INTERNATIONAL STANDARD

ISO/IEC 20237

First edition
2023-10

Information technology — Sparkplug® version 3.0

Reference number
ISO/IEC 20237:2023(E)

ISO copyright office
CP 401 • Ch. de Blandonnet 8
CH-1214 Vernier, Geneva
Phone: +41 22 749 01 11
Email: copyright@iso.org
Website: www.iso.org

Published in Switzerland

Foreword

ISO (the International Organization for Standardization) and IEC (the International Electrotechnical Commission) form the specialized system for worldwide standardization. National bodies that are members of ISO or IEC participate in the development of International Standards through technical committees established by the respective organization to deal with particular fields of technical activity. ISO and IEC technical committees collaborate in fields of mutual interest. Other international organizations, governmental and non-governmental, in liaison with ISO and IEC, also take part in the work.

The procedures used to develop this document and those intended for its further maintenance are described in the ISO/IEC Directives, Part 1. In particular, the different approval criteria needed for the different types of document should be noted (see www.iso.org/directives or www.iec.ch/members_experts/refdocs).

ISO and IEC draw attention to the possibility that the implementation of this document may involve the use of (a) patent(s). ISO and IEC take no position concerning the evidence, validity or applicability of any claimed patent rights in respect thereof. As of the date of publication of this document, ISO and IEC had not received notice of (a) patent(s) which may be required to implement this document. However, implementers are cautioned that this may not represent the latest information, which may be obtained from the patent database available at www.iso.org/patents and https://patents.iec.ch. ISO and IEC shall not be held responsible for identifying any or all such patent rights.

Any trade name used in this document is information given for the convenience of users and does not constitute an endorsement.

For an explanation of the voluntary nature of standards, the meaning of ISO specific terms and expressions related to conformity assessment, as well as information about ISO's adherence to the World Trade Organization (WTO) principles in the Technical Barriers to Trade (TBT), see www.iso.org/iso/foreword.html. In the IEC, see www.iec.ch/understanding-standards.

This document was prepared by OASIS (as Sparkplug 3.0.0, Sparkplug Specification) and drafted in accordance with its editorial rules. It was adopted, under the JTC 1 PAS procedure, by Joint Technical Committee ISO/IEC JTC 1, *Information technology*.

Any feedback or questions on this document should be directed to the user's national standards body. A complete listing of these bodies can be found at www.iso.org/members.html and www.iec.ch/national-committees.

Table of Contents

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
3.0.0	11/16/22	Eclipse Sparkplug Specification Project Team	Reorganized to be in AsciiDoc format and to include normative and non-normative statements

Sparkplug®, Sparkplug Compatible, and the Sparkplug Logo are trademarks of the Eclipse Foundation.

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 it 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 a way 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 outage prone network infrastructures. These can include cellular, satellite, and other radio based networks. 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.

It is important to note that reducing bandwidth usage and being resilient to network drops is advantageous on more reliable and high bandwidth networks as well. By reducing the bandwidth usage, Sparkplug is able to move more data through the network because of its efficiency. This in turn can reduce network costs.

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. Disclaimers

1.3. Organization of the Sparkplug Specification

This specification is split into the following chapters and appendices:

- Chapter 1 - Introduction
- Chapter 2 - Principles
- Chapter 3 - Sparkplug Architecture and Infrastructure Components
- Chapter 4 - Topics and Messages
- Chapter 5 - Operational Behavior
- Chapter 6 - Payloads
- Chapter 7 - Security
- Chapter 8 - High Availability
- Chapter 9 - Acknowledgements
- Chapter 10 - Conformance
- Appendix A - Open Source Software
- Appendix B - List of Normative Statements

1.4. Terminology

1.4.1. Infrastructure Components

This section details the infrastructure components implemented.

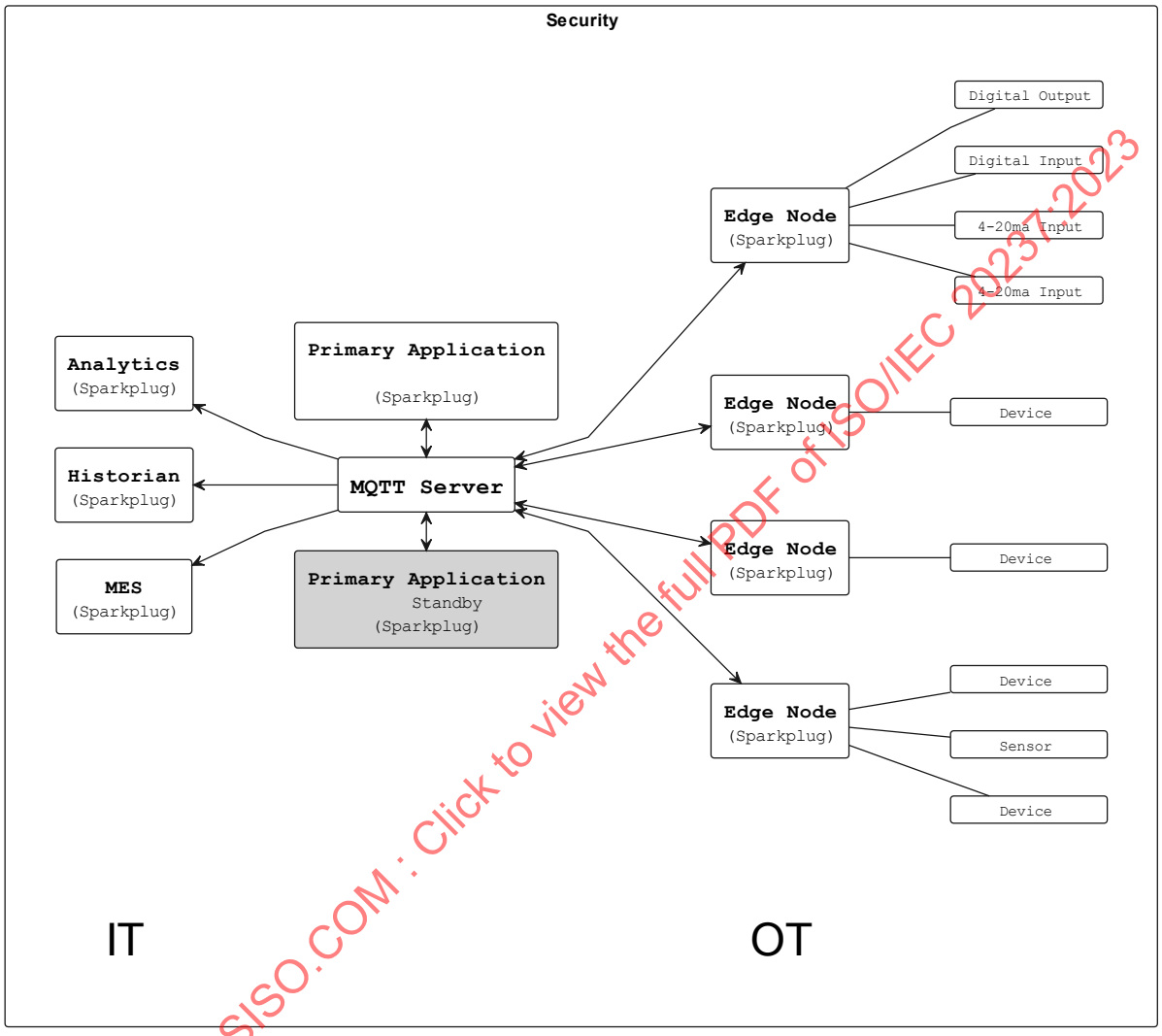

Figure 1 - MQTT SCADA Infrastructure

MQTT Server(s)

Program or device that acts as an intermediary between Clients which publish Application Messages and Clients which have made Subscriptions[MQTTV5-1.2]. 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

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

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

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

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 Edge Node 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

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.

[tck-id-intro-sparkplug-host-state] Sparkplug Host Applications MUST publish STATE messages denoting their online and offline status.

Primary Host Application

Most important consumer of Sparkplug Edge Node data. The Primary Host Application must be online to keep operations running.

A Primary Host Application 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. 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
  - **[tck-id-intro-group-id-string] The Group ID MUST be a UTF-8 string and used as part of the Sparkplug topics as defined in the Topics Section.**
  - **[tck-id-intro-group-id-chars] 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
  - **[tck-id-intro-edge-node-id-string] The Edge Node ID MUST be a UTF-8 string and used as part of the Sparkplug topics as defined in the Topics Section.**
  - **[tck-id-intro-edge-node-id-chars] Because the Edge Node ID is used in MQTT topic strings the Edge Node 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
  - **[tck-id-intro-device-id-string] The Device ID MUST be a UTF-8 string and used as part of the Sparkplug topics as defined in the Topics Section.**

  - **[tck-id-intro-device-id-chars] Because the Device ID is used in MQTT topic strings the Device 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.
  - **[tck-id-intro-edge-node-id-uniqueness] 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

Identifies 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.

Data Types

There are different uses of the term 'datatype' in the specification. Sparkplug encodes the payloads using Google Protocol Buffers. Google Protocol Buffers has its own scalar value types here: https://developers.google.com/protocol-buffers/docs/proto#scalar

The Google Protocol Buffer datatypes define what actually travels over the TCP/IP socket in the MQTT payload. For ease of programming, Google Protobuf includes a compiler tool that generates code in multiple different languages. These Protobuf datatypes are then represented by their proper native programming language datatypes. This is done on a per language basis after the Google Protobuf file is used to generate the code for each specific language.

In addition to Protobuf datatypes and native programming language datatypes there are also 'Sparkplug datatypes'. These are defined in the Sparkplug Protobuf Schema. These datatypes are those that are used for Sparkplug Metrics. Every Metric must include a Sparkplug Datatype in the NBIRTH or DBIRTH message depending on whether the Metric is a 'Node level' or 'Device level' metric. Each of the Sparkplug Datatypes is then represented by a Google Protobuf datatype.

1.5. Normative References

- [BCP14] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997. Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, May 2017.

- [MQTTV50] MQTT Version 5.0. Edited by Andrew Banks, Ed Briggs, Ken Borgendale, and Rahul Gupta. 07 March 2019. OASIS Standard. https://docs.oasis-open.org/mqtt/mqtt/v5.0/os/mqtt-v5.0-os.html. Latest version: https://docs.oasis-open.org/mqtt/mqtt/v5.0/mqtt-v5.0.html.
- [MQTTV311] MQTT Version 3.1.1 Plus Errata 01. Edited by Andrew Banks and Rahul Gupta. 10 December 2015. OASIS Standard Incorporating Approved Errata 01. http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/errata01/os/mqtt-v3.1.1-errata01-os-complete.html. Latest version: http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/mqtt-v3.1.1.html.
- [ISO/IEC 20922:2016] Information technology — Message Queuing Telemetry Transport (MQTT) v3.1.1

1.6. Consolidated List of Normative Statements

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

1.7. 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.

1.7.1. 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.

1.7.2. 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.

1.7.3. 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.

1.8. Normative Keywords

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 BCP 14 [BCP14].

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

1.9. Leveraging Standards and Open Source

In addition to leveraging MQTT v3.1.1 [MQTTV311] and MQTT v5.0 [MQTTV50] 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

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 devices from any direct connection to any one consumer application.

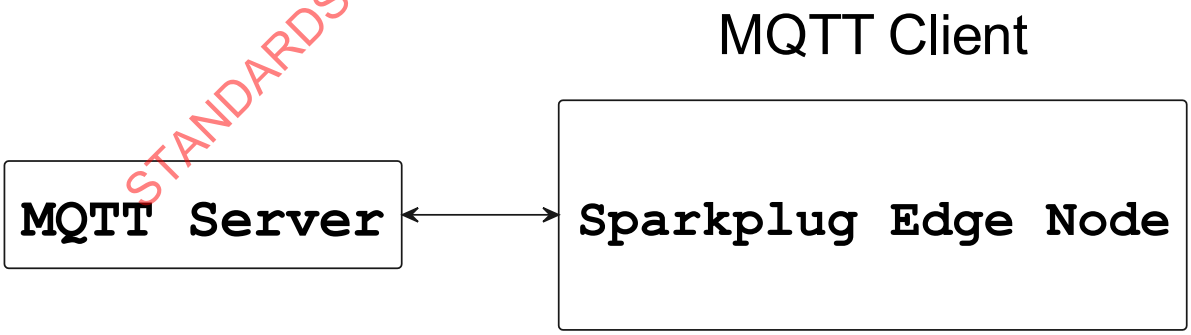

Figure 2 - Simple MQTT Infrastructure

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 [MQTTV5-4.1], 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.

[tck-id-principles-rbe-recommended] 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. Note the uses of the term 'session' here should not be confused with MQTT's 'clean session' concept/flags. That is covered later in this specification. 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 [MQTTV5-3.1.2.5] 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 spBv1.0/STATE/sparkplug_host_id topic is used. More information on Death certificates can be found in Edge Node Death Certificates and Host Application Death Certificates

- **[tck-id-principles-birth-certificates-order] Birth Certificates MUST be the first MQTT messages published by any Edge Node or any Host Application.**

Birth Certificates denote to any subscribing MQTT clients that the Edge Node or Host Application is now online. For Edge Nodes, the Birth Certificate uses the NBIRTH Sparkplug verb in the topic. For Host Applications, the spBv1.0/STATE/sparkplug_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 [MQTTV5-3.14] 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 not be sent to Sparkplug Host Applications. 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 Edge 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, the following rules must be followed:

- **[tck-id-principles-persistence-clean-session-311] If the MQTT client is using MQTT v3.1.1, the Edge Node's MQTT CONNECT packet MUST set the 'Clean Session' flag to true.**
- **[tck-id-principles-persistence-clean-session-50] If the MQTT client is using MQTT v5.0, the Edge Node's MQTT CONNECT packet MUST set the 'Clean Start' flag to true and the 'Session Expiry Interval' to 0.**

3. Sparkplug Architecture and Infrastructure Components

This section details the infrastructure components implemented.

