hardware Make",
                "timestamp": 1486144502122,
                "dataType": "String",
                "value": "Pibrella"
        }],
        "seq": 1
}
```

This would result in a structure as follows on the Host Application.

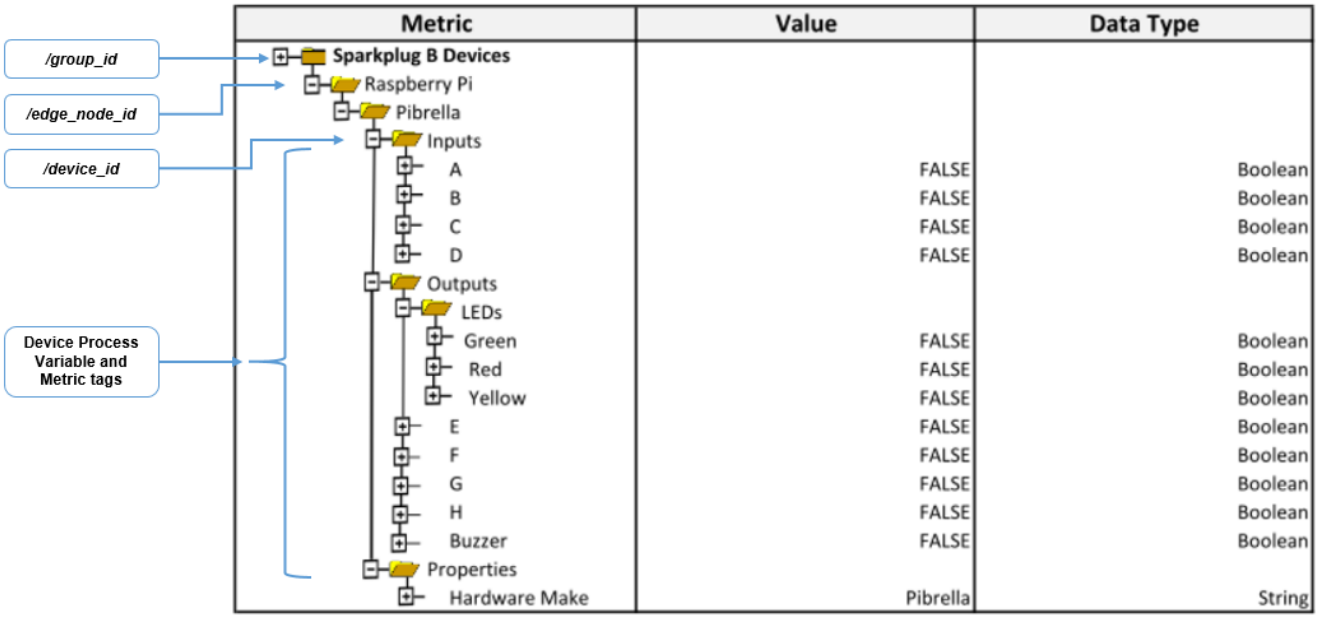

Figure 10 – Sparkplug B Metric Structure 2

6.4.21. NDATA

NDATA messages are used to update the values of any Edge Node metrics that were originally published in the NBIRTH message. Any time an input changes on the Edge Node, a NDATA message should be generated and published to the MQTT Server. If multiple metrics on the Edge Node change, they can all be included in a single NDATA message. It is also important to note that changes can be aggregated and published together in a single NDATA message. Because the Sparkplug B payload uses an ordered List of metrics, multiple different change events for multiple different metrics can all be included in a single NDATA message.

- **NDATA messages MUST include a payload timestmp that denotes the time at which the message was published.**
- **Every NDATA message MUST include a sequence number.**
- **Every NDATA message MUST include a sequence number value that is one greater than the previous sequence number sent by the Edge Node. This value MUST never exceed 255. If in the previous sequence number sent by the Edge Node was 255, the next sequence number sent MUST have a value of 0.**
- **All NDATA messages sent by and Edge Node MUST NOT be sent until all the NBIRTH and all DBIRTH messages have been published by the Edge Node.**
- **NDATA messages MUST be published with the MQTT QoS set to 0.**
- **NDATA messages MUST be published with the MQTT retain flag set to false.**

The following is a representation of a simple NDATA message on the topic:

```
spBv1.0/Sparkplug B Devices/NDATA/Raspberry Pi
```

In the topic above the following information is known based on the Sparkplug topic definition:

- The ‘Group ID’ is: Sparkplug B Devices
- The ‘Edge Node ID’ is: Raspberry Pi
- This is an NDATA message based on the *NDATA* Sparkplug Verb

Consider the following Sparkplug B payload in the NDATA message shown above:

```
{
        "timestamp": 1486144502122,
        "metrics": [{
                "name": "Supply Voltage",
                "timestamp": 1486144502122,
                "dataType": "Float",
                "value": 12.3
        }],
        "seq": 2
}
```

This would result in the host application updating the value of the *Supply Voltage* metric.

6.4.22. DDATA

DDATA messages are used to update the values of any device metrics that were originally published in the DBIRTH message. Any time an input changes on the device, a DDATA message should be generated and published to the MQTT Server. If multiple metrics on the device change, they can all be included in a single DDATA message. It is also important to note that changes can be aggregated and published together in a single DDATA message. Because the Sparkplug B payload uses an ordered List of metrics, multiple different change events for multiple different metrics can all be included in a single DDATA message.

- **DDATA messages MUST include a payload timestamp that denotes the time at which the message was published.**
- **Every DDATA message MUST include a sequence number.**
- **Every DDATA message MUST include a sequence number value that is one greater than the previous sequence number sent by the Edge Node. This value MUST never exceed 255. If in the previous sequence number sent by the Edge Node was 255, the next sequence number sent MUST have a value of 0.**
- **All DDATA messages sent by and Edge Node MUST NOT be sent until all the NBIRTH and all DBIRTH messages have been published by the Edge Node.**
- **DDATA messages MUST be published with the MQTT QoS set to 0.**
- **DDATA messages MUST be published with the MQTT retain flag set to false.**

The following is a representation of a simple DDATA message on the topic:

spBv1.0/Sparkplug B Devices/DDATA/Raspberry Pi/Pibrella

- The ‘Group ID’ is: Sparkplug B Devices

- The ‘Edge Node ID’ is: Raspberry Pi
- The ‘Device ID’ is: Pibrella
- This is an DDATA message based on the *NDATA* Sparkplug Verb

Consider the following Sparkplug B payload in the DDATA message shown above:

```
{
        "timestamp": 1486144502122,
        "metrics": [{
                "name": "Inputs/A",
                "timestamp": 1486144502122,
                "dataType": "Boolean",
                "value": true
        }, {
                "name": "Inputs/C",
                "timestamp": 1486144502122,
                "dataType": "Boolean",
                "value": true
        }],
        "seq": 0
}
```

This would result in the Host Application updating the value of the ‘Inputs/A’ metric and ‘Inputs/C’ metric.

6.4.23. NCMD

NCMD messages are used by Host Applications to write to Edge Node outputs and send Node Control commands to Edge Nodes. Multiple metrics can be supplied in a single NCMD message.

- **NCMD messages MUST include a payload timestamp that denotes the time at which the message was published.**
- **Every NCMD message MUST NOT include a sequence number.**
- **NCMD messages MUST be published with the MQTT QoS set to 0.**
- **NCMD messages MUST be published with the MQTT retain flag set to false.**

The following is a representation of a simple NCMD message on the topic:

```
spBv1.0/Sparkplug B Devices/NCMD/Raspberry Pi
```

- The ‘Group ID’ is: Sparkplug B Devices
- The ‘Edge Node ID’ is: Raspberry Pi
- This is an NCMD message based on the *NDATA* Sparkplug Verb

Consider the following Sparkplug B payload in the NCMD message shown above:

```
{
        "timestamp": 1486144502122,
        "metrics": [{
                "name": "Node Control/Rebirth",
                "timestamp": 1486144502122,
                "dataType": "Boolean",
                "value": true
        }]
}
```

This NCMD payload tells the Edge Node to republish its NBIRTH and DBIRTH(s) messages. This can be requested if a Host Application gets an out of order seq number or if a metric arrives in an NDATA or DDATA message that was not provided in the original NBIRTH or DBIRTH messages.

6.4.24. DCMD

DCMD messages are used by Host Applications to write to device outputs and send Device Control commands to devices. Multiple metrics can be supplied in a single DCMD message.

- **DCMD messages MUST include a payload timestamp that denotes the time at which the message was published.**
- **Every DCMD message MUST NOT include a sequence number.**
- **DCMD messages MUST be published with the MQTT QoS set to 0.**
- **DCMD messages MUST be published with the MQTT retain flag set to false.**

The following is a representation of a simple DCMD message on the topic:

```
spBv1.0/Sparkplug B Devices/DCMD/Raspberry Pi/Pibrella
```

- The ‘Group ID’ is: Sparkplug B Devices
- The ‘Edge Node ID’ is: Raspberry Pi
- The ‘Device ID’ is: Pibrella
- This is an DCMD message based on the *DCMD* Sparkplug Verb

Consider the following Sparkplug B payload in the DCMD message shown above:

```
{
        "timestamp": 1486144502122,
        "metrics": [{
                "name": "Outputs/LEDs/Green",
                "timestamp": 1486144502122,
                "dataType": "Boolean",
                "value": true
        }, {
                "name": "Outputs/LEDs/Yellow",
                "timestamp": 1486144502122,
                "dataType": "Boolean",
                "value": true
        }]
}
```

The DCMD payload tells the Edge Node to write true to the attached device's green and yellow LEDs. As a result, the LEDs should turn on and result in a DDATA message back to the MQTT Server after the LEDs are successfully turned on.

6.4.25. NDEATH

The NDEATH messages are registered with the MQTT Server in the MQTT CONNECT packet as the *Will Message*. This is used by Host Applications to know when an Edge Node has lost its MQTT connection with the MQTT Server.

- **Every NDEATH message MUST NOT include a sequence number.**
- **An NDEATH message MUST be registered as a Will Message in the MQTT CONNECT packet.**
- **The NDEATH message MUST set the MQTT Will QoS to 1 in the MQTT CONNECT packet.**
- **The NDEATH message MUST set the MQTT Will Retained flag to false in the MQTT CONNECT packet.**
- **The NDEATH message MUST include the same bdSeq number value that will be used in the associated NBIRTH message.** This is used by Host Applications to correlate the NDEATH messages with a previously received NBIRTH message.
- **An NDEATH message SHOULD be published by the Edge Node before it intentionally disconnects from the MQTT Server.** This allows host applications advanced notice that an Edge Node has disconnected rather than waiting for the NDEATH to be delivered by the MQTT Server based on an MQTT keep alive timeout.
- An NDEATH message MAY include a timestamp.
    - It should be noted that this timestamp is typically set at the time of the MQTT CONNECT message and as a result may not be useful to Host Applications. If the timestamp is set, Host Applications SHOULD NOT use it to determine corresponding NBIRTH messages. Instead, the bdSeq number used in the NBIRTH and NDEATH messages MUST be used to determine that an NDEATH matches a prior NBIRTH.

The following is a representation of a NDEATH message on the topic:

```
spBv1.0/Sparkplug B Devices/NDEATH/Raspberry Pi
```

- The ‘Group ID’ is: Sparkplug B Devices
- The ‘Edge Node ID’ is: Raspberry Pi
- This is an NDEATH message based on the *NDEATH* Sparkplug Verb

Consider the following Sparkplug B payload in the NDEATH message shown above:

```
{
        "timestamp": 1486144502122,
        "metrics": [{
                "name": "bdSeq",
                "timestamp": 1486144502122,
                "dataType": "UInt64",
                "value": 0
        }]
}
```

The payload metric named bdSeq allows a Host Application to reconcile this NDEATH with the NBIRTH that occurred previously.

6.4.26. DDEATH

The DDEATH messages are published by an Edge Node on behalf of an attached device. If the Edge Node determines that a device is no longer accessible (i.e. it has turned off, stopped responding, etc.) the Edge Node should publish a DDEATH to denote that device connectivity has been lost.

- **DDEATH messages MUST include a payload timestamp that denotes the time at which the message was published.**
- **Every NDATA message MUST include a sequence number.**
- **Every NDATA message MUST include a sequence number value that is one greater than the previous sequence number sent by the Edge Node. This value MUST never exceed 255. If in the previous sequence number sent by the Edge Node was 255, the next sequence number sent MUST have a value of 0.**

The following is a representation of a simple DDEATH message on the topic:

```
spBv1.0/Sparkplug B Devices/DDEATH/Raspberry Pi/Pibrella
```

- The ‘Group ID’ is: Sparkplug B Devices
- The ‘Edge Node ID’ is: Raspberry Pi
- The ‘Device ID’ is: Pibrella
- This is an DDEATH message based on the *DDEATH* Sparkplug Verb

Consider the following Sparkplug B payload in the DDEATH message shown above:

```
{
        "timestamp": 1486144502122,
        "seq": 123
}
```

A sequence number MUST be included with the DDEATH messages so the Host Application can ensure order of messages and maintain the state of the data.

6.4.27. STATE

As noted previously, the STATE messages published by Sparkplug Host Applications do not use Sparkplug B payloads. State messages are used by Sparkplug Host Applications to denote to Edge Nodes whether or not the Sparkplug Host Application is online and operational or not.

- **Sparkplug Host Applications MUST register a Will Message in the MQTT CONNECT packet on the topic *STATE/[sparkplug_host_id]*.** The [sparkplug_host_id] should be replaced with the Sparkplug Host Application's ID. This can be any UTF-8 string.
- **The Sparkplug Host Application MUST set the the MQTT Will QoS to 1 in the MQTT CONNECT packet.**
- **The Sparkplug Host Application MUST set the Will Retained flag to true in the MQTT CONNECT packet.**
- **The Sparkplug Host Application MUST set the Will Payload to the UTF-8 string of *OFFLINE* in the MQTT CONNECT packet.**
- **After establishing an MQTT connection, the Sparkplug Host Application MUST subscribe on it's own *STATE/[sparkplug_host_id]* topic.**
  - Non-normative comment: This allows the Sparkplug Host Application handle timing issues around STATE *OFFLINE* messages being published on it's behalf by the MQTT Server when it is in fact online.
- **After subscribing on it's own STATE/[sparkplug_host_id] topic, the Sparkplug Host Application MUST publish an MQTT message on the topic *STATE/[sparkplug_host_id]* with a payload of the UTF-8 string of *ONLINE*, a QoS of 1, and the retain flag set to true.** The [sparkplug_host_id] should be replaced with the Sparkplug Host Application's ID. This can be any UTF-8 string.

Chapter 7. Security

This chapter is provided for guidance only and is non-normative.

7.1. TLS

The MQTT specification does not specify any TCP/IP security scheme as it was envisaged that TCP/IP security would (and did) change over time. Although this document will not specify any TCP/IP specific security requirements it will provide guidelines on how to secure a Sparkplug infrastructure.

7.2. Authentication

There are several levels of security and access control configured within an MQTT infrastructure. From a pure MQTT client perspective, the client does need to provide a unique Client ID, and an optional username and password.

7.3. Authorization

Although access control is not mandated in the MQTT specification for use in MQTT Server implementations, Access Control List (ACL) functionality is available for many MQTT Server implementations. The ACL of an MQTT Server implementation is used to specify which MQTT topics any MQTT Client can subscribe and/or publish on. Examples are provided on how to setup and manage MQTT Client credentials and some considerations on setting up proper ACL's on the MQTT Servers.

7.4. Implementation Notes

7.4.1. Underlying MQTT Security

All aspects specified in the MQTT Specification's link:http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/os/mqtt-v3.1.1-os.html#_Toc398718111[Security Section] should be considered when implementing a Sparkplug solution.

7.4.2. Encrypted Sockets

When using public networks and data is sensitive, the underlying socket connections being used by Sparkplug components should be encrypted. This can be done using link:https://datatracker.ietf.org/doc/html/rfc5246[TLS] or potential future mechanisms for securing and encrypting the underlying TCP/IP connection.

7.4.3. Access Control Lists

ACLs can be defined for Sparkplug clients to restrict each Edge Node to a specific set of topics it can publish and subscribe on. Many MQTT Servers offer the ability to configure ACLs based on client connection credentials. When supported by the MQTT Server, ACLs offer some security in

preventing compromised credentials from being able to be used to spoof Edge Nodes, write to other Edge Node outputs, or see all messages flowing through the MQTT Server. Consider the Edge Node with a single attached Device with the following Sparkplug IDs.