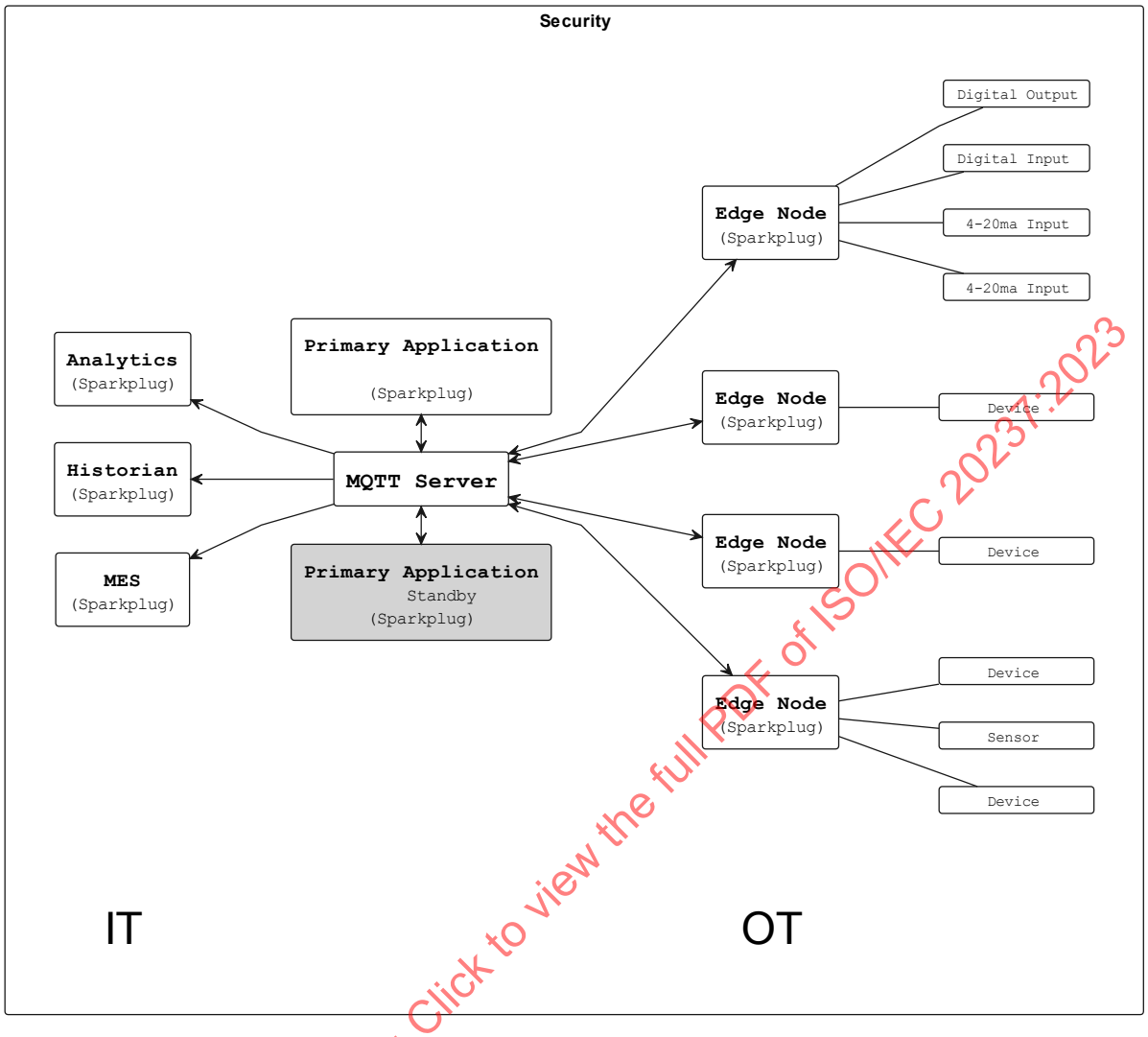

Figure 3 - MQTT SCADA Infrastructure

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 [MQTTV3.1.1] or v5.0 [MQTTV5] 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.

[tck-id-components-ph-state] A Sparkplug Host Application MUST utilize the STATE messages to denote whether it is online or offline at any given point in time.

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

[tck-id-topic-structure] 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 the second is for the Sparkplug payload definition B.

[tck-id-topic-structure-namespace-a] 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.

[tck-id-topic-structure-namespace-valid-group-id] 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, the Group ID 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.

[tck-id-topic-structure-namespace-unique-edge-node-descriptor] 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.

[tck-id-topic-structure-namespace-valid-edge-node-id] 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.

[tck-id-topic-structure-namespace-valid-device-id] 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).

[tck-id-topic-structure-namespace-unique-device-id] The device_id MUST be unique from other devices being reported on by the same Edge Node.

[tck-id-topic-structure-namespace-duplicate-device-id-across-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.

[tck-id-topic-structure-namespace-device-id-associated-message-types] The device_id MUST be included with message_type elements DBIRTH, DDEATH, DDATA, and DCMD based topics.

[tck-id-topic-structure-namespace-device-id-non-associated-message-types] 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 message types can be used.

4.2.1. Edge Node

Topic (NBIRTH)

- **[tck-id-topics-nbirth-topic] The Birth Certificate topic for a Sparkplug Edge Node MUST be of the form 'namespace/group_id/NBIRTH/edge_node_id' where the namespace is replaced with the specific namespace for this version of Sparkplug and the group_id and edge_node_id are replaced with the Group and Edge Node ID for this specific Edge Node.**

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 and 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 new DBIRTH messages associated with the new Sparkplug session..

The NBIRTH message requires the following payload components.

- **[tck-id-topics-nbirth-mqtt] NBIRTH messages MUST be published with MQTT QoS equal to 0 and retain equal to false.**
- **[tck-id-topics-nbirth-seq-num] The NBIRTH MUST include a sequence number in the payload and it MUST have a value of 0.**
- **[tck-id-topics-nbirth-timestamp] The NBIRTH MUST include a timestamp denoting the date and time the message was sent from the Edge Node.**
- **[tck-id-topics-nbirth-metric-reqs] The NBIRTH MUST include every metric the Edge Node will ever report on.**
- **[tck-id-topics-nbirth-metrics] At a minimum each metric MUST include the metric name, datatype, and current value.**
- **[tck-id-topics-nbirth-templates] If Template instances will be published by this Edge Node or any devices, all Template definitions MUST be published in the NBIRTH.**
- **[tck-id-topics-nbirth-bdseq-included] A bdSeq number as a metric MUST be included in the payload.**
- **[tck-id-topics-nbirth-bdseq-matching] 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.
- **[tck-id-topics-nbirth-bdseq-increment] The bdSeq number MUST start at zero and increment by one on every new MQTT CONNECT packet.**
- **[tck-id-topics-nbirth-rebirth-metric] The NBIRTH message MUST include a metric with the name 'Node Control/Rebirth'. It MUST be of datatype boolean and have a value of false.**
  - The 'Node Control/Rebirth' metric is used by Host Application(s) to request a new NBIRTH and DBIRTH(s) from an Edge Node.

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)

- **[tck-id-topics-ndata-topic] The Edge Node data topic for a Sparkplug Edge Node MUST be of the form 'namespace/group_id/NDATA/edge_node_id' where the namespace is replaced with the specific namespace for this version of Sparkplug and the group_id and edge_node_id are replaced with the Group and Edge Node ID for this specific Edge Node.**

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.

- **[tck-id-topics-ndata-mqtt] NDATA messages MUST be published with MQTT QoS equal to 0 and retain equal to false.**
- **[tck-id-topics-ndata-seq-num] 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.**
- **[tck-id-topics-ndata-timestamp] The NDATA MUST include a timestamp denoting the date and time the message was sent from the Edge Node.**

- **[tck-id-topics-ndata-payload] 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)

- **[tck-id-topics-ndeath-topic] The Edge Node Death Certificate topic for a Sparkplug Edge Node MUST be of the form 'namespace/group_id/NDEATH/edge_node_id' where the namespace is replaced with the specific namespace for this version of Sparkplug and the group_id and edge_node_id are replaced with the Group and Edge Node ID for this specific Edge Node.**

Payload (NDEATH)

- **[tck-id-topics-ndeath-payload] 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, **[tck-id-topics-ndeath-seq] 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.

- **[tck-id-topics-ncmd-topic] The Edge Node command topic for a Sparkplug Edge Node MUST be of the form 'namespace/group_id/NCMD/edge_node_id' where the namespace is replaced with the specific namespace for this version of Sparkplug and the group_id and edge_node_id are replaced with the Group and Edge Node ID for this specific Edge Node.**

Payload (NCMD)

The NCMD message requires the following payload components.

- **[tck-id-topics-ncmd-mqtt] NCMD messages MUST be published with MQTT QoS equal to 0 and retain equal to false.**
- **[tck-id-topics-ncmd-timestamp] The NCMD MUST include a timestamp denoting the date and time the message was sent from the Host Application's MQTT client.**
- **[tck-id-topics-ncmd-payload] The NCMD MUST include the metrics that need to be written to on the Edge Node.**

4.2.2. Device/Sensor

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 and time this message was received.

Topic (DBIRTH)

- **[tck-id-topics-dbirth-topic] The Device Birth topic for a Sparkplug Device MUST be of the form 'namespace/group_id/DBIRTH/edge_node_id/device_id' where the namespace is replaced with the specific namespace for this version of Sparkplug and the group_id, edge_node_id, and device_id are replaced with the Group, Edge Node, and Device ID for this specific Device.**

Payload (DBIRTH)

The DBIRTH message requires the following payload components.

- **[tck-id-topics-dbirth-mqtt] DBIRTH messages MUST be published with MQTT QoS equal to 0 and retain equal to false.**
- **[tck-id-topics-dbirth-seq] 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.**
- **[tck-id-topics-dbirth-timestamp] The DBIRTH MUST include a timestamp denoting the date and time the message was sent from the Edge Node.**
- **[tck-id-topics-dbirth-metric-reqs] The DBIRTH MUST include every metric the Edge Node will ever report on.**
- **[tck-id-topics-dbirth-metrics] At a minimum each metric MUST include the metric name, metric datatype, and 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)

- **[tck-id-topics-ddata-topic] The Device command topic for a Sparkplug Device MUST be of the form 'namespace/group_id/DDATA/edge_node_id/device_id' where the namespace is replaced with the specific namespace for this version of Sparkplug and the group_id, edge_node_id, and device_id are replaced with the Group, Edge Node, and Device ID for this specific Device.**

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.

- **[tck-id-topics-ddata-mqtt] DDATA messages MUST be published with MQTT QoS equal to 0 and retain equal to false.**
- **[tck-id-topics-ddata-seq-num] 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.**

- **[tck-id-topics-ddata-timestamp] The DDATA MUST include a timestamp denoting the date and time the message was sent from the Edge Node.**
- **[tck-id-topics-ddata-payload] 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)

- **[tck-id-topics-ddeath-topic] The Device Death Certificate topic for a Sparkplug Device MUST be of the form 'namespace/group_id/DDEATH/edge_node_id/device_id' where the namespace is replaced with the specific namespace for this version of Sparkplug and the group_id, edge_node_id, and device_id are replaced with the Group, Edge Node, and Device ID for this specific Device.**

Payload (DDEATH)

The DDEATH message requires the following payload components.

- **[tck-id-topics-ddeath-mqtt] DDEATH messages MUST be published with MQTT QoS equal to 0 and retain equal to false.**
- **[tck-id-topics-ddeath-seq-num] 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)

- **[tck-id-topics-dcmd-topic] The Device command topic for a Sparkplug Device MUST be of the form 'namespace/group_id/DCMD/edge_node_id/device_id' where the namespace is replaced with the specific namespace for this version of Sparkplug and the group_id, edge_node_id, and device_id are replaced with the Group, Edge Node, and Device ID for this specific Device.**

Payload (DCMD)

The DCMD message requires the following payload components.

- **[tck-id-topics-dcmd-mqtt] DCMD messages MUST be published with MQTT QoS equal to 0 and retain equal to false.**
- **[tck-id-topics-dcmd-timestamp] The DCMD MUST include a timestamp denoting the date and time the message was sent from the Host Application's MQTT client.**
- **[tck-id-topics-dcmd-payload] The DCMD MUST include the metrics that need to be written to on the Device.**

4.2.3. Birth Certificate Message (STATE)

[tck-id-host-topic-phid-birth-message] 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 application.

- **[tck-id-host-topic-phid-birth-qos] The MQTT Quality of Service (QoS) MUST be set to 1**
- **[tck-id-host-topic-phid-birth-retain] The MQTT retain flag for the Birth Certificate MUST be set to TRUE**

Birth Certificate Topic (STATE)

The topic used for the Host Birth Certificate is identical to the topic used for the Death Certificate. **[tck-id-host-topic-phid-birth-topic] The Sparkplug Host Application Birth topic MUST be of the form spBv1.0/STATE/sparkplug_host_id where the sparkplug_host_id must be replaced with the specific Spakrplug Host ID of this Sparkplug Host Application.**

- **[tck-id-host-topic-phid-birth-sub-required] The Sparkplug Host Application MUST subscribe to its own spBv1.0/STATE/sparkplug_host_id and the appropriate spBv1.0 topic(s) immediately after successfully connecting to the MQTT Server.**
  - An 'appropriate' spBv1.0 topic could simply be 'spBv1.0/'. *However, it may also make sense for a Host Application to subscribe only to a specific Sparkplug Group. For example subscribing to spBv1.0/Group1/* is also valid. A Host Application could even issue a subscription to subscribe to only a single Sparkplug Edge Node using this: spBv1.0/Group1/+/EdgeNode1/#. A Sparkplug Host Application could subscribe to a combination of specific Sparkplug Groups and/or Edge Nodes as well.
- **[tck-id-host-topic-phid-birth-required] The Sparkplug Host Application MUST publish a Sparkplug Host Application BIRTH message to the MQTT Server immediately after successfully subscribing its own spBv1.0/STATE/sparkplug_host_id topic.**

Birth Certificate Payload (STATE)

- **[tck-id-host-topic-phid-birth-payload] The Birth Certificate Payload MUST be JSON UTF-8 data. It MUST include two key/value pairs where the one key MUST be 'online' and it's value is a boolean 'true'. The other key MUST be 'timestamp' and**

the value MUST be a numeric value representing the current UTC time in milliseconds since Epoch.

- **[tck-id-host-topic-phid-birth-payload-timestamp] The timestamp metric value MUST be the same timestamp value set in the immediately prior MQTT CONNECT packet's Will Message payload.**

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

4.2.4. 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.

- **[tck-id-host-topic-phid-death-qos] The MQTT Quality of Service (QoS) MUST be set to 1**
- **[tck-id-host-topic-phid-death-retain] The MQTT retain flag for the Birth Certificate MUST be set to TRUE**

Death Certificate Topic (STATE)

- **[tck-id-host-topic-phid-death-topic] The Sparkplug Host Application Death topic MUST be of the form spBv1.0/STATE/sparkplug_host_id where the sparkplug_host_id must be replaced with the specific Sparkplug Host ID of this Sparkplug Host Application.**
- **[tck-id-host-topic-phid-death-required] The Sparkplug Host Application MUST provide a Will message in the MQTT CONNECT packet**
  - This is the Sparkplug Host Application DEATH certificate

Death Certificate Payload (STATE)

- **[tck-id-host-topic-phid-death-payload] The STATE Death Certificate Payload MUST be JSON UTF-8 data. It MUST include two key/value pairs where one key MUST be 'online' and it's value is a boolean 'false'. The other key MUST be 'timestamp' and the value MUST be a numeric value representing the current UTC time in milliseconds since Epoch.**
- **[tck-id-host-topic-phid-death-payload-connect] The Death Certificate's used in the MQTT CONNECT packet Will message MUST use a timestamp value that represents the current UTC time at the time of the CONNECT packet is sent to the MQTT Server.**
- **[tck-id-host-topic-phid-death-payload-disconnect-clean] If a Host Application is disconnecting cleanly using MQTT DISCONNECT packet, the Host Application MUST publish a Death Certificate payload before sending the MQTT DISCONNECT packet with the timestamp set to the current UTC time the disconnect is occurring.**
- **[tck-id-host-topic-phid-death-payload-disconnect-with-no-disconnect-packet] If a Host Application is disconnecting and not using an MQTT DISCONNECT packet, the Host Application MUST publish a Death Certificate payload before terminating the**

MQTT connection with the timestamp set to the current UTC time the disconnect is occurring.