```
Group ID = G1
Edge Node ID = E1
Device ID = D1
```

Based on this, the following could be reasonable MQTT ACLs to be provisioned in the MQTT Server for the MQTT client associated with this Edge Node:

```
Publish: spBv1.0/G1/NBIRTH/E1
Publish: spBv1.0/G1/NDATA/E1
Publish: spBv1.0/G1/NDEATH/E1
Publish: spBv1.0/G1/DBIRTH/E1/D1
Publish: spBv1.0/G1/DDATA/E1/D1
Publish: spBv1.0/G1/DDEATH/E1/D1
Subscribe: STATE/my_primary_host
Subscribe: spBv1.0/G1/NCMD/E1
Subscribe: spBv1.0/G1/DCMD/E1/D1
```

However, there may be other considerations when creating ACLs for clients. It may be the case that an Edge Node has many dynamic associated devices. In this case, it may make sense to wildcard the device level topic token. For example, it could look like this:

```
Publish: spBv1.0/G1/NBIRTH/E1
Publish: spBv1.0/G1/NDATA/E1
Publish: spBv1.0/G1/NDEATH/E1
Publish: spBv1.0/G1/DBIRTH/E1/+
Publish: spBv1.0/G1/DDATA/E1/+
Publish: spBv1.0/G1/DDEATH/E1/+
Subscribe: STATE/my_primary_host
Subscribe: spBv1.0/G1/NCMD/E1
Subscribe: spBv1.0/G1/DCMD/E1/+
```

Also, it may be the case that DCMD messages should not be *writable*. In this case, maybe DCMD subscriptions should not be allowed at all. In this case, the ACLs could look like this:

```
Publish: spBv1.0/G1/NBIRTH/E1
Publish: spBv1.0/G1/NDATA/E1
Publish: spBv1.0/G1/NDEATH/E1
Publish: spBv1.0/G1/DBIRTH/E1/+
Publish: spBv1.0/G1/DDATA/E1/+
Publish: spBv1.0/G1/DDEATH/E1/+
Subscribe: STATE/my_primary_host
Subscribe: spBv1.0/G1/NCMD/E1
```

By using ACLs in this way, the access each Edge Node has is restricted to only topics that it should be able to publish and subscribe on. If the client credentials for some Edge Node were to be compromised, the potential harm that could be done by a malicious client would be limited in scope. For example, a client would not be able to appear to be as some other client. Subscribing on # would not be allowed so the full scope of a Sparkplug topic namespace could not be realized by the malicious client with the compromised credentials.

Chapter 8. High Availability

Sparkplug™ based infrastructures are often used in mission-critical environments. Planning for high availability is a key requirement for many Sparkplug users. This section discusses non-normative approaches to achieving high availability.

8.1. High Availability for MQTT Servers

A core component of MQTT based infrastructures is the MQTT Server. It is the central data broker and together with the Primary Host Application a potential single point of failure. All components are connected to the MQTT Server all the time and a failure of the MQTT Server will cause unavailability of the whole infrastructure.

There are two options for MQTT Server High Availability in Sparkplug:

1. MQTT Server HA Clustering
2. Multiple isolated MQTT Servers

Both approaches have been deployed successfully in mission critical environments and depending on the MQTT Server software used, not all options might be available.

8.1.1. MQTT Server HA Clustering (non-normative)

A single MQTT Server is a single point of failure in a Sparkplug infrastructure, which means a failure of the Server will cause a downtime for all other components.

MQTT Servers that support clustering allow to install multiple MQTT Servers and connect them to a cluster. This means all relevant MQTT data is synchronized between these servers. If one or multiple MQTT Server fail, all data is still present at the other MQTT Servers.

The main advantage of MQTT Server clusters is operations simplicity. Sparkplug components don't need to distribute state themselves between MQTT Servers (which is required for Multiple MQTT Server Topologies). From the perspective of an MQTT Edge node, MQTT enabled Device, or Sparkplug Host Application, any of the MQTT Servers can be used and devices are not required to be connected to the same MQTT Server. A MQTT cluster provides the illusion to MQTT clients that there is only one MQTT server while providing High Availability.

There are two options available for deploying HA MQTT Server clusters:

1. High Availability Cluster without Load Balancer
2. High Availability Cluster with Load Balancer

8.1.2. High Availability Cluster

In traditional clustered MQTT Server settings, each MQTT Server is reachable by all MQTT Edge Nodes, Sparkplug Host Applications, and MQTT enabled Devices. Each component can connect directly to any MQTT Server. A message sent to any MQTT Server will be distributed to all available MQTT Servers in the cluster (which will distribute the message to all subscribing Sparkplug

components).

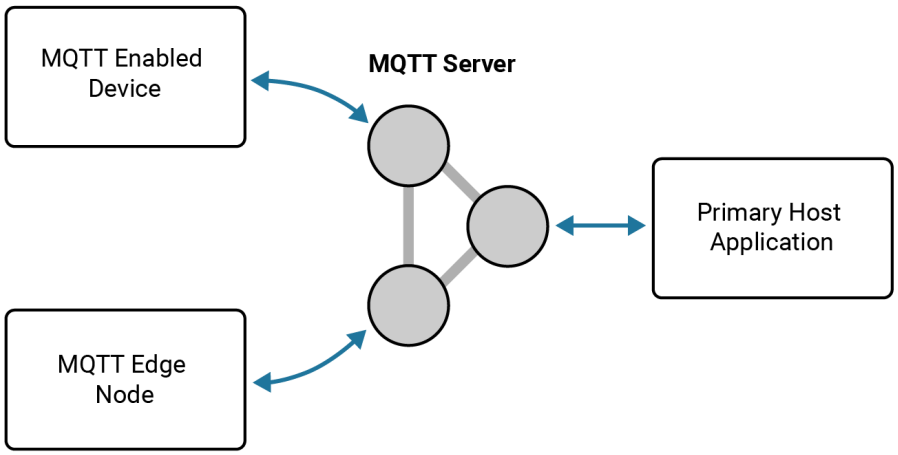

Figure X – High Availability MQTT Server Cluster

If any MQTT Server would fail, the MQTT connection for components connected to the broker will break and the component can connect to any other MQTT Server to resume operations.

8.1.3. High Availability Cluster with Load Balancer

For dynamic environments where the IP addresses of the MQTT Servers might not be available beforehand (like in cloud native deployment environments such as Kubernetes) or for cases where it's not desired that all IP addresses (or DNS lookup names) are configured on the Sparkplug components, a load balancer might be used.

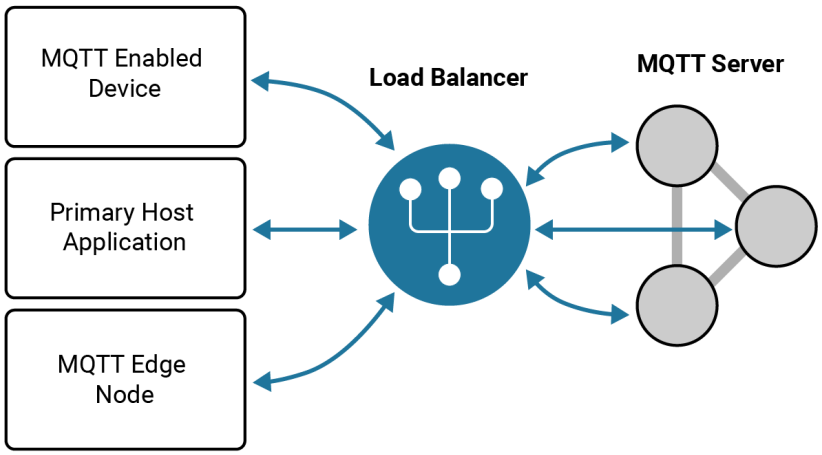

Figure X – High Availability MQTT Server Cluster with Load Balancer

A load balancer acts as the single point of contact for Sparkplug components, so only a single IP address or DNS name needs to be configured on the components. The load balancer will proxy the MQTT connections of the components and route to one available MQTT Server. In case of a MQTT Server failure, the component only needs to reconnect to the load balancer again.

The use of a specific load balancer depends on the MQTT Server used. Usually most load balancers work with most Sparkplug compatible MQTT Servers on the market.

8.2. Multiple Isolated MQTT Servers (non-normative)

A second approach to high availability is the use of several isolated MQTT Servers. This approach works with all Sparkplug certified MQTT Servers and does not need cluster technology but requires Primary Host Applications that support multiple isolated MQTT brokers. The *Primary Host Application* is responsible for managing state across the several MQTT brokers.

When multiple MQTT Servers are available there is the possibility of “stranding” and EoN node if the Primary command/control of the *Primary Host Application* loses network connectivity to one of the MQTT Servers. In this instance the EoN node would stay properly connected to the MQTT Server publishing information not knowing that *Primary Host Application* was not able to receive the messages. When using multiple MQTT Servers, the *Primary Host Application* instance must be configured to publish a STATE Birth Certificate and all EoN nodes need to subscribe to this STATE message.

The *Primary Host Application* will need to specify whether it is a “Primary” command/control instance or not. If it is a primary instance then every time it establishes a new MQTT Session with an MQTT Server, the STATE Birth Certificate defined in section above is the first message that is published after a successful MQTT Session is established.

EoN node devices in an infrastructure that provides multiple MQTT Servers can establish a session to any one of the MQTT Servers. Upon establishing a session, the EoN node should issue a subscription to the STATE message published by *Primary Host Application*. Since the STATE message is published with the RETAIN message flag set, MQTT will guarantee that the last STATE message is always available. The EoN node should examine the payload of this message to ensure that it is a value of “ONLINE”. If the value is “OFFLINE”, this indicates the Primary Application has lost its MQTT Session to this particular MQTT Server. This should cause the EoN node to terminate its session with this MQTT Server and move to the next available MQTT Server that is available. This use of the STATE message in this manner ensures that any loss of connectivity to an MQTT Server to the *Primary Host Application* does not result in EoN nodes being “stranded” on an MQTT server because of network issues. The following message flow diagram outlines how the STATE message is used when three (3) MQTT Servers are available in the infrastructure:

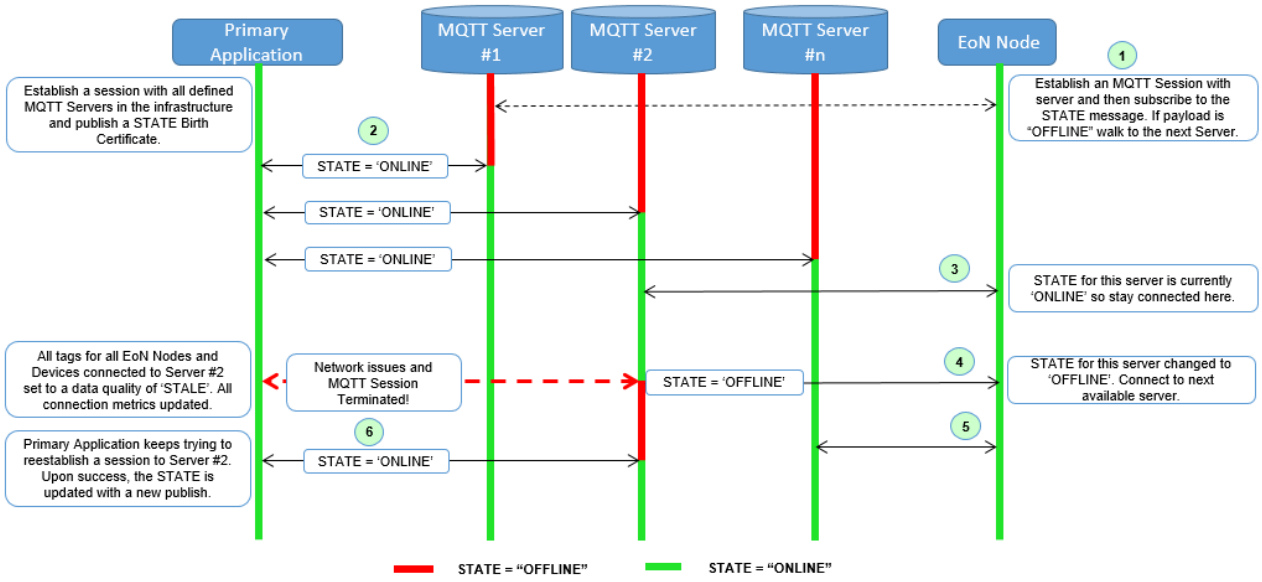

Figure 7 – Primary Application STATE flow diagram

1. When an EoN node is configured with multiple available MQTT Servers in the infrastructure it should issue a subscription to the *Primary Host Application* STATE message. The EoN nodes are free to establish an MQTT Session to any of the available servers over any available network at any time and examine the current STATE value. If the STATE message payload is ‘OFFLINE’ then the EoN node should disconnect and walk to the next available server.
2. Upon startup, the configured Primary Application, the MQTT Session will be configured to register the *Primary Host Application* DEATH Certificate that indicates STATE is ‘OFFLINE’ with the message RETAIN flag set to true. Then the *Primary Host Application* BIRTH Certificate will be published with a STATE payload of ‘ONLINE’.
3. As the EoN node walks its available MQTT Server table, it will establish an MQTT Session with a server that has a STATE message with a payload of ‘ONLINE’. The EoN node can stay connected to this server if its MQTT Session stays intact and it does not receive the *Primary Host Application* DEATH Certificate.
4. Having a subscription registered to the MQTT Server on the STATE topic will result in any change to the current the *Primary Host Application* STATE being received immediately. In this case, a network disruption causes the *Primary Host Application* MQTT Session to server #2 to be terminated. This will cause the MQTT Server, on behalf of the now terminated the *Primary Host*