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 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 is connected 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 shows 'online=true'.

5.1. Timestamps in Sparkplug

An important aspect of Sparkplug is its use of time. All timestamps must be in Coordinated Universal Time (UTC). In order to ensure this is the case, all Sparkplug Edge Nodes and Sparkplug Host Applications must have an accurate mechanism for ensuring their clocks remain accurate. This is typically left to the system operating system using technologies such as Network Time Protocol (NTP). Regardless of the mechanism used, ensuring all timestamps are accurate and in UTC is critical to all timestamps in Sparkplug.

5.2. Case Sensitivity in Sparkplug

The MQTT specification states that MQTT topics are case sensitive [MQTTV5-4.7.3]. For example, the topic a/b is different than A/b. Sparkplug in turn is also case sensitive with regard to both topics as well as metric names. So, a metric 'temperature' is not the same as the metric 'Temperature'. However, this generally should be avoided. Many Host Applications may not be able to differentiate the two metric names as unique. Many databases are case-insensitive and would not be able to handle this situation well.

- **[tck-id-case-sensitivity-sparkplug-ids] Edge Nodes in a Sparkplug environment SHOULD NOT have Sparkplug IDs (Group, Edge Node, or Device IDs) that when converted to lower case match**
  - For example there should not be two different Edge Nodes publishing NBIRTH messages on these two topics spBv1.0/Group1/NBIRTH/EdgeNode1 and spBv1.0/group1/NBIRTH/edgenode1
- **[tck-id-case-sensitivity-metric-names] An Edge Node SHOULD NOT publish metric names that when converted to all lower case match.**
  - For example a DBIRTH should not contain a metric 'a' and another metric 'A'.

5.3. 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.

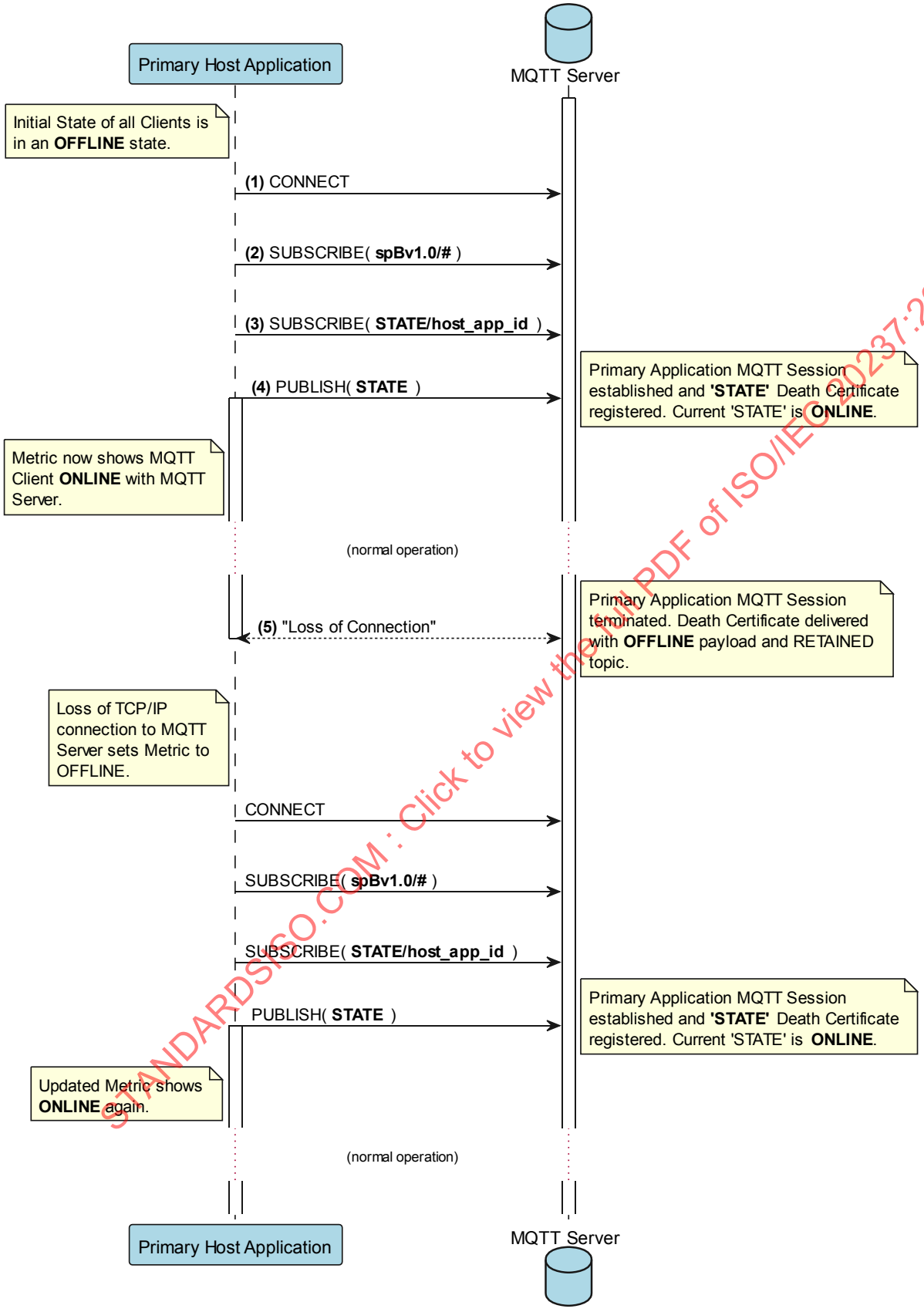

Figure 4 - Host Session Establishment

The session diagram in Figure 4 - 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 CONNECT Control Packet with a Will QoS set to 1 and Will Retain flag set to true.

   [tck-id-message-flow-phid-sparkplug-clean-session-311] The CONNECT Control Packet for all Sparkplug Host Applications when using MQTT 3.1.1 MUST set the MQTT 'Clean Session' flag to true.

   [tck-id-message-flow-phid-sparkplug-clean-session-50] The CONNECT Control Packet for all Sparkplug Host Applications when using MQTT 5.0 MUST set the the MQTT 'Clean Start' flag to true and the 'Session Expiry Interval' to 0.

   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.

   [tck-id-message-flow-phid-sparkplug-subscription] 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. **[tck-id-message-flow-phid-sparkplug-state-publish] 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 established, the Sparkplug Host Application MUST publish a new STATE message.**

   [tck-id-message-flow-phid-sparkplug-state-publish-payload] The Host Application Birth Certificate Payload MUST be JSON UTF-8 data. It MUST include two key/value pairs where one key MUST be 'online' and its value is a boolean 'true'. The other key MUST be 'timestamp' and the value MUST be the same value set in the immediately prior MQTT CONNECT packet's Will Message payload.

   [tck-id-message-flow-phid-sparkplug-state-publish-payload-timestamp] The timestamp value in the Host Application Birth Certificate payload MUST be the same value set in the immediately prior MQTT CONNECT packet's Will Message payload.

   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=true 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.

[tck-id-message-flow-hid-sparkplug-state-message-delivered] After publishing its own Host Application STATE message, if at any point the Host Application is delivered a STATE message on its own Host Application ID with a 'online' value of false, it MUST immediately republish its STATE message to the same MQTT Server with a 'online' value of true and the 'timestamp' set to the same value that was used for the timestamp in its own prior MQTT CONNECT packet Will Message payload.

5.4. Edge Node Session Establishment

Prior to sending an NBIRTH message, the MQTT client associated with the Edge Node must subscribe to receive NCMD messages with the following rules.

- **[tck-id-message-flow-edge-node-ncmd-subscribe] The MQTT client associated with the Edge Node MUST subscribe to a topic of the form 'spBv1.0/group_id/NCMD/edge_node_id' where group_id is the Sparkplug Group ID and the edge_node_id is the Sparkplug Edge Node ID for this Edge Node. It MUST subscribe on this topic with a QoS of 1.**
  - This subscription is mandatory as Edge Nodes MUST be able to respond to 'rebirth requests'.

After subscribing, the Edge Node must follow these additional rules.

- **[tck-id-message-flow-edge-node-birth-publish-connect] Any Edge Node in the MQTT infrastructure MUST establish an MQTT Session prior to publishing NBIRTH and DBIRTH messages.**
- **[tck-id-message-flow-edge-node-birth-publish-will-message] When a Sparkplug Edge Node sends its MQTT CONNECT packet, it MUST include a Will Message.**
- **[tck-id-message-flow-edge-node-birth-publish-will-message-topic] The Edge Node's MQTT Will Message's topic MUST be of the form 'spBv1.0/group_id/NDEATH/edge_node_id' where group_id is the Sparkplug Group ID and the edge_node_id is the Sparkplug Edge Node ID for this Edge Node**
- **[tck-id-message-flow-edge-node-birth-publish-will-message-payload] The Edge Node's MQTT Will Message's payload MUST be a Sparkplug Google Protobuf encoded payload.**
- **[tck-id-message-flow-edge-node-birth-publish-will-message-payload-bdSeq] The Edge Node's MQTT Will Message's payload MUST include a metric with the name of 'bdSeq', the datatype of INT64, and the value MUST be incremented by one from the value in the previous MQTT CONNECT packet unless the value would be greater than 255. If in the previous NBIRTH a value of 255 was sent, the next MQTT Connect packet Will Message payload bdSeq number value MUST have a value of 0.**

- **[tck-id-message-flow-edge-node-birth-publish-will-message-qos] The Edge Node's MQTT Will Message's MQTT QoS MUST be 1.**
- **[tck-id-message-flow-edge-node-birth-publish-will-message-will-retained] The Edge Node's MQTT Will Message's retained flag MUST be set to false.**

Edge Nodes can be configured to support the concept of a 'Primary Host Application'. In this case, the Edge Node must wait until the Primary Host Application is online and subscribed to Sparkplug messages before the Edge Node publishes its NBIRTH and DBIRTH messages. Specifying a Primary Host is not required for an Edge Node. But it is often desired. For example say an Edge Node is in a Sparkplug environment and there is a single consuming Host Application that historizes the data. It would not be beneficial for the Sparkplug Edge Node to publish data if the Host Application is not connected and subscribed to messages. Instead, it would be better for the Edge Node to store data while the Host Application is offline. Once the Host Application is properly connected, it could then send all of its stored data and continue publishing normally. Once the Sparkplug Edge Node has successfully connected to the MQTT Server, it must publish a NBIRTH message. The NBIRTH message must follow the following rules. Note if Primary Host is configured for the Edge Node, it must also wait until the Primary Host denotes it is online before the Edge Node publishes its NBIRTH message.

- **[tck-id-message-flow-edge-node-birth-publish-phid-wait] If the Edge Node is configured to wait for a Primary Host Application it MUST verify the Primary Host Application is online via the STATE topic before publishing NBIRTH and DBIRTH messages.**
  - **[tck-id-message-flow-edge-node-birth-publish-phid-wait-id] If the Edge Node is configured to wait for a Primary Host Application it MUST validate the Host Application ID as the last token in the STATE message topic string matches the configured Primary Host Application ID for this Edge Node.**
  - **[tck-id-message-flow-edge-node-birth-publish-phid-wait-online] If the Edge Node is configured to wait for a Primary Host Application it MUST validate the 'online' boolean flag is true in the STATE message payload before considering the Primary Host Application to be online and active.**
  - **[tck-id-message-flow-edge-node-birth-publish-phid-wait-timestamp] If the Edge Node is configured to wait for a Primary Host Application it MUST validate the timestamp value is greater than or equal to the previous STATE message timestamp value in the STATE message payload before considering the Primary Host Application to be online and active. If no previous STATE message timestamp value has been received by this Edge Node it MUST consider the incoming STATE message to be the latest/valid.**
- **[tck-id-message-flow-edge-node-birth-publish-nbirth-topic] The Edge Node's NBIRTH MQTT topic MUST be of the form 'spBv1.0/group_id/NBIRTH/edge_node_id' where group_id is the Sparkplug Group ID and the edge_node_id is the Sparkplug Edge Node ID for this Edge Node**
- **[tck-id-message-flow-edge-node-birth-publish-nbirth-payload] The Edge Node's NBIRTH payload MUST be a Sparkplug Google Protobuf encoded payload.**
- **[tck-id-message-flow-edge-node-birth-publish-nbirth-payload-bdSeq] The Edge Node's NBIRTH payload MUST include a metric with the name of 'bdSeq' the**

datatype of INT64 and the value MUST be the same as the previous MQTT CONNECT packet.

- **[tck-id-message-flow-edge-node-birth-publish-nbirth-qos] The Edge Node's NBIRTH MQTT QoS MUST be 0.**
- **[tck-id-message-flow-edge-node-birth-publish-nbirth-retained] The Edge Node's NBIRTH retained flag MUST be set to false.**
- **[tck-id-message-flow-edge-node-birth-publish-nbirth-payload-seq] The Edge Node's NBIRTH payload MUST include a 'seq' number that is between 0 and 255 (inclusive).**
  - This will become the starting sequence number which all following messages will include a sequence number that is one more than the previous up to 255 where it wraps back to zero.
- **[tck-id-message-flow-edge-node-birth-publish-phid-offline] If the Edge Node is configured to wait for a Primary Host Application, it is connected to the MQTT Server, and receives a STATE message on its configured Primary Host, the timestamp value in the payload is greater than or equal to the timestamp value included in the prior 'online' STATE message, and the 'online' value is false, it MUST immediately publish an NDEATH message and disconnect from the MQTT Server and start the connection establishment process over.**
  - If the Edge Node did not previously receive a STATE message from this Primary Host Application, it can not check the timestamp value against the previous value. In this case it MUST honor the 'online' boolean status flag as denoted in the payload.

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 in MQTT 3.1.1 or the 'Clean Start' flag in MQTT 5.0. All types of MQTT clients (both Host and Edge Nodes) in a Sparkplug system MUST always set the 'Clean Session' flag in the MQTT 3.1.1 CONNECT packet to true. When using MQTT 5.0 the 'Clean Start' flag must be set to true and the MQTT 'Session Expiry Interval' to zero.

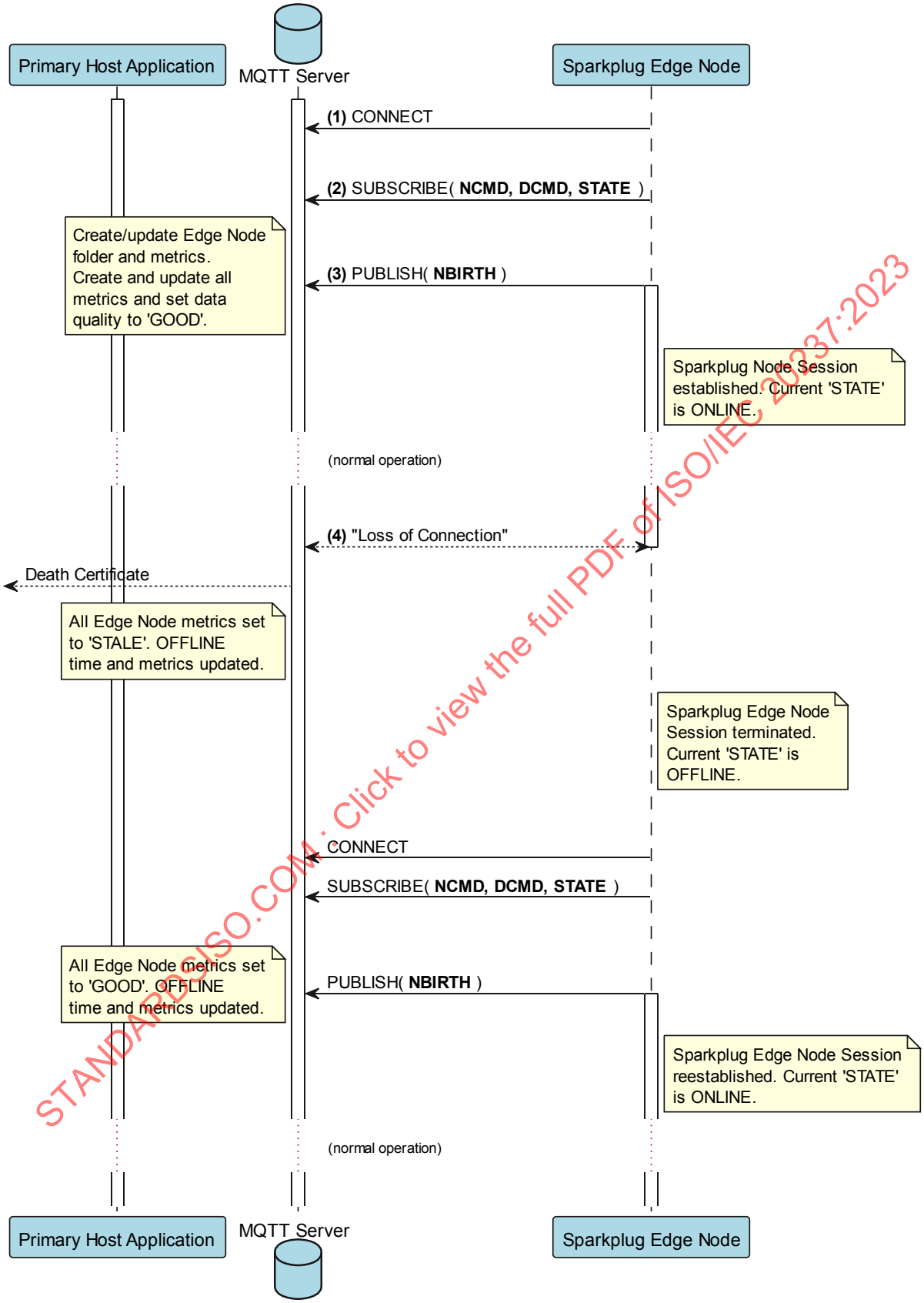

Figure 5 - Edge Node MQTT Session Establishment

The session diagram in Figure 5 - 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 ‘online=false’ 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.

5.5. Edge Node Session Termination

[tck-id-operational-behavior-edge-node-intentional-disconnect-ndeath] When an Edge Node disconnects intentionally, it MUST publish an NDEATH before terminating the connection.

[tck-id-operational-behavior-edge-node-intentional-disconnect-packet] Immediately following the NDEATH publish, a DISCONNECT packet MAY be sent to the MQTT Server.

- If an MQTT DISCONNECT packet is sent by the Edge Node, this signals to the MQTT Server that the Will Message MUST not be delivered by the MQTT Server to subscribers

of that message. These subscribers are typically Sparkplug Host Applications. This is why a Death message MUST be published before disconnecting from the MQTT Server. It ensures Edge Nodes are notified the Edge Node is now offline.

- If an MQTT DISCONNECT packet is not sent by the Sparkplug Edge Node, the MQTT Server will eventually deliver the Will Message (Death Certificate) to the subscribers. However, this can take some time to occur based on when the MQTT Server detects that the Edge Node is no longer connected. By sending the Death Certificate before disconnecting without sending an MQTT DISCONNECT packet, we are ensuring that a Death message will be delivered to subscribing clients promptly. The fact that a second Death message will arrive when the Will Message is delivered is not significant. This is because the Will Message Death message will contain a bdSeq number that matches the bdSeq number that is published by the Edge Node immediately before the disconnect. Because it has a duplicate bdSeq, the Will Message Death message MUST be ignored by the subscribing Sparkplug Host Application clients.

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 or if an MQTT Server delivers an NDEATH on behalf of an Edge Node, it is implied that all of the Edge Node's associated Devices are also offline. In addition, it is also implied that all metrics in the previous associated NBIRTH and all DBIRTHs in this Sparkplug session under that Edge Node are now STALE.

For the following normative statements it is up to the designers of the Sparkplug Host Application with regard to how they 'mark' the Sparkplug Edge Node or Sparkplug Device as 'offline'. It is also up to the designers of the Sparkplug Host Application on how they 'mark' a metric as STALE. This is an important aspect of Sparkplug in that an NDEATH means the data was accurate at a time, but now that the MQTT session has been lost can no longer be considered current or up to date.

Because an NDEATH may be sent on behalf of an Edge Node by an MQTT Server in the MQTT Will Message, the Sparkplug payload timestamp does not represent the time that the Edge Node actually went offline. As a result, the timestamp associated with NDEATH events must use the timestamp of receipt on the Sparkplug Host Application. This is in part why Sparkplug Edge Nodes and Host Applications must have synced system clocks and all Sparkplug timestamps must be in UTC time.

- **[tck-id-operational-behavior-edge-node-termination-host-action-ndeath-node-offline] Immediately after receiving an NDEATH from an Edge Node, Host Applications MUST mark the Edge Node as offline using the current Host Application's system UTC time**
- **[tck-id-operational-behavior-edge-node-termination-host-action-ndeath-node-tags-stale] Immediately after receiving an NDEATH from an Edge Node, Host Applications MUST mark all metrics that were included in the previous NBIRTH as STALE using the current Host Application's system UTC time**
- **[tck-id-operational-behavior-edge-node-termination-host-action-ndeath-devices-offline] Immediately after receiving an NDEATH from an Edge Node, Host Applications MUST mark all Sparkplug Devices associated with the Edge Node as offline using the current Host Application's system UTC time**

- **[tck-id-operational-behavior-edge-node-termination-host-action-ndeath-devices-tags-stale] Immediately after receiving an NDEATH from an Edge Node, Host Applications MUST mark all of the metrics that were included with associated Sparkplug Device DBIRTH messages as STALEusing the current Host Application's system UTC time**

For the following assertions an 'online STATE message' is one where a Host Application's JSON payload has the 'online' key's value set to true. An 'offline STATE message' is one where the Host Application's JSON payload has the 'online' key's value set to false.

If the Edge Node is configured to use a Primary Host Application, it must also watch for 'STATE' messages from the Primary Host Application via an MQTT subscription. If the Primary Host Application denotes it is offline, the Edge Node must disconnect from the current MQTT server following these rules:

- **[tck-id-operational-behavior-edge-node-termination-host-offline] If the Edge Node is configured to use a Primary Host Application, it MUST disconnect from the current MQTT Server if the online JSON value is false and the timestamp value is greater than or equal to the previous online STATE message timestamp value.**
  - **[tck-id-operational-behavior-edge-node-termination-host-offline-reconnect] If the Edge Node disconnects after being in a Sparkplug session due to a valid 'offline STATE message', it MUST attempt to connect to the next MQTT Server in its connection list to start the session establishment procedure over again.**
- **[tck-id-operational-behavior-edge-node-termination-host-offline-timestamp] Consider an Edge Node that is configured to use a Primary Host Application and the Edge Node is connected and publishing. Then it receives an offline STATE message. It MUST NOT disconnect if the timestamp value is less than the value from the previous online STATE message.**
  - It must not disconnect because the older timestamp value indicates the Host Application MQTT session that is being denoted as lost is not the one the current session the Host Application has established with the MQTT Server. Due to how an MQTT connection can be lost it is possible and likely that an old Host Application death message could be delivered after a new Host Application MQTT session is established. In this case, the timestamp value on the incoming death message will be older than the current timestamp value. For this reason, it must be ignored.

5.6. Device Session Establishment

The aim of the Sparkplug Specification is to enable the transport of real time process variable information from existing and new end devices measuring, monitoring, and controlling a physical process into an MQTT infrastructure subsequently a Sparkplug Host Application. In the context of this document an MQTT Device can represent anything from existing legacy poll/response driven PLCs, RTUs, HART Smart Transmitters, 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.

Prior to sending a DBIRTH message, if the Device supports 'writing to outputs' the MQTT client associated with the Sparkplug Device must subscribe to receive DCMD messages with the following rules.

- **[tck-id-message-flow-device-dcmd-subscribe] If the Device supports writing to outputs, the MQTT client associated with the Device MUST subscribe to a topic of the form 'spBv1.0/group_id/DCMD/edge_node_id/device_id' where group_id is the Sparkplug Group ID the edge_node_id is the Sparkplug Edge Node ID and the device_id is the Sparkplug Device ID for this Device. It MUST subscribe on this topic with a QoS of 1.**

A Device can publish a DBIRTH as long as an NBIRTH has been sent previously and the MQTT session is active. The DBIRTH message must follow the following rules.

- **[tck-id-message-flow-device-birth-publish-nbirth-wait] The NBIRTH message MUST have been sent within the current MQTT session prior to a DBIRTH being published.**
- **[tck-id-message-flow-device-birth-publish-dbirth-topic] The Device's DBIRTH MQTT topic MUST be of the form 'spBv1.0/group_id/DBIRTH/edge_node_id/device_id' where group_id is the Sparkplug Group ID the edge_node_id is the Sparkplug Edge Node ID and the device_id is the Sparkplug Device ID for this Device.**
- **[tck-id-message-flow-device-birth-publish-dbirth-match-edge-node-topic] The Device's DBIRTH MQTT topic group_id and edge_node_id MUST match the group_id and edge_node_id that were sent in the prior NBIRTH message for the Edge Node this Device is associated with.**
- **[tck-id-message-flow-device-birth-publish-dbirth-payload] The Device's DBIRTH payload MUST be a Sparkplug Google Protobuf encoded payload.**
- **[tck-id-message-flow-device-birth-publish-dbirth-qos] The Device's DBIRTH MQTT QoS MUST be 0.**
- **[tck-id-message-flow-device-birth-publish-dbirth-retained] The Device's DBIRTH retained flag MUST be set to false.**
- **[tck-id-message-flow-device-birth-publish-dbirth-payload-seq] The Device's DBIRTH payload MUST include a 'seq' number that is between 0 and 255 (inclusive) and be one more than was included in the prior Sparkplug message sent from the Edge Node associated with this Device.**

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

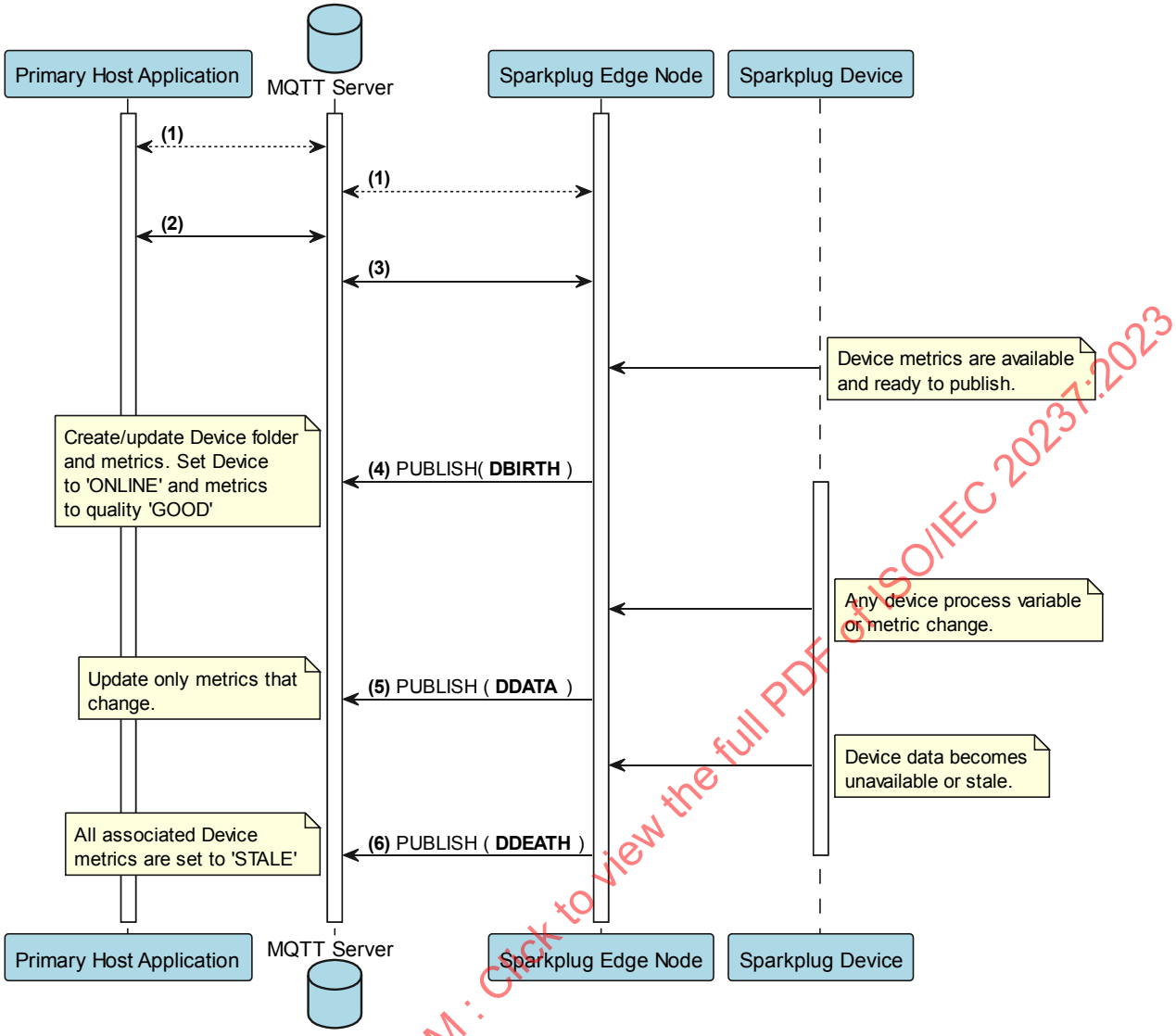

Figure 6 - MQTT Device Session Establishment

The session diagram in Figure 6 - 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 a Raspberry Pi. 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 the 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. Time based reporting is not explicitly disallowed by the Sparkplug Specification but it is discouraged and often unnecessary.