Application MQTT Client to publish the DEATH certificate to anyone that is currently subscribed to it. Upon receipt of the *Primary Host Application* DEATH Certificate this EoN node will move to the next MQTT Server in its table.

5. The EoN node moved to the next available MQTT Server and since the current STATE on this server is ‘ONLINE’, it can stay connected.
6. In the meantime, the network disruption between *Primary Host Application* and MQTT Server #2 has been corrected. The *Primary Host Application* has a new MQTT Session established to server #2 with an update Birth Certificate of ‘ONLINE’. Now MQTT Server #2 is ready to accept new EoN node session requests.

Chapter 9. Acknowledgements

Chapter 10. Conformance

10.1. Conformance Profiles

There are four Sparkplug target applications. A Sparkplug infrastructure typically consists of one or more of the following application types

- Sparkplug Edge Node
- Sparkplug Host Application
- Sparkplug Compliant MQTT Server
- Sparkplug Aware MQTT Server

Each application type has specific conformance requirements that must be met. Typically a Sparkplug application would only implement one of these profiles. For example an MQTT client wouldn't typically be both an Edge Node and a Host Application.

10.1.1. Sparkplug Edge Node

A Sparkplug Edge Node is typically an *Edge Gateway*. It sends and receives data to an MQTT Server using the spBv1.0/# namespace. Edge Nodes typically interact with physical devices to gather data and also write to device outputs.

10.1.2. Sparkplug Host Application

A Sparkplug Host Application is typically at a *central location* and primarily receives data from multiple Sparkplug Edge Nodes. It also may send command messages to Sparkplug Edge Nodes to write to outputs of Sparkplug Edge Nodes and/or Devices. Sparkplug Host Applications may also sends rebirth requests to Edge Nodes when required. **Sparkplug Host Applications MUST publish *STATE* messages that represent its Birth and Death Certificates.**

10.1.3. Sparkplug Compliant MQTT Server

Sparkplug infrastructures have a specific subset of requirements on MQTT Servers. Any fully MQTT v3.1.1 or v5.0 MQTT Server will meet the requirements of Sparkplug infrastructures. However, not all of the features of the MQTT Specification are required. The following are the required portions of the MQTT Specification that a Sparkplug Compliant MQTT Server must meet.

- **A Sparkplug conformant MQTT Server MUST support publish and subscribe on QoS 0**
- **A Sparkplug conformant MQTT Server MUST support publish and subscribe on QoS 1**
- **A Sparkplug conformant MQTT Server MUST support all aspects of Will Messages including use of the *retain flag* and QoS 1**
- **A Sparkplug conformant MQTT Server MUST support all aspects of the *retain flag***

10.1.4. Sparkplug Aware MQTT Server

A *Sparkplug Aware* MQTT Server includes all of the aspects of a Sparkplug Compliant MQTT Server. In addition, it also must have the ability to store NBIRTH and DBIRTH messages of Sparkplug Edge Nodes that pass through it. Any stored NBIRTH or DBIRTH message must always be the most recent NBIRTH or DBIRTH that was published to the MQTT Server. In addition, it must make the stored NBIRTH and DBIRTH messages available to MQTT clients via a retained MQTT message on the appropriate $sparkplug topic.

It is important to note these stored messages are the original NBIRTH and DBIRTH messages published by each Edge Node. As a result, the metric values can not be expected to be the current values. In a typical Sparkplug environment the Edge Node likely would have published NDATA and/or DDATA messages after the NBIRTH and DBIRTH messages denoting metric values that had changed at the Edge Node and its associated Sparkplug Devices. Consumers of the stored NBIRTH and DBIRTH messages should take this into consideration when using the information in the stored NBIRTH and DBIRTH payloads.

- **A Sparkplug Aware MQTT Server MUST support all aspects of a Sparkplug Compliant MQTT Server**
- **A Sparkplug Aware MQTT Server MUST store NBIRTH and DBIRTH messages as they pass through the MQTT Server**
- **A Sparkplug Aware MQTT Server MUST make NBIRTH messages available on a topic of the form: $sparkplug/certificates/namespace/group_id/NBIRTH/edge_node_id**
    - Example: Given a group_id=G1 and edge_node_id=E1, the topic the Sparkplug Aware MQTT Server must make the NBIRTH message available on is: $sparkplug/certificates/spBv1.0/G1/NBIRTH/E1
- **A Sparkplug Aware MQTT Server MUST make NBIRTH messages available on the topic: $sparkplug/certificates/namespace/group_id/NBIRTH/edge_node_id with the MQTT retain flag set to true**
- **A Sparkplug Aware MQTT Server MUST make DBIRTH messages available on a topic of the form: $sparkplug/certificates/namespace/group_id/DBIRTH/edge_node_id/device_id**
    - Example: Given a group_id=G1, edge_node_id=E1 and device_id=D1, the topic the Sparkplug Aware MQTT Server must make the DBIRTH message available on is: $sparkplug/certificates/spBv1.0/G1/DBIRTH/E1/D1
- **A Sparkplug Aware MQTT Server MUST make DBIRTH messages available on the topic: $sparkplug/certificates/namespace/group_id/DBIRTH/edge_node_id/device_id with the MQTT retain flag set to true**
- **A Sparkplug Aware MQTT Server MAY replace the timestmap of NDEATH messages. If it does, it MUST set the timestamp to the UTC time at which it attempts to deliver the NDEATH to subscribed clients**

Chapter 11. Appendix A: Open Source Software (non-normative)

11.1. OASIS MQTT Specifications

The Sparkplug Specification specifies that MQTT Server/Clients in the infrastructure adhere to the MQTT v3.1.1 and MQTT v5.0 Specifications. The Sparkplug Specification documentation refers to the following two links for the MQTT v3.1.1 and v5.0 Specifications.

- MQTT v3.1.1: http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/mqtt-v3.1.1.html
- MQTT v5.0: https://docs.oasis-open.org/mqtt/mqtt/v5.0/mqtt-v5.0.html

Also referred is an addendum document to the MQTT v3.1.1 Specification document that discusses best practices for implementing security on MQTT TCP/IP networks:

- http://docs.oasis-open.org/mqtt/mqtt-nist-cybersecurity/v1.0/mqtt-nist-cybersecurity-v1.0.doc

11.2. Eclipse Foundation IoT Resources

The Eclipse Foundation is an excellent resource for open source software supporting industry standards. There is a Sparkplug Working Group responsible for maintaining and developing the Sparkplug Specification.

- https://sparkplug.eclipse.org/

In addition to the Sparkplug Working Group, the Eclipse Foundation has an Internet of Things (IoT) working group providing a wealth of information and projects around the Internet of Things.

- https://iot.eclipse.org/

11.3. Eclipse Paho

Eclipse Paho™ is an Eclipse Foundation project that offers excellent resources for mature, compliant MQTT Client and MQTT Server implementations and well as additional resources for all things MQTT.

- http://www.eclipse.org/paho/

11.4. Google Protocol Buffers

Protocol buffers are Google's language-neutral, platform-neutral, extensible mechanism for serializing structured data. Google Protocol Buffers are used to encode the Sparkplug payload in both payload formats A and B of the Sparkplug Specification.

https://developers.google.com/protocol-buffers/

11.5. Eclipse Kura Google Protocol Buffer Schema

Eclipse Kura is another Eclipse Foundation project under the IoT resources. Kura provides open source resources for the Google Protocol Buffer representation of MQTT payloads as defined in the original Sparkplug A payload definition. While no longer used in Sparkplug it was critical to the evolution of Sparkplug.

- https://github.com/eclipse/kura/blob/develop/kura/org.eclipse.kura.core.cloud/src/main/protobuf/kurapayload.proto

11.6. Raspberry Pi Hardware

For the sake of keeping the Sparkplug Specification as real world as possible, a reference implementation of a Sparkplug Edge Node and associated Device is provided for the examples and screen shots in this document. All of this was implemented on Raspberry Pi hardware representing the Edge Node with a Pibrella I/O board representing the Device.

Chapter 12. Appendix B: List of Normative Statements (non-normative)

12.1. Host Applications

- Sparkplug Host Applications MUST publish STATE messages denoting their online and offline status.

12.2. Sparkplug Identifiers

- The Group ID MUST be UTF-8 string and used as part of the Sparkplug topics as defined in the Topics Section.
- Because the Group ID is used in MQTT topic strings the Group ID MUST only contain characters allowed for MQTT topics per the MQTT Specification.
- The Edge Node ID MUST be UTF-8 string and used as part of the Sparkplug topics as defined in the Topics Section.
- Because the Edge Node ID is used in MQTT topic strings the Group ID MUST only contain characters allowed for MQTT topics per the MQTT Specification.