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.

5.7. Device Session Termination

[tck-id-operational-behavior-device-ddeath] If a Sparkplug Edge Node loses connection with an attached Sparkplug Device, it MUST publish a DDEATH message on behalf of the device.

When a Sparkplug Device goes offline by having its DDEATH published by an Edge Node, it allows Sparkplug Host Applications to know that the Sparkplug Device is no longer reporting current and accurate values to the Edge Node. Therefore the Edge Node is not able to report live/accurate data values on behalf of the Sparkplug Device to the MQTT Server or in turn to Sparkplug Host Applications. As a result the Sparkplug Host Applications must mark the Device as offline and denote the Sparkplug Device's tags as stale.

For the following normative statements it is up to the designers of the Sparkplug Host Application with regard to how they 'mark' the Sparkplug Device as 'offline'. It is also up to the designers of the Sparkplug Host Application on how they 'mark' a metric as STALE. This is an important aspect of Sparkplug in that an DDEATH means the data was accurate at a time, but now that the connection between the Sparkplug Edge Node and the Sparkplug Device has been lost can no longer be considered current or up to date.

The DDEATH is sent on behalf of a Sparkplug Device by a Sparkplug Edge Node. Because of this, the Sparkplug payload timestamp associated with a DDEATH is considered accurate and must be used as the timestamp for a Sparkplug Device being marked as offline and for its associated metrics being set to STALE.

[tck-id-operational-behavior-edge-node-termination-host-action-ddeath-devices-offline] Immediately after receiving an DDEATH from an Edge Node, Host Applications MUST mark the Sparkplug Device associated with the Edge Node as offline using the timestamp in the DDEATH payload

[tck-id-operational-behavior-edge-node-termination-host-action-ddeath-devices-tags-stale] Immediately after receiving an DDEATH from an Edge Node, Host Applications MUST mark all of the metrics that were included with the associated Sparkplug Device DBIRTH messages as STALE using the timestamp in the DDEATH payload

5.8. Sparkplug Host Applications

As noted above, the Sparkplug Host Application has the required permissions to send commands to Edge Nodes and Sparkplug Devices because Edge Nodes need to know that the Primary Host Application is connected to the same MQTT Server that it is connected to or to walk to another server 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 through 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.

5.9. 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.

For example, if a Sparkplug Host Application receives messages from an Edge Node with sequence numbers 1, 2, and 4 then at the time the message with a sequence number 4 arrives, a timer SHOULD be started within the Host Application. This is the start of the Reordering Timeout timer. A message 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. This ensures that the session state is properly reestablished. If a message with a sequence number of 3 arrives before the Reorder Timeout occurs then 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. For example, a Host Application that is a time series database may want to insert all data that arrives regardless of the message order. However, a rules engine Host Application may require that

messages are processed in order of their sequence numbers to preserve the order of events as they occurred at the Edge Node.

- **[tck-id-operational-behavior-host-reordering-param] Sparkplug Host Applications SHOULD provide a configurable 'Reorder Timeout' parameter**
- **[tck-id-operational-behavior-host-reordering-start] If a Sparkplug Host Application is configured with a 'reordering timeout' parameter and a message arrives with an out of order sequence number, the Host Application MUST start a timer denoting the start of the Reorder Timeout window**
- **[tck-id-operational-behavior-host-reordering-rebirth] If a Sparkplug Host Application is configured with a 'reordering timeout' parameter and the Reorder Timeout elapses and the missing message(s) have not been received, the Sparkplug Host Application MUST 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
- **[tck-id-operational-behavior-host-reordering-success] If the missing message(s) that triggered the start of the Reorder Timeout timer arrive before the reordering timer elapses, the timer MUST be terminated and normal operation in the Host Application MUST continue until another out of order message arrives.**

5.10. 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.

[tck-id-operational-behavior-primary-application-state-with-multiple-servers-state-subs] When using multiple MQTT Servers and Edge Nodes are configured with a Primary Host Application, the Primary Host Application instance MUST be configured to publish a STATE Birth Certificate and all Edge Nodes configured with a Primary Host Application MUST subscribe to this STATE message.

[tck-id-operational-behavior-primary-application-state-with-multiple-servers-state] 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.

[tck-id-operational-behavior-primary-application-state-with-multiple-servers-single-server] 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 the Primary Host Application. Since the STATE message is published with the MQTT RETAIN flag set, MQTT will guarantee that the last STATE message is always available. The Edge Node should examine the JSON payload of this message to ensure that the value of the 'online' key is true. If the value is false, this indicates the Primary Application has lost its MQTT Session to this particular MQTT Server.

[tck-id-operational-behavior-primary-application-state-with-multiple-servers-walk] 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.

[tck-id-operational-behavior-edge-node-birth-sequence-wait] The Edge Node MUST also wait to publish its BIRTH sequence until an online=true STATE message is received by the Edge Node. This use of the STATE message in this manner ensures that any loss of connectivity between an MQTT Server and 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:

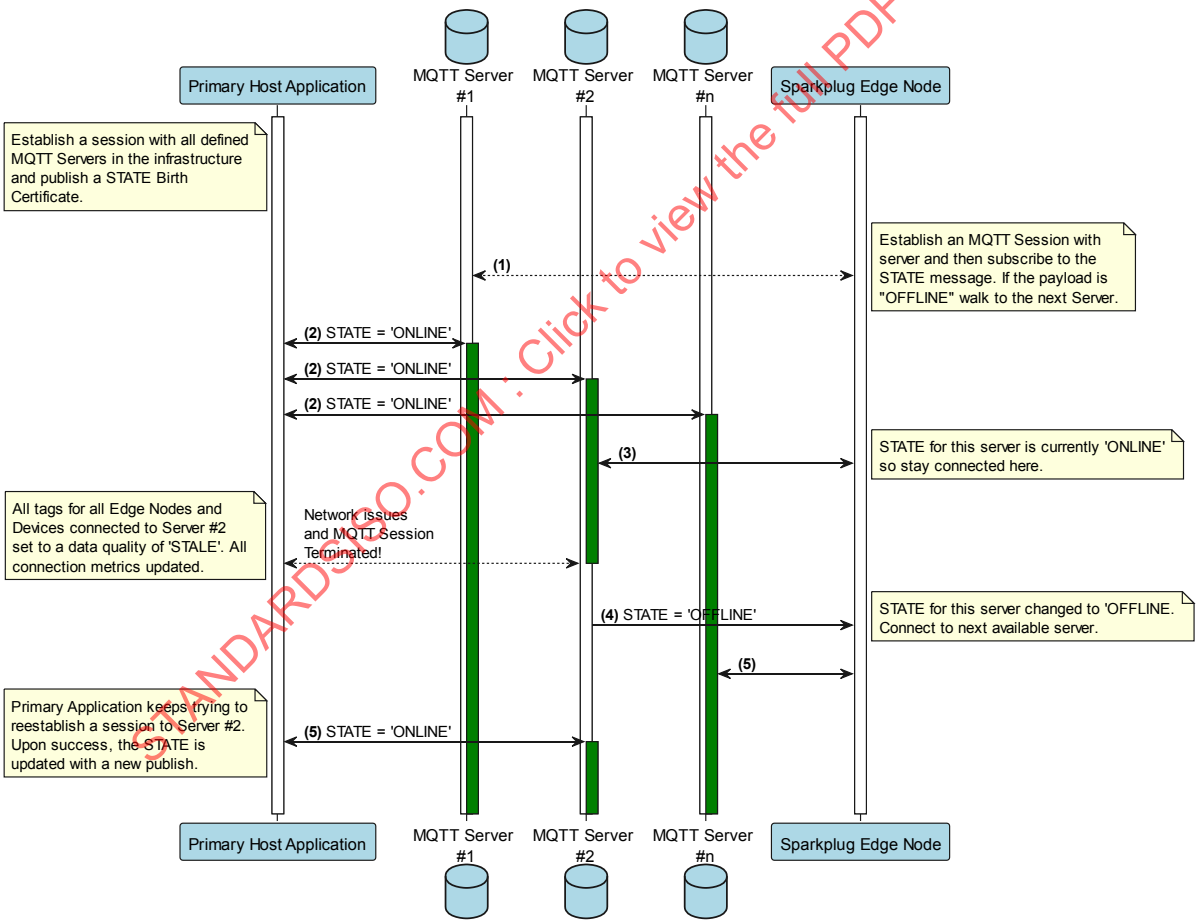

Figure 7 – Primary Host Application STATE flow diagram

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 online=false 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 online=false 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=true. In both of these messages the timestamp value must match each other and represent the current connection time. The timestamp value must be a JSON number and represent the number of UTC milliseconds since Epoch.

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 JSON payload that has online=true. 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 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 deliver the Death Certificate to clients that are currently subscribed to it. Upon receipt of the Primary Host Application Death Certificate each Edge Node will disconnect from the current MQTT Server and connect to the next MQTT Server in its list. Before the Edge Node disconnects and connects to the next MQTT Server it must validate that the JSON payload denotes online=false and the timestamp value is greater than or equal to the prior STATE message timestamp value from that Host Application's BIRTH message.

5. The Edge Node connects to the next available MQTT Server and since the current STATE on this server is online=true, it can stay connected. In the meantime, the network disruption between the 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 updated Birth Certificate of online=true. Now MQTT Server #2 is ready to accept new Edge Node session requests.

5.11. 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.

It is also important to note that an Access Control List (ACL) can be used to allow one or more Sparkplug Host Applications to publish NCMD and DCMD messages to one or more Edge Nodes. Furthermore the ability to publish NCMD or DCMD messages by other Sparkplug Host Applications could be blocked. The decoupled nature of the commands and data messages allows for this type of granular access and control.

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.

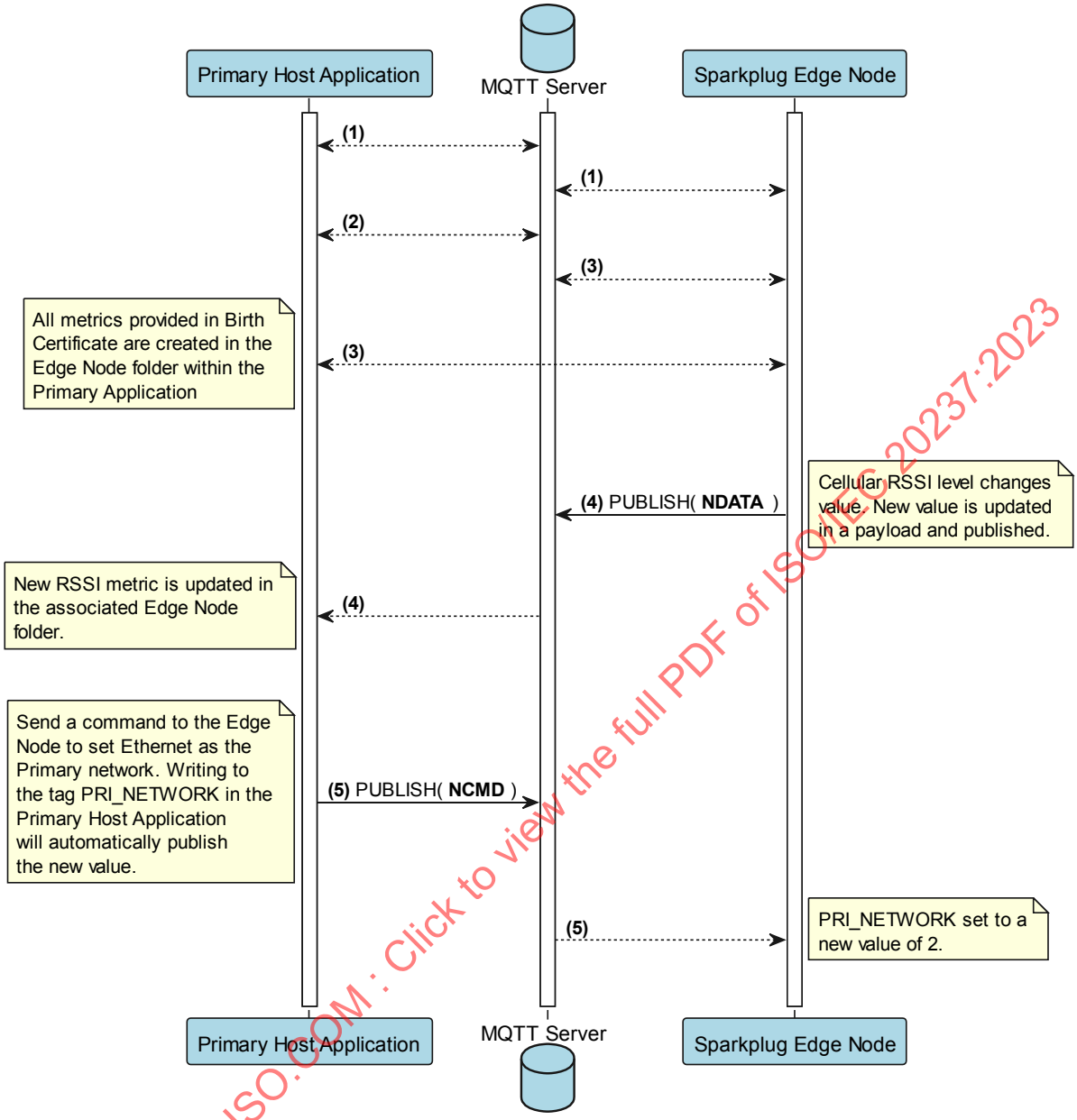

Figure 8 - Edge Node NDATA and NCMD Message Flow

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.

5.12. 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.

5.13. Sparkplug Host Application Session Establishment

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

- **[tck-id-operational-behavior-host-application-host-id] The sparkplug_host_id MUST be unique to all other Sparkplug Host IDs in the infrastructure.**
- **[tck-id-operational-behavior-host-application-connect-will] When a Sparkplug Host Application sends its MQTT CONNECT packet, it MUST include a Will Message.**
- **[tck-id-operational-behavior-host-application-connect-will-topic] The MQTT Will Message's topic MUST be of the form 'spBv1.0/STATE/sparkplug_host_id' where host_id is the unique identifier of the Sparkplug Host Application**
- **[tck-id-operational-behavior-host-application-connect-will-payload] The Death Certificate Payload MUST be JSON UTF-8 data. It MUST include two key/value pairs where one key MUST be 'online' and it's value is a boolean 'false'. The other key MUST be 'timestamp' and the value MUST be the same value that was used for the timestamp in its own prior MQTT CONNECT packet Will Message payload.**
- **[tck-id-operational-behavior-host-application-connect-will-qos] The MQTT Will Message's MQTT QoS MUST be 1 (at least once).**
- **[tck-id-operational-behavior-host-application-connect-will-retained] 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.