- The Device ID MUST be UTF-8 string and used as part of the Sparkplug topics as defined in the Topics Section.
- Because the Device ID is used in MQTT topic strings the Group ID MUST only contain characters allowed for MQTT topics per the MQTT Specification.
- The Edge Node Descriptor MUST be unique within the context of all of other Edge Nodes within the Sparkplug infrastructure.

12.3. Report by Exception

- Because of the stateful nature of Sparkplug sessions, data SHOULD NOT be published from Edge Nodes on a periodic basis and instead SHOULD be published using a RBE based approach.

12.4. Birth and Death Certificates

- Birth Certificates MUST be the first MQTT messages published by any Edge Node or any Host Application.

12.5. Persistent vs Non-Persistent Connections for Edge Nodes

- Edge Node MQTT CONNECT packets MUST set the *Clean Session* flag to false.

12.6. Sparkplug Host Application

- A Sparkplug Host Application MUST utilize the STATE messages to denote whether it is online or offline at any given point in time.

12.7. Topic Namespace Elements

- All MQTT clients using the Sparkplug specification MUST use the following topic namespace structure

12.8. namespace Element

- For the Sparkplug B version of the payload definition, the UTF-8 string constant for the namespace element MUST be

12.9. group_id Element

- The format of the Group ID MUST be a valid UTF-8 string with the exception of the reserved characters of + (plus), / (forward slash), and # (number sign).

12.10. edge_node_id Element

- The group_id combined with the edge_node_id element MUST be unique from any other group_id/edge_node_id assigned in the MQTT infrastructure.
- The format of the edge_node_id MUST be a valid UTF-8 string with the exception of the reserved characters of + (plus), / (forward slash), and # (number sign).

12.11. device_id Element

- The format of the device_id MUST be a valid UTF-8 string except for the reserved characters of + (plus), / (forward slash), and # (number sign).
- The device_id MUST be unique from other devices being reported on by the same Edge Node.
- The device_id MAY be duplicated from Edge Node to other Edge Nodes.
- The device_id MUST be included with message_type elements DBIRTH, DDEATH, DDATA, and DCMD based topics
- The device_id MUST NOT be included with message_type elements NBIRTH, NDEATH, NDATA, NCMD, and STATE based topics

12.12. Payload (NBIRTH)

- NBIRTH messages MUST be published with MQTT QoS equal to 0 and retain equal to false.
- The NBIRTH MUST include a sequence number in the payload and it MUST have a value of 0.
- The NBIRTH MUST include a timestamp denoting the Date/Time the message was sent from the

Edge Node.

- The NBIRTH MUST include every metric the Edge Node will ever report on.
- At a minimum each metric MUST include the following
- If Template instances will be published by this Edge Node or any devices, all Template definitions MUST be published in the NBIRTH.
- A bdseq number as a metric MUST be included in the payload.
- This MUST match the bdseq number provided in the MQTT CONNECT packet's Will Message payload.
- The bdseq number MUST start at zero and increment by one on every new MQTT CONNECT packet.
- The NBIRTH message MUST include the following metric

12.13. Payload (NDATA)

- NDATA messages MUST be published with MQTT QoS equal to 0 and retain equal to false.
- The NDATA MUST include a sequence number in the payload and it MUST have a value of one greater than the previous MQTT message from the Edge Node contained unless the previous MQTT message contained a value of 255. In this case the sequence number MUST be 0.
- The NDATA MUST include a timestamp denoting the Date/Time the message was sent from the Edge Node.
- The NDATA MUST include the Edge Node's metrics that have changed since the last NBIRTH or NDATA message.

12.14. Payload (NDEATH)

- The NDEATH message contains a very simple payload that MUST only include a single metric, the bdseq number, so that the NDEATH event can be associated with the NBIRTH.
- The NDEATH message MUST NOT include a sequence number.

12.15. Payload (NCMD)

- NCMD messages MUST be published with MQTT QoS equal to 0 and retain equal to false.
- The NCMD MUST include a timestamp denoting the Date/Time the message was sent from the Host Application's MQTT client.
- The NCMD MUST include the metrics that need to be written to on the Edge Node.

12.16. Payload (DBIRTH)

- DBIRTH messages MUST be published with MQTT QoS equal to 0 and retain equal to false.
- The DBIRTH MUST include a sequence number in the payload and it MUST have a value of one greater than the previous MQTT message from the Edge Node contained unless the previous

MQTT message contained a value of 255. In this case the sequence number MUST be 0.

- The DBIRTH MUST include a timestamp denoting the Date/Time the message was sent from the Edge Node.
- The DBIRTH MUST include every metric the Edge Node will ever report on.
- At a minimum each metric MUST include the following

12.17. Payload (DDATA)

- DDATA messages MUST be published with MQTT QoS equal to 0 and retain equal to false.
- The DDATA MUST include a sequence number in the payload and it MUST have a value of one greater than the previous MQTT message from the Edge Node contained unless the previous MQTT message contained a value of 255. In this case the sequence number MUST be 0.
- The DDATA MUST include a timestamp denoting the Date/Time the message was sent from the Edge Node.
- The DDATA MUST include the Device's metrics that have changed since the last DBIRTH or DDATA message.

12.18. Payload (DDEATH)

- DDEATH messages MUST be published with MQTT QoS equal to 0 and retain equal to false.
- The DDEATH MUST include a sequence number in the payload and it MUST have a value of one greater than the previous MQTT message from the Edge Node contained unless the previous MQTT message contained a value of 255. In this case the sequence number MUST be 0.

12.19. Payload (DCMD)

- DCMD messages MUST be published with MQTT QoS equal to 0 and retain equal to false.
- The DCMD MUST include a timestamp denoting the Date/Time the message was sent from the Host Application's MQTT client.
- The DCMD MUST include the metrics that need to be written to on the Device.

12.20. Birth Certificate Message (STATE)

- The first MQTT message a Host Application MUST publish is a Birth Certificate.

12.21. Birth Certificate Topic (STATE)

- STATE/sparkplug_host_application_id
- The Birth Certificate Payload MUST be the UTF-8 string "ONLINE"
- The MQTT Quality of Service (QoS) MUST be set to 1
- The MQTT retain flag for the Birth Certificate MUST be set to TRUE

12.22. Birth Certificate Payload (STATE)

- The STATE message from the Sparkplug Host Application Birth Certificate message MUST include a payload that is a UTF-8 string that is the following

12.23. Death Certificate Topic (STATE)

- STATE/sparkplug_host_application_id
- The Sparkplug Host Application MUST provide a Will message in the MQTT CONNECT packet
- The MQTT Will Payload MUST be the UTF-8 string “OFFLINE”
- The MQTT Will QoS MUST be set to 1
- The MQTT Will retain flag MUST be set to TRUE

12.24. Death Certificate Payload (STATE)

- The STATE messages from the Sparkplug Host Application Death Certificate message MUST include a payload that is a UTF-8 string that is the following

12.25. Host Application Session Establishment

- The CONNECT Control Packet for all Sparkplug Host Applications MUST set the MQTT *Clean Session* flag to true.
- The subscription on the Sparkplug Topic Namespace and the STATE topic MUST be done immediately after successfully establishing the MQTT session and before publishing its own STATE message.
- Once an MQTT Session has been established, the Sparkplug Host Application subscriptions on the Sparkplug Topic Namespace have been established, and the STATE topic subscription has been been established, the Sparkplug Host Application MUST publish a new STATE message with a Payload of a UTF-8 string of *ONLINE*.
- All metric data associated with any Sparkplug Edge Node that was connected to that MQTT Server and known by the Host Application MUST be updated to a STALE data quality if the Host Application loses connection to the MQTT Server.

12.26. Edge Node Session Establishment

- Any Edge Node in the MQTT infrastructure MUST establish an MQTT Session prior to publishing NBIRTH and DBIRTH messages.
- Any Edge Node in the MQTT infrastructure MUST verify the Primary Host Application is ONLINE via the STATE topic if a Primary Host Application is configured for the Edge Node before publishing NBIRTH and DBIRTH messages.

12.27. Edge Node Session Termination

- An Edge Node MUST publish an NDEATH before terminating the connection.
- Immediately following the NDEATH publish, a DISCONNECT packet MUST be sent to the MQTT Server.

12.28. Device Session Termination

- If a Sparkplug Edge Node loses connection with an attached Sparkplug Device, it MUST publish a DDEATH message on behalf of the device.

12.29. Sparkplug Host Application Message Ordering

- Sparkplug Host Applications SHOULD provide a configurable *Reorder Timeout* parameter
- If a message arrives with a out of order sequence number, the Host Application SHOULD start a timer denoting the start of the Reorder Timeout window
- If the Reorder Timeout elapses and the missing message(s) have not been received, the Sparkplug Host Application SHOULD send an NCMD to the Edge Node with a *Node Control/Rebirth* request
- If the missing messages that triggered the start of the Reorder Timeout timer arrive before the reordering timer elapses, the timer can be terminated and normal operation in the Host Application can continue

12.30. Primary Host Application STATE in Multiple MQTT Server Topologies

- Regardless of the number of MQTT Servers in a Sparkplug Infrastructure, every time a Primary Host Application establishes a new MQTT Session with an MQTT Server, the STATE Birth Certificate defined in the STATE description section MUST be the first message that is published after a successful MQTT Session is established with each MQTT Server.
- The Edge Nodes MUST not connected to more than one server at any point in time.
- If the Primary Host Application is OFFLINE as denoted via the STATE MQTT Message, the Edge Node MUST terminate its session with this MQTT Server and move to the next available MQTT Server that is available.
- The Edge Node MUST also wait to publish its BIRTH sequence until an "ONLINE" STATE message is received by the Edge Node.

12.31. Sparkplug Host Application Session Establishment

- The host_id MUST be unique to all other Sparkplug Host IDs in the infrastructure.
- When a Sparkplug Host Application sends its MQTT CONNECT packet, it MUST include a Will

Message.

- The MQTT Will Message's topic MUST be of the form *STATE/host_id* where host_id is the unique identifier of the Sparkplug Host Application
- The MQTT Will Message's payload MUST be the ASCII String of *OFFLINE*.
- The MQTT Will Message's MQTT QoS MUST be 1 (at least once).
- The MQTT Will Message's retained flag MUST be set to true.
- The MQTT Client associated with the Sparkplug Host Application MUST send a birth message immediately after successfully connecting to the MQTT Server.
- The Host Application's Birth topic MUST be of the form *STATE/host_id* where host_id is the unique identifier of the Sparkplug Host Application
- The Host Application's Birth payload MUST be the ASCII String of *ONLINE*.
- The Host Application's Birth MQTT QoS MUST be 1 (at least once).
- The Host Application's Birth retained flag MUST be set to true.

12.32. Sparkplug Host Application Session Termination

- The Sparkplug Host Application's Death topic MUST be of the form *STATE/host_id* where host_id is the unique identifier of the Sparkplug Host Application.
- The Sparkplug Host Application's Death payload MUST be the ASCII String of *OFFLINE*.
- The Sparkplug Host Application's Death MQTT QoS MUST be 1 (at least once).
- The Sparkplug Host Application's Death retained flag MUST be set to true.
- In the case of intentionally disconnecting, an MQTT DISCONNECT packet MUST be sent immediately after the Death message is sent.

12.33. Data Publish

- NBIRTH messages MUST include all metrics for the specified Edge Node that will ever be published for that Edge Node within the established Sparkplug session.
- NBIRTH messages MUST include current values for all metrics.
- NDATA messages MUST only be published when Edge Node level metrics change.
- For all metrics where is_historical=false, NBIRTH and NDATA messages MUST keep metric values in chronological order in the list of metrics in the payload.
- DBIRTH messages MUST include all metrics for the specified Device that will ever be published for that Device within the established Sparkplug session.
- DBIRTH messages MUST include current values for all metrics.
- DDATA messages MUST only be published when Device level metrics change.
- For all metrics where is_historical=false, DBIRTH and DDATA messages MUST keep metric values in chronological order in the list of metrics in the payload.

12.34. Commands

- An NBIRTH message MUST include a metric with a name of *Node Control/Rebirth*.
- The *Node Control/Rebirth* metric in the NBIRTH message MUST have a datatype of *Boolean*.
- The *Node Control/Rebirth* metric value in the NBIRTH message MUST have a value of false.
- A Rebirth Request MUST use the NCMD Sparkplug verb.
- A Rebirth Request MUST include a metric with a name of *Node Control/Rebirth*.
- A Rebirth Request MUST include a metric value of true.
- When an Edge Node receives a Rebirth Request, it MUST immediately stop sending DATA messages.
- After an Edge Node stops sending DATA messages, it MUST send a complete BIRTH sequence including the NBIRTH and DBIRTH(s) if applicable.
- The NBIRTH MUST include the same bdSeq metric with the same value it had included in the Will Message of the previous MQTT CONNECT packet.
- An Edge Node level command MUST use the NCMD Sparkplug verb.
- An NCMD message MUST include a metric name that was included in the associated NBIRTH message for the Edge Node.
- An NCMD message MUST include a compatible metric value for the metric name that it is writing to.
- A Device level command MUST use the DCMD Sparkplug verb.
- A DCMD message MUST include a metric name that was included in the associated DBIRTH message for the Device.
- A DCMD message MUST include a compatible metric value for the metric name that it is writing to.

12.35. Payload

- This timestamp MUST be in UTC.
- A sequence number MUST be included in the payload of every Sparkplug MQTT message except NDEATH messages.
- A NBIRTH message MUST always contain a sequence number of zero.
- All subsequent messages MUST contain a sequence number that is continually increasing by one in each message until a value of 255 is reached. At that point, the sequence number of the following message MUST be zero.

12.36. Metric

- The name MUST be included with every metric unless aliases are being used.
- If supplied in an NBIRTH or DBIRTH it MUST be a unique number across this Edge Node's entire set of metrics.

- NBIRTH and DBIRTH messages MUST include both a metric name and alias.
- NDATA, DDATA, NCMD, and DCMD messages MUST only include an alias and the metric name MUST be excluded.
- The timestamp MUST be included with every metric in all NBIRTH, DBIRTH, NDATA, and DDATA messages.
- The timestamp MAY be included with metrics in NCMD and DCMD messages.
- This timestamp MUST be in UTC.
- This MUST be an unsigned 32-bit integer representing the datatype.
- This value MUST be one of the enumerated values as shown in the valid Sparkplug Data Types.
- This MUST be included with each metric definitions in NBIRTH and DBIRTH messages.
- This SHOULD NOT be included with metric definitions in NDATA, NCMD, DDATA, and DCMD messages.

12.37. PropertySet

- The array of keys in a PropertySet MUST contain the same number of values included in the array of PropertyValue objects.
- The array of values in a PropertySet MUST contain the same number of items that are in the keys array.

12.38. PropertyValue

- This MUST be an unsigned 32-bit integer representing the datatype.
- This value MUST be one of the enumerated values as shown in the Sparkplug Basic Data Types or the Sparkplug Property Value Data Types.
- This MUST be included in Property Value Definitions in NBIRTH and DBIRTH messages.

12.39. Quality Codes

- The *type* of the Property Value MUST be a value of 3 which represents a Signed 32-bit Integer.
- The *value* of the Property Value MUST be an int_value and be one of the valid quality codes of 0, 192, or 500.

12.40. DataSet

- This MUST be an unsigned 64-bit integer representing the number of columns in this DataSet.
- The size of the array MUST have the same number of elements that the types array contains.
- This MUST be an array of unsigned 32 bit integers representing the datatypes of the columns.
- The array of types MUST have the same number of elements that the columns array contains.
- The values in the types array MUST be a unsigned 32-bit integer representing the datatype.

- This values in the types array MUST be one of the enumerated values as shown in the Sparkplug Basic Data Types.
- This MUST be included in DataSet Definitions in NBIRTH and DBIRTH messages.

12.41. DataSet.DataSetValue

- The value supplied MUST be one of the following types: uint32, uint64, float, double, bool, or string.