- **[tck-id-operational-behavior-host-application-connect-birth] The MQTT Client associated with the Sparkplug Host Application MUST send a birth message immediately after successfully connecting to the MQTT Server.**
- **[tck-id-operational-behavior-host-application-connect-birth-topic] The Host Application's Birth topic MUST be of the form 'spBv1.0/STATE/sparkplug_host_id' where host_id is the unique identifier of the Sparkplug Host Application**

- **[tck-id-operational-behavior-host-application-connect-birth-payload] The Birth Certificate Payload MUST be JSON UTF-8 data. It MUST include two key/value pairs where one key MUST be 'online' and it's value is a boolean 'true'. The other key MUST be 'timestamp' and the value MUST match the timestamp value that was used in the immediately prior MQTT CONNECT packet Will Message payload.**
- **[tck-id-operational-behavior-host-application-connect-birth-qos] The Host Application's Birth MQTT QoS MUST be 1 (at least once).**
- **[tck-id-operational-behavior-host-application-connect-birth-retained] The Host Application's Birth retained flag MUST be set to true.**

The following additional rule applies if the Host Application is connecting to more than one MQTT Server.

- **[tck-id-operational-behavior-host-application-multi-server-timestamp] The Host Application MUST maintain a STATE Message timestamp value on a per MQTT Server basis.**
  - For example if a connection is lost to one MQTT Server, when the Host Application reconnects and publishes a new STATE message, it must update the STATE Message timestamp for only this MQTT Server and not any others it may be connected to.

5.14. Sparkplug Host Application Session Termination

- **[tck-id-operational-behavior-host-application-termination] If the Sparkplug Host Application ever disconnects intentionally, it MUST publish a Death message with the following characteristics.**
- **[tck-id-operational-behavior-host-application-death-topic] The Sparkplug Host Application's Death topic MUST be of the form 'spBv1.0/STATE/sparkplug_host_id' where host_id is the unique identifier of the Sparkplug Host Application.**
- **[tck-id-operational-behavior-host-application-death-payload] The Death Certificate Payload registered as the MQTT Will Message in the MQTT CONNECT packet MUST be JSON UTF-8 data. It MUST include two key/value pairs where one key MUST be 'online' and it's value is a boolean 'false'. The other key MUST be 'timestamp' and the value MUST be a numeric value representing the current UTC time in milliseconds since Epoch.**
- **[tck-id-operational-behavior-host-application-death-qos] The Sparkplug Host Application's Death MQTT QoS MUST be 1 (at least once).**
- **[tck-id-operational-behavior-host-application-death-retained] The Sparkplug Host Application's Death retained flag MUST be set to true.**
- **[tck-id-operational-behavior-host-application-disconnect-intentional] In the case of intentionally disconnecting, an MQTT DISCONNECT packet MAY be sent immediately after the Death message is published.**
  - If an MQTT DISCONNECT packet is sent by the Host Application, this signals to the MQTT Server that the Will Message MUST not be delivered by the MQTT

Server to subscribers of that message. These subscribers are typically Sparkplug Edge Nodes. This is why a Death message MUST be published before disconnecting from the MQTT Server. It ensures Edge Nodes are notified the Host Application is now offline.

- If an MQTT DISCONNECT packet is not sent by the Sparkplug Host Application, the MQTT Server will eventually deliver the Will Message (Death Certificate) to the subscribers. However, this can take some time to occur based on when the MQTT Server detects that the Host Application is no longer connected. By sending the Death Certificate before disconnecting without sending an MQTT DISCONNECT packet, we are ensuring that a Death message will be delivered to subscribing clients promptly. The fact that a second Death message will arrive when the Will Message is delivered is not significant. This is because the Will Message Death message will contain a timestamp older than the timestamp that is published by the Host Application immediately before the disconnect. Because it has an older timestamp, the Will Message Death message MUST be ignored by the subscribing Sparkplug clients.

5.15. Sparkplug Host Application Receive Data

Sparkplug Host Applications are typically designed to receive data from Sparkplug Edge Nodes and optionally write commands back to them. What they do with that data is not specified by the Sparkplug specification. It is left to the implementor of a Sparkplug Host Application to define what they do with the data and what (if anything) they potentially write back to the Edge Nodes via CMD messages. Example Host Applications may use graphical interfaces or dashboards to display Edge Node data. Other Host Applications may insert data into a historical database for later querying. Other Host Applications may perform real-time analytics on the data as it flows from the Sparkplug Edge Nodes.

Because there is so much flexibility in what a Sparkplug Host Application may do with the data it receives there aren't hard requirements on what it does with it once it receives it. However, there are some things to consider:

- A Sparkplug Host Application MAY send Node Control/Rebirth NCMD messages if messages arrive out of sequence order and can not be reordered within the sequence reordering timeout. It is often reasonable for whether or not a Host Application sends Rebirths to be a configuration option as this can have an impact on the overall Sparkplug system.
- A Sparkplug Host Application MAY send Node Control/Rebirth NCMD messages if malformed payloads arrive. Because this can have an impact on the overall system this should be configurable by the Host Application.
- There are other reasons a Host Application may send out Node Control/Rebirth NCMD messages. These include but are not limited to:
  - Receiving any DBIRTH, NDATA, DDATA, or DDEATH before receiving an NBIRTH from a Sparkplug Edge Node
  - Receiving a metric in an NDATA message that was not included in the previous NBIRTH message

- Receiving a metric in a DDATA message that was not included in the previous DBIRTH message
- Receiving an alias value that was not included in the corresponding NBIRTH or DBIRTH

5.16. 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:

- **[tck-id-operational-behavior-data-publish-nbirth] 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.**
- **[tck-id-operational-behavior-data-publish-nbirth-values] For each metric in the NBIRTH, the value MUST be set to the current value or if the current value is null, the is_null flag MUST be set to true and MUST NOT have a value specified.**
- **[tck-id-operational-behavior-data-publish-nbirth-change] NDATA messages SHOULD only be published when Edge Node level metrics change.**
  - In other words, metric values that have not changed within the same Sparkplug Session SHOULD 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.
- **[tck-id-operational-behavior-data-publish-nbirth-order] 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:

- **[tck-id-operational-behavior-data-publish-dbirth] DBIRTH messages MUST include all metrics for the specified Device that will ever be published for that Device within the established Sparkplug session.**
- **[tck-id-operational-behavior-data-publish-dbirth-values] For each metric in the DBIRTH, the value MUST be set to the current value or if the current value is null, the is_null flag MUST be set to true and MUST NOT have a value specified.**

- **[tck-id-operational-behavior-data-publish-dbirth-change] DDATA messages SHOULD only be published when Device level metrics change.**
  - In other words, metric values that have not changed within the same Sparkplug Session SHOULD not be resent until a new Sparkplug session is established.
- DDATA 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.
- **[tck-id-operational-behavior-data-publish-dbirth-order] 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.**

5.17. 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 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 achieved based on the use case. 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.

- **[tck-id-operational-behavior-data-commands-rebirth-name] An NBIRTH message MUST include a metric with a name of 'Node Control/Rebirth'.**
- **[tck-id-operational-behavior-data-commands-rebirth-name-aliases] When aliases are being used by an Edge Node an NBIRTH message MUST NOT include an alias for the 'Node Control/Rebirth' metric.**
  - This is to ensure that any Host Application connecting to the MQTT Server is capable of requesting a rebirth without knowledge of any potential alias being used for this metric.
- **[tck-id-operational-behavior-data-commands-rebirth-datatype] The 'Node Control/Rebirth' metric in the NBIRTH message MUST have a datatype of 'Boolean'.**
- **[tck-id-operational-behavior-data-commands-rebirth-value] 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.

- **[tck-id-operational-behavior-data-commands-ncmd-rebirth-verb] A Rebirth Request MUST use the NCMD Sparkplug verb.**
- **[tck-id-operational-behavior-data-commands-ncmd-rebirth-name] A Rebirth Request MUST include a metric with a name of 'Node Control/Rebirth'.**
- **[tck-id-operational-behavior-data-commands-ncmd-rebirth-value] A Rebirth Request MUST include a metric value of true.**

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

- **[tck-id-operational-behavior-data-commands-rebirth-action-1] When an Edge Node receives a Rebirth Request, it MUST immediately stop sending DATA messages.**
- **[tck-id-operational-behavior-data-commands-rebirth-action-2] After an Edge Node stops sending DATA messages, it MUST send a complete BIRTH sequence including the NBIRTH and DBIRTH(s) if applicable.**
- **[tck-id-operational-behavior-data-commands-rebirth-action-3] 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.

- **[tck-id-operational-behavior-data-commands-ncmd-verb] An Edge Node level command MUST use the NCMD Sparkplug verb.**
- **[tck-id-operational-behavior-data-commands-ncmd-metric-name] An NCMD message SHOULD include a metric name that was included in the associated NBIRTH message for the Edge Node.**
  - Sparkplug Edge Node Applications should be resilient to receiving metrics names that were not included in the NBIRTH message.
- **[tck-id-operational-behavior-data-commands-ncmd-metric-value] 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.

- **[tck-id-operational-behavior-data-commands-dcmd-verb] A Device level command MUST use the DCMD Sparkplug verb.**
- **[tck-id-operational-behavior-data-commands-dcmd-metric-name] A DCMD message SHOULD include a metric name that was included in the associated DBIRTH message for the Device.**
  - Sparkplug Edge Node Applications should be resilient to receiving metrics names that were not included in the DBIRTH message.
- **[tck-id-operational-behavior-data-commands-dcmd-metric-value] 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.

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 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 the 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

The Sparkplug B payload 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 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;          // MUST be a reference to
a template definition if this is an instance (i.e. the name of the template
definition) - MUST be omitted for template definitions
        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:

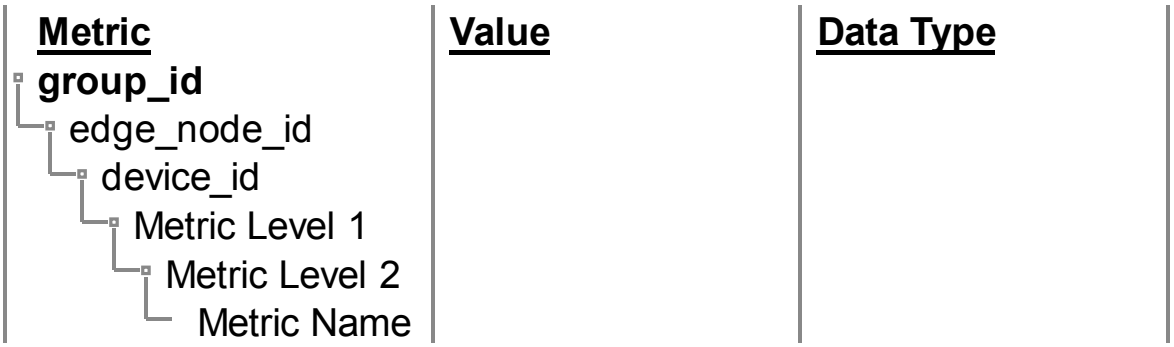

Figure 9 – Payload Metric Folder Structure

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).
    - **[tck-id-payloads-timestamp-in-UTC] 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.

  - **[tck-id-payloads-sequence-num-always-included] A sequence number MUST be included in the payload of every Sparkplug MQTT message from an Edge Node except NDEATH messages.**
  - **[tck-id-payloads-sequence-num-zero-nbirth] A NBIRTH message from an Edge Node MUST always contain a sequence number between 0 and 255 (inclusive).**
  - **[tck-id-payloads-sequence-num-incrementing] All subsequent messages after an NBIRTH from an Edge Node MUST contain a sequence number that is continually increasing by one in each message from that Edge Node 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.
  - **[tck-id-payloads-name-requirement] 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.**
    - **[tck-id-payloads-alias-uniqueness] 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.
    - **[tck-id-payloads-alias-birth-requirement] NBIRTH and DBIRTH messages MUST include both a metric name and alias.**
    - **[tck-id-payloads-alias-data-cmd-requirement] 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).
  - **[tck-id-payloads-name-birth-data-requirement] The timestamp MUST be included with every metric in all NBIRTH, DBIRTH, NDATA, and DDATA messages.**
  - **[tck-id-payloads-name-cmd-requirement] The timestamp MAY be included with metrics in NCMD and DCMD messages.**
  - **[tck-id-payloads-metric-timestamp-in-UTC] The timestamp MUST be in UTC.**
    - Non-normative comment: This timestamp represents the time at which the value of a metric was captured.
- **datatype**
  - **[tck-id-payloads-metric-datatype-value-type] The datatype MUST be an unsigned 32-bit integer representing the datatype.**
  - **[tck-id-payloads-metric-datatype-value] The datatype MUST be one of the enumerated values as shown in the valid Sparkplug Data Types.**
  - **[tck-id-payloads-metric-datatype-req] The datatype MUST be included with each metric definition in NBIRTH and DBIRTH messages.**
  - **[tck-id-payloads-metric-datatype-not-req] The datatype 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. More information on the Google Protocol Buffer types can be found here: https://developers.google.com/protocol-buffers/docs/proto#scalar
    - Google Protocol Buffer Type: *uint32*
    - Google Protocol Buffer Type: *uint64*
    - Google Protocol Buffer Type: *float*
    - Google Protocol Buffer Type: *double*
    - Google Protocol Buffer Type: *bool*
    - Google Protocol Buffer Type: *string*
    - Google Protocol Buffer Type: *bytes*
    - Sparkplug *DataSet*
      - Defined here.
    - Sparkplug *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.
  - **[tck-id-payloads-propertyset-keys-array-size] 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.
  - **[tck-id-payloads-propertyset-values-array-size] 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**
  - **[tck-id-payloads-metric-propertyvalue-type-type] This MUST be an unsigned 32-bit integer representing the datatype.**
  - **[tck-id-payloads-metric-propertyvalue-type-value] This value MUST be one of the enumerated values as shown in the Sparkplug Basic Data Types or the Sparkplug Property Value Data Types.**
  - **[tck-id-payloads-metric-propertyvalue-type-req] 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. More information on the Google Protocol Buffer types can be found here: https://developers.google.com/protocol-buffers/docs/proto#scalar
    - Google Protocol Buffer Type: *uint32*
    - Google Protocol Buffer Type: *uint64*
    - Google Protocol Buffer Type: *float*
    - Google Protocol Buffer Type: *double*

- Google Protocol Buffer Type: *bool*
- Google Protocol Buffer Type: *string*
- Sparkplug *PropertySet*
  - Defined here.
- Sparkplug *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

[tck-id-payloads-propertyset-quality-value-type] The 'type' of the Property Value MUST be a value of 3 which represents a Signed 32-bit Integer.