12.42. Template

- A Template Definition MUST have is_definitiion set to true.
- A Template Definition MUST omit the template_ref field.
- A Template Instance MUST have is_definitiion set to false.
- A Template Instance MUST have template_ref set to the type of template definition it is.
- If included, the version MUST be a UTF-8 string representing the version of the Template.
- This MUST be omitted if this is a Template Definition.
- This MUST be a UTF-8 string representing a reference to a Template Definition name if this is a Template Instance.
- This MUST be included in every Template Definition and Template Instance.
- This MUST be set to true if this is a Template Definition.
- This MUST be set to false if this is a Template Instance.

12.43. Template.Parameter

- This MUST be included in every Template Parameter definition.
- This MUST be a UTF-8 string representing the name of the Template parameter.
- This MUST be an unsigned 32-bit integer representing the datatype.
- This value MUST be one of the enumerated values as shown in the Sparkplug Basic Data Types.
- This MUST be included in Template Parameter Definitions in NBIRTH and DBIRTH messages.
- The value supplied MUST be one of the following types: uint32, uint64, float, double, bool, or string.

12.44. NBIRTH

- NBIRTH messages MUST include a payload timestamp that denotes the time at which the message was published.
- Every Edge Node Descriptor in any Sparkplug infrastructure MUST be unique in the system.
- Every NBIRTH message MUST include a sequence number and it MUST have a value of 0.
- Every NBIRTH message MUST include a bdSeq number metric.

- Every NBIRTH message in a new Sparkplug MQTT session SHOULD include a bdSeq number value that is one greater than the previous NBIRTH's bdSeq number. This value MUST never exceed 255. If in the previous NBIRTH a value of 255 was sent, the next NBIRTH MUST have a value of 0.
- Every NBIRTH MUST include a metric with the name *Node Control/Rebirth* and have a boolean value of false.
- NBIRTH messages MUST be published with the MQTT QoS set to 0.
- NBIRTH messages MUST be published with the MQTT retain flag set to false.

12.45. DBIRTH

- DBIRTH messages MUST include a payload timestamp that denotes the time at which the message was published.
- Every DBIRTH message MUST include a sequence number.
- Every DBIRTH message MUST include a sequence number value that is one greater than the previous sequence number sent by the Edge Node. This value MUST never exceed 255. If in the previous sequence number sent by the Edge Node was 255, the next sequence number sent MUST have a value of 0.
- All DBIRTH messages sent by and Edge Node MUST be sent immediately after the NBIRTH and before any NDATA or DDATA messages are published by the Edge Node.
- DBIRTH messages MUST be published with the MQTT QoS set to 0.
- DBIRTH messages MUST be published with the MQTT retain flag set to false.

12.46. NDATA

- NDATA messages MUST include a payload timestmp that denotes the time at which the message was published.
- Every NDATA message MUST include a sequence number.
- Every NDATA message MUST include a sequence number value that is one greater than the previous sequence number sent by the Edge Node. This value MUST never exceed 255. If in the previous sequence number sent by the Edge Node was 255, the next sequence number sent MUST have a value of 0.
- All NDATA messages sent by and Edge Node MUST NOT be sent until all the NBIRTH and all DBIRTH messages have been published by the Edge Node.
- NDATA messages MUST be published with the MQTT QoS set to 0.
- NDATA messages MUST be published with the MQTT retain flag set to false.

12.47. DDATA

- DDATA messages MUST include a payload timestamp that denotes the time at which the message was published.

- Every DDATA message MUST include a sequence number.
- Every DDATA message MUST include a sequence number value that is one greater than the previous sequence number sent by the Edge Node. This value MUST never exceed 255. If in the previous sequence number sent by the Edge Node was 255, the next sequence number sent MUST have a value of 0.
- All DDATA messages sent by and Edge Node MUST NOT be sent until all the NBIRTH and all DBIRTH messages have been published by the Edge Node.
- DDATA messages MUST be published with the MQTT QoS set to 0.
- DDATA messages MUST be published with the MQTT retain flag set to false.

12.48. NCMD

- NCMD messages MUST include a payload timestamp that denotes the time at which the message was published.
- Every NCMD message MUST NOT include a sequence number.
- NCMD messages MUST be published with the MQTT QoS set to 0.
- NCMD messages MUST be published with the MQTT retain flag set to false.

12.49. DCMD

- DCMD messages MUST include a payload timestamp that denotes the time at which the message was published.
- Every DCMD message MUST NOT include a sequence number.
- DCMD messages MUST be published with the MQTT QoS set to 0.
- DCMD messages MUST be published with the MQTT retain flag set to false.

12.50. NDEATH

- Every NDEATH message MUST NOT include a sequence number.
- An NDEATH message MUST be registered as a Will Message in the MQTT CONNECT packet.
- The NDEATH message MUST set the MQTT Will QoS to 1 in the MQTT CONNECT packet.
- The NDEATH message MUST set the MQTT Will Retained flag to false in the MQTT CONNECT packet.
- The NDEATH message MUST include the same bdSeq number value that will be used in the associated NBIRTH message.
- An NDEATH message SHOULD be published by the Edge Node before it intentionally disconnects from the MQTT Server.

12.51. DDEATH

- DDEATH messages MUST include a payload timestamp that denotes the time at which the message was published.
- Every NDATA message MUST include a sequence number.
- Every NDATA message MUST include a sequence number value that is one greater than the previous sequence number sent by the Edge Node. This value MUST never exceed 255. If in the previous sequence number sent by the Edge Node was 255, the next sequence number sent MUST have a value of 0.
- A sequence number MUST be included with the DDEATH messages so the Host Application can ensure order of messages and maintain the state of the data.

12.52. STATE

- Sparkplug Host Applications MUST register a Will Message in the MQTT CONNECT packet on the topic *STATE/[sparkplug_host_id]*.
- The Sparkplug Host Application MUST set the the MQTT Will QoS to 1 in the MQTT CONNECT packet.
- The Sparkplug Host Application MUST set the Will Retained flag to true in the MQTT CONNECT packet.
- The Sparkplug Host Application MUST set the Will Payload to the UTF-8 string of *OFFLINE* in the MQTT CONNECT packet.
- After establishing an MQTT connection, the Sparkplug Host Application MUST subscribe on it's own *STATE/[sparkplug_host_id]* topic.
- After subscribing on it's own STATE/[sparkplug_host_id] topic, the Sparkplug Host Application MUST publish an MQTT message on the topic *STATE/[sparkplug_host_id]* with a payload of the UTF-8 string of *ONLINE*, a QoS of 1, and the retain flag set to true.

12.53. Sparkplug Host Application

- Sparkplug Host Applications MUST publish *STATE* messages that represent its Birth and Death Certificates.

12.54. Sparkplug Compliant MQTT Server

- A Sparkplug conformant MQTT Server MUST support publish and subscribe on QoS 0
- A Sparkplug conformant MQTT Server MUST support publish and subscribe on QoS 1
- A Sparkplug conformant MQTT Server MUST support all aspects of Will Messages including use of the *retain flag* and QoS 1
- A Sparkplug conformant MQTT Server MUST support all aspects of the *retain flag*

12.55. Sparkplug Aware MQTT Server

- A Sparkplug Aware MQTT Server MUST support all aspects of a Sparkplug Compliant MQTT Server
- A Sparkplug Aware MQTT Server MUST store NBIRTH and DBIRTH messages as they pass through the MQTT Server
- A Sparkplug Aware MQTT Server MUST make NBIRTH messages available on a topic of the form: $sparkplug/certificates/namespace/group_id/NBIRTH/edge_node_id
- A Sparkplug Aware MQTT Server MUST make NBIRTH messages available on the topic: $sparkplug/certificates/namespace/group_id/NBIRTH/edge_node_id with the MQTT retain flag set to true
- A Sparkplug Aware MQTT Server MUST make DBIRTH messages available on a topic of the form: $sparkplug/certificates/namespace/group_id/DBIRTH/edge_node_id/device_id
- A Sparkplug Aware MQTT Server MUST make DBIRTH messages available on the topic: $sparkplug/certificates/namespace/group_id/DBIRTH/edge_node_id/device_id with the MQTT retain flag set to true
- A Sparkplug Aware MQTT Server MAY replace the timestmap of NDEATH messages. If it does, it MUST set the timestamp to the UTC time at which it attempts to deliver the NDEATH to subscribed clients