[tck-id-payloads-propertyset-quality-value-value] 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 PropertySet 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**
  - **[tck-id-payloads-dataset-column-size] 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.
  - **[tck-id-payloads-dataset-column-num-headers] The size of the array MUST have the same number of elements that the types array contains.**
- **types**
  - **[tck-id-payloads-dataset-types-def] This MUST be an array of unsigned 32 bit integers representing the datatypes of the columns.**
  - **[tck-id-payloads-dataset-types-num] The array of types MUST have the same number of elements that the columns array contains.**
  - **[tck-id-payloads-dataset-types-type] The values in the types array MUST be a unsigned 32-bit integer representing the datatype.**
  - **[tck-id-payloads-dataset-types-value] This values in the types array MUST be one of the enumerated values as shown in the Sparkplug Basic Data Types.**
  - **[tck-id-payloads-dataset-parameter-type-req] The types array MUST be included in all DataSets.**
- **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.
  - **[tck-id-payloads-template-dataset-value] The value supplied MUST be one of the following Google Protobuf types: *uint32*, *uint64*, *float*, *double*, *bool*, or *string*.**

More information on the types above can be found 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.
    - **[tck-id-payloads-template-definition-nbirth-only] Template Definitions MUST only be included in NBIRTH messages.**
    - **[tck-id-payloads-template-definition-is-definition] A Template Definition MUST have is_definition set to true.**
    - **[tck-id-payloads-template-definition-ref] A Template Definition MUST omit the template_ref field.**
    - **[tck-id-payloads-template-definition-members] A Template Definition MUST include all member metrics that will ever be included in corresponding template instances.**
    - **[tck-id-payloads-template-definition-nbirth] A Template Definition MUST be included in the NBIRTH for all Template Instances that are included in the NBIRTH and DBIRTH messages.**
      - A Template Instance can not reference a Template Definition that was not included in the NBIRTH.
    - **[tck-id-payloads-template-definition-parameters] A Template Definition MUST include all parameters that will be included in the corresponding Template Instances.**
    - **[tck-id-payloads-template-definition-parameters-default] A Template Definition MAY include values for parameters in the Template Definition parameters.**
      - These act as the defaults for any template instances that don't include parameter values in the NBIRTH or DBIRTH messages.
- **Template Instance**
  - This is an instance of a Sparkplug Template.
    - **[tck-id-payloads-template-instance-is-definition] A Template Instance MUST have is_definition set to false.**
    - **[tck-id-payloads-template-instance-ref] A Template Instance MUST have template_ref set to the type of template definition it is.**
      - It must be set to the name of the metric that represents the template definition.

- **[tck-id-payloads-template-instance-members] A Template Instance MUST include only members that were included in the corresponding template definition.**
- **[tck-id-payloads-template-instance-members-birth] A Template Instance in a NBIRTH or DBIRTH message MUST include all members that were included in the corresponding Template Definition.**
- **[tck-id-payloads-template-instance-members-data] A Template Instance in a NDATA or DDATA message MAY include only a subset of the members that were included in the corresponding template definition.**
  - A Template Instance does not need to be a complete set of all member metrics that were included in the Template Definition.
- **[tck-id-payloads-template-instance-parameters] A Template Instance MAY include parameter values for any parameters that were included in the corresponding Template Definition.**
  - If a parameter value was included in the Template Definition but not in the Template Instance the value of that parameter is implicitly the default value from 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.
  - **[tck-id-payloads-template-version] 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**
  - **[tck-id-payloads-template-ref-definition] This MUST be omitted if this is a Template Definition.**
  - **[tck-id-payloads-template-ref-instance] 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.
- **[tck-id-payloads-template-is-definition] This MUST be included in every Template Definition and Template Instance.**
- **[tck-id-payloads-template-is-definition-definition] This MUST be set to true if this is a Template Definition.**
- **[tck-id-payloads-template-is-definition-instance] 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**
  - **[tck-id-payloads-template-parameter-name-required] This MUST be included in every Template Parameter definition.**
  - **[tck-id-payloads-template-parameter-name-type] This MUST be a UTF-8 string representing the name of the Template parameter.**
- **type**
  - **[tck-id-payloads-template-parameter-value-type] This MUST be an unsigned 32-bit integer representing the datatype.**
  - **[tck-id-payloads-template-parameter-type-value] This value MUST be one of the enumerated values as shown in the Sparkplug Basic Data Types.**
  - **[tck-id-payloads-template-parameter-type-req] 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.
  - **[tck-id-payloads-template-parameter-value] The value supplied MUST be one of the following Google Protocol Buffer types: *uint32*, *uint64*, *float*, *double*, *bool*, or *string*.**
    - More information on these types can be found here: https://developers.google.com/protocol-buffers/docs/proto#scalar
  - For a template definition, this is the default value of the parameter. For a template instance, this is the value unique to that instance.

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**

All array types use the bytes_value field of the Metric value field. They are simply little-endian packed byte arrays.

For example, consider an Int32 array with two decimal values [123456789, 987654321]

Array converted to little endian hex: [0x15CD5B07, 0xB168DE3A]

The bytes_value of the Sparkplug Metric must be: [0x15, 0xCD, 0x5B, 0x07, 0xB1, 0x68, 0xDE, 0x3A]

- *Int8Array*
    - Int8Array as an array of packed little endian int8 bytes
    - Google Protocol Buffer Type: bytes
    - Sparkplug enum value: 22
    - Example (Decimal to Metric bytes_value): [-23, 123] → [0xEF, 0x7B]
- *Int16Array*
    - Int16Array as an array of packed little endian int16 bytes
    - Google Protocol Buffer Type: bytes
    - Sparkplug enum value: 23
    - Example (Decimal to Metric bytes_value): [-30000, 30000] → [0xD0, 0x8A, 0x30, 0x75]
- *Int32Array*
    - Int8Array as an array of packed little endian int32 bytes
    - Google Protocol Buffer Type: bytes
    - Sparkplug enum value: 24
    - Example (Decimal to Metric bytes_value): [-1, 315338746] → [0xFF, 0xFF, 0xFF, 0xFF, 0xFA, 0xAF, 0xCB, 0x12]
- *Int64Array*
    - Int8Array as an array of packed little endian int64 bytes
    - Google Protocol Buffer Type: bytes
    - Sparkplug enum value: 25
    - Example (Decimal to Metric bytes_value): [-4270929666821191986, -3601064768563266876] → [0xCE, 0x06, 0x72, 0xAC, 0x18, 0x9C, 0xBA, 0xC4, 0xC4, 0xBA, 0x9C, 0x18, 0xAC, 0x72, 0x06, 0xCE]

- *UInt8Array*
  - UInt8Array as an array of packed little endian uint8 bytes
  - Google Protocol Buffer Type: bytes
  - Sparkplug enum value: 26
  - Example (Decimal to Metric bytes_value): [23, 250] → [0x17, 0xFA]
- *UInt16Array*
  - UInt16Array as an array of packed little endian uint16 bytes
  - Google Protocol Buffer Type: bytes
  - Sparkplug enum value: 27
  - Example (Decimal to Metric bytes_value): [30, 52360] → [0x1E, 0x00, 0x88, 0xCC]
- *UInt32Array*
  - UInt32Array as an array of packed little endian uint32 bytes
  - Google Protocol Buffer Type: bytes
  - Sparkplug enum value: 28
  - Example (Decimal to Metric bytes_value): [52, 3293969225] → [0x34, 0x00, 0x00, 0x00, 0x49, 0xFB, 0x55, 0xC4]
- *UInt64Array*
  - UInt64Array as an array of packed little endian uint64 bytes
  - Google Protocol Buffer Type: bytes
  - Sparkplug enum value: 29
  - Example (Decimal to Metric bytes_value): [52, 16444743074749521625] → [0x34, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0xD9, 0x9E, 0x02, 0xD1, 0xB2, 0x76, 0x37, 0xE4]
- *FloatArray*
  - FloatArray as an array of packed little endian 32-bit float bytes
  - Google Protocol Buffer Type: bytes
  - Sparkplug enum value: 30
  - Example (Decimal to Metric bytes_value): [1.23, 89.341] → [0x3F, 0x9D, 0x70, 0xA4, 0x42, 0xB2, 0xAE, 0x98]
- *DoubleArray*
  - DoubleArray as an array of packed little endian 64-bit float bytes

  - Google Protocol Buffer Type: bytes
  - Sparkplug enum value: 31
  - Example (Decimal to Metric bytes_value): [12.354213, 1022.9123213] → [0x40, 0x28, 0xB5, 0x5B, 0x68, 0x05, 0xA2, 0xD7, 0x40, 0x8F, 0xF7, 0x4C, 0x6F, 0x1C, 0x17, 0x8E]

- *BooleanArray*
  - BooleanArray as an array of bit-packed bytes preceeded by a 4-byte integer that represents the total number of boolean values
  - Google Protocol Buffer Type: bytes
  - Sparkplug enum value: 32
  - Example (boolean array to Metric bytes_value): [false, false, true, true, false, true, false, false, true, true, false, true] → [0x0C, 0x00, 0x00, 0x00, 0x34, 0xDX]
    - Note an 'X' above is a 'do not care'. It can be either 1 or 0 but must be present so the array ends on a byte boundary.

- *StringArray*
  - StringArray as an array of null terminated strings
  - Google Protocol Buffer Type: bytes
  - Sparkplug enum value: 33
  - Example (string array to Metric bytes_value): [ABC, hello] → [0x41, 0x42, 0x43, 0x00, 0x68, 0x65, 0x6c, 0x6c, 0x6f, 0x00]

- *DateTimeArray*
  - DateTimeArray as an array of packed little endian bytes where each Datetime value is an 8-byte value representing the number of milliseconds since epoch in UTC
  - Google Protocol Buffer Type: bytes
  - Sparkplug enum value: 34
  - Example (DateTime array → ms since epoch → Metric bytes_value): ['Wednesday, October 21, 2009 5:27:55.335 AM', 'Friday, June 24, 2022 9:57:55 PM'] → [1256102875335, 1656107875000] → [0xC7, 0xD0, 0x90, 0x75, 0x24, 0x01, 0xB8, 0xBA, 0xB8, 0x97, 0x81, 0x01]

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.

There is a dependency on the MQTT CONNECT packet with regard to NBIRTH messages that are subsequently sent for that given MQTT Session. These are can be found in the Edge Node Session Establishment Section

- **[tck-id-payloads-nbirth-timestamp] NBIRTH messages MUST include a payload timestamp that denotes the time at which the message was published.**
- **[tck-id-payloads-nbirth-edge-node-descriptor] 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.
- **[tck-id-payloads-nbirth-seq] Every NBIRTH message MUST include a sequence number and it MUST have a value between 0 and 255 (inclusive).**
- **[tck-id-payloads-nbirth-bdseq] Every NBIRTH message MUST include a bdSeq number metric.**
- **[tck-id-payloads-nbirth-bdseq-repeat] The bdSeq number value MUST match the bdSeq number value that was sent in the prior MQTT CONNECT packet WILL Message.**
  - Note if a new NBIRTH is being sent (due to a Rebirth request or ant other reason) the Sparkplug Edge Node MQTT client MUST publish the same bdSeq number that was sent in the prior MQTT CONNECT packet Will Message payload. This is because the NDEATH bdSeq number MUST always match the bdSeq number of the associated NBIRTH that is stored in the MQTT Server via the MQTT Will Message.
- **[tck-id-payloads-nbirth-rebirth-req] 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.
- **[tck-id-payloads-nbirth-qos] NBIRTH messages MUST be published with the MQTT QoS set to 0.**
- **[tck-id-payloads-nbirth-retain] 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": "Int64",
                "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.

Metric		Value	Data Type
Sparkplug B Devices	/group_id		
Raspberry Pi	/edge_node_id		
Node Control	Node Control		
Reboot		FALSE	Boolean
Rebirth		FALSE	Boolean
Next Server		FALSE	Boolean
Scan Rate		3000	Int64
Properties	Node Properties		
Hardware Make		Raspberry Pi	String
Mardware Model		Pi 3 Model B	String
OS Name		Raspbian	String
OS Version		Jessie with PIXEL/11.01.2017	String
Supply Voltage		12.1	Float

Figure 10 – 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.

- **[tck-id-payloads-dbirth-timestamp] DBIRTH messages MUST include a payload timestamp that denotes the time at which the message was published.**
- **[tck-id-payloads-dbirth-seq] Every DBIRTH message MUST include a sequence number.**
- **[tck-id-payloads-dbirth-seq-inc] 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.**
- **[tck-id-payloads-dbirth-order] All DBIRTH messages sent by an Edge Node MUST be sent immediately after the NBIRTH and before any NDATA or DDATA messages are published by the Edge Node.**
- **[tck-id-payloads-dbirth-qos] DBIRTH messages MUST be published with the MQTT QoS set to 0.**
- **[tck-id-payloads-dbirth-retain] 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 a 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.

Metric		Value	Data Type
Sparkplug B Devices	/group_id		
Raspberry Pi	/edge_node_id		
Pibrella	/device_id		
Inputs			
A		FALSE	Boolean
B		FALSE	Boolean
C		FALSE	Boolean
D		FALSE	Boolean
Outputs			
LEDs			
Green		FALSE	Boolean
Red		FALSE	Boolean
Yellow		FALSE	Boolean
E		FALSE	Boolean
F		FALSE	Boolean
G		FALSE	Boolean
H		FALSE	Boolean
Buzzer		FALSE	Boolean
Properties			
Hardware Make		Pibrella	String

Everything under the Pibrella node is Device Process Variables and Metric Tags

Figure 11 – 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.

- **[tck-id-payloads-ndata-timestamp] NDATA messages MUST include a payload timestamp that denotes the time at which the message was published.**
- **[tck-id-payloads-ndata-seq] Every NDATA message MUST include a sequence number.**
- **[tck-id-payloads-ndata-seq-inc] 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.**

- **[tck-id-payloads-ndata-order] All NDATA messages sent by an Edge Node MUST NOT be sent until all the NBIRTH and all DBIRTH messages have been published by the Edge Node.**
- **[tck-id-payloads-ndata-qos] NDATA messages MUST be published with the MQTT QoS set to 0.**
- **[tck-id-payloads-ndata-retain] 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.

- **[tck-id-payloads-ddata-timestamp] DDATA messages MUST include a payload timestamp that denotes the time at which the message was published.**
- **[tck-id-payloads-ddata-seq] Every DDATA message MUST include a sequence number.**
- **[tck-id-payloads-ddata-seq-inc] 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.

- **[tck-id-payloads-ddata-order] All DDATA messages sent by an Edge Node MUST NOT be sent until all the NBIRTH and all DBIRTH messages have been published by the Edge Node.**
- **[tck-id-payloads-ddata-qos] DDATA messages MUST be published with the MQTT QoS set to 0.**
- **[tck-id-payloads-ddata-retain] 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.

- **[tck-id-payloads-ncmd-timestamp] NCMD messages MUST include a payload timestamp that denotes the time at which the message was published.**
- **[tck-id-payloads-ncmd-seq] Every NCMD message MUST NOT include a sequence number.**

- **[tck-id-payloads-ncmd-qos] NCMD messages MUST be published with the MQTT QoS set to 0.**
- **[tck-id-payloads-ncmd-retain] 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.

- **[tck-id-payloads-dcmd-timestamp] DCMD messages MUST include a payload timestamp that denotes the time at which the message was published.**
- **[tck-id-payloads-dcmd-seq] Every DCMD message MUST NOT include a sequence number.**
- **[tck-id-payloads-dcmd-qos] DCMD messages MUST be published with the MQTT QoS set to 0.**
- **[tck-id-payloads-dcmd-retain] 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.

- **[tck-id-payloads-ndeath-seq] Every NDEATH message MUST NOT include a sequence number.**
- **[tck-id-payloads-ndeath-will-message] An NDEATH message MUST be registered as a Will Message in the MQTT CONNECT packet.**
- **[tck-id-payloads-ndeath-will-message-qos] The NDEATH message MUST set the MQTT Will QoS to 1 in the MQTT CONNECT packet.**
- **[tck-id-payloads-ndeath-will-message-retain] The NDEATH message MUST set the MQTT Will Retained flag to false in the MQTT CONNECT packet.**
- **[tck-id-payloads-ndeath-bdseq] 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.
  - It is important to note that any new CONNECT packet must increment the bdSeq number in the payload compared to what was in the previous CONNECT packet. This ensures that any Host Applications will be able to distinguish between current and old bdSeq numbers in the event that messages are delivered out of order. When incrementing the bdSeq number, if the previous value was 255, the next must be zero.

- **[tck-id-payloads-ndeath-will-message-publisher] 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.
- **[tck-id-payloads-ndeath-will-message-publisher-disconnect-mqtt311] If the Edge Node is using MQTT 3.1.1 and it sends an MQTT DISCONNECT packet, the Edge Node MUST publish an NDEATH message to the MQTT Server before it sends the MQTT DISCONNECT packet.**
  - This is to ensure Host Applications are notified that the Edge Node is disconnecting. Because an MQTT DISCONNECT packet is sent, the MQTT Server will not deliver the Will Message/NDEATH on behalf of the disconnecting Edge Node.
- **[tck-id-payloads-ndeath-will-message-publisher-disconnect-mqtt50] If the Edge Node is using MQTT 5.0 and it sends an MQTT DISCONNECT packet, the MQTT v5.0 'Disconnect with Will Message' reason code MUST be set in the DISCONNECT packet.**
  - This is to ensure Host Applications are notified that the Edge Node is disconnecting by the MQTT Server.
- 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.

- **[tck-id-payloads-ddeath-timestamp] DDEATH messages MUST include a payload timestamp that denotes the time at which the message was published.**
- **[tck-id-payloads-ddeath-seq] Every DDEATH message MUST include a sequence number.**
- **[tck-id-payloads-ddeath-seq-inc] Every DDEATH 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 a DDEATH message based on the 'DDEATH' Sparkplug Verb

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

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

[tck-id-payloads-ddeath-seq-number] 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.

- **[tck-id-payloads-state-will-message] Sparkplug Host Applications MUST register a Will Message in the MQTT CONNECT packet on the topic 'spBv1.0/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.
- **[tck-id-payloads-state-will-message-qos] The Sparkplug Host Application MUST set the the MQTT Will QoS to 1 in the MQTT CONNECT packet.**
- **[tck-id-payloads-state-will-message-retain] The Sparkplug Host Application MUST set the Will Retained flag to true in the MQTT CONNECT packet.**
- **[tck-id-payloads-state-will-message-payload] The Death Certificate Payload MUST be JSON UTF-8 data. It MUST include two key/value pairs where one key MUST be 'online' and it's value is a boolean 'false'. The other key MUST be 'timestamp' and the value MUST be a numeric value representing the current UTC time in milliseconds since Epoch.**
- **[tck-id-payloads-state-subscribe] After establishing an MQTT connection, the Sparkplug Host Application MUST subscribe on it's own 'spBv1.0/STATE/[sparkplug_host_id]' topic.**
  - The [sparkplug_host_id] should be replaced with the Sparkplug Host Application's ID. This can be any UTF-8 string.
  - 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.
- **[tck-id-payloads-state-birth] After subscribing on it's own spBv1.0/STATE/[sparkplug_host_id] topic, the Sparkplug Host Application MUST publish an MQTT message on the topic 'spBv1.0/STATE/[sparkplug_host_id]' with 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.
- **[tck-id-payloads-state-birth-payload] The Birth Certificate Payload MUST be JSON UTF-8 data. It MUST include two key/value pairs where one key MUST be 'online' and it's value is a boolean 'true'. The other key MUST be 'timestamp' and the value MUST match the timestamp value that was used in the immediately prior MQTT CONNECT packet Will Message payload.**

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 Security Section [MQTTV3.1.1-5] should be considered when implementing a Sparkplug solution. If using MQTT v5, please refer to this section [MQTTV5-5] instead.

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 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: spBv1.0/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: spBv1.0/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: spBv1.0/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.

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).

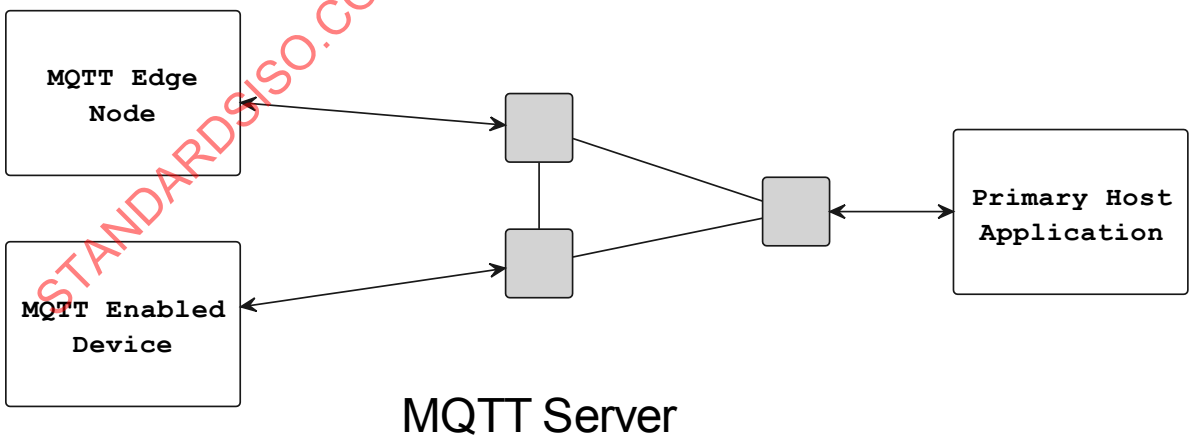

Figure 12 – High Availability MQTT Server Cluster

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.

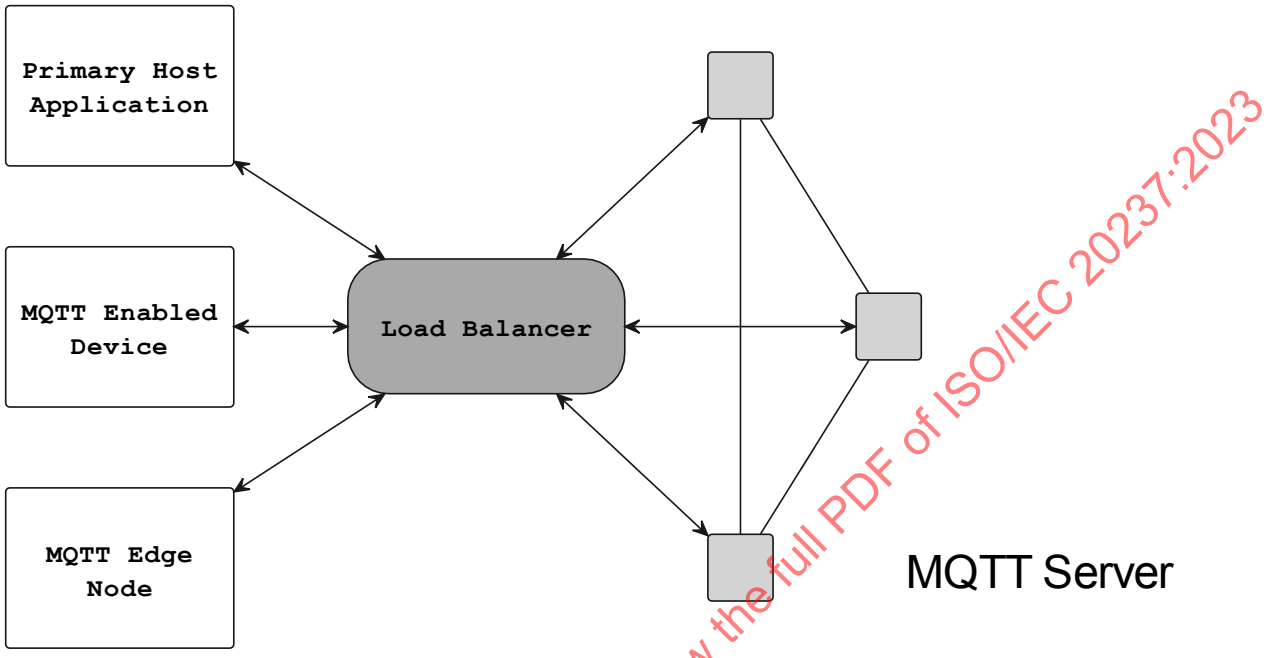

Figure 13 – High Availability MQTT Server Cluster with Load Balancer

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 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.

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.

Edge Nodes in an infrastructure that provides multiple MQTT Servers can establish a session to any one of the MQTT Servers. 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 JOSN payload of this message to ensure that it is a value of “online=true”. If the value is “online=false”, this indicates the Primary Application has lost its MQTT Session to this particular MQTT Server. This should cause the Edge 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 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:

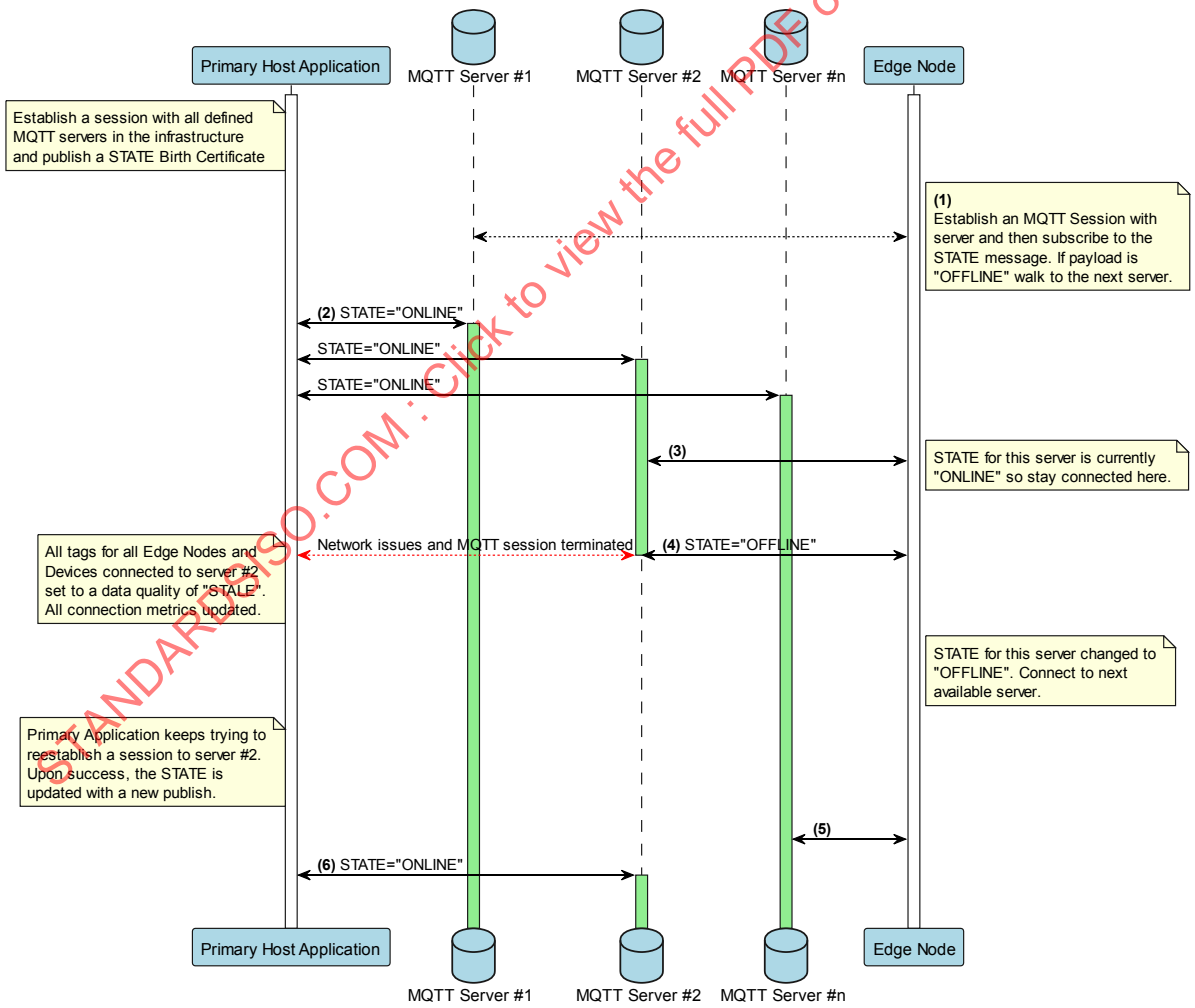

Figure 14 – Primary Application STATE flow diagram

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 online value. If the STATE message payload contains ‘online=false’ and the bdSeq number value in the paylaod matches the bdSeq number in the prior Host Application BIRTH message then the Edge 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 ‘online=false’ with the message RETAIN flag set to true. Then the Primary Host Application BIRTH Certificate will be published with a STATE payload of ‘online=true’.

3. As the Edge Node walks its available MQTT Server table, it will establish an MQTT Session with a server that has a STATE message with a JSON payload that contains ‘online=true’. 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 table.

5. The Edge Node moved to the next available MQTT Server and since the current STATE on this server is ‘online=true’, 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 with 'online=true’. Now MQTT Server #2 is ready to accept new Edge Node session requests.

9. Acknowledgements

The specification would not exist without the initial contribution of the Sparkplug specification by Cirrus Link Solutions, Wes Johnson, Chad Kienle, and Arlen Nipper. They have also continued to be involved in promoting, developing, and contributing to the Sparkplug community.

The following individuals are members of the Eclipse Sparkplug Working Group, the Eclipse Sparkplug Specification project, the Eclipse Tahu project, or otherwise contributed in a meaningful way to the Sparkplug Specification.

- Lukas Brand (HiveMQ)
- Ilya Binshtok (Cirrus Link Solutions)
- Justin Brzozoski (SignalFire)
- Travis Cox (Inductive Automation)
- Ian Craggs (individual)

- Nathan Davenport (Cirrus Link Solutions)
- Frédéric Desbiens (Eclipse Foundation)
- Alex Godbehere (AMRC)
- Anja Helmbrecht-Schaar (HiveMQ)
- Benson Hougland (Opto 22)
- Wes Johnson (Cirrus Link Solutions)
- Chad Kienle (Cirrus Link Solutions)
- Mitchell McPartland (Inductive Automation)
- Bryce Nakatani (Opto 22)
- Arlen Nipper (Cirrus Link Solutions)
- Dominik Obermaier (HiveMQ)
- Alexander Schwartz (individual)
- Josh Wolf (Canary Labs)

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