CLASS D MESSAGING SPECIFICATION

Standard
S-9356

Adopted: 2010

TABLE OF CONTENTS

1.0 INTRODUCTION

1.1 Purpose

The Class D protocol is intended for applications requiring reliable point-to-point message delivery. The Class D protocol is divided into five layers. The first of these layers, the Protocol Layer, is mandatory for all Class D protocol implementations. There are four optional layers—the High Availability Layer, the Persistence Layer, the Management Layer, and the Built-in Test Layer.

1.2 Document Conventions

Throughout the requirements given in this document, references to configurable link or node attributes are stated using the attribute name in italics, such as *data ACK time-out*.

1.3 Intended Audience and Reading Suggestions

This document is intended for software developers and testers implementing the Class D protocol. The material is divided into sections corresponding to the layers of the protocol. The layers provided by a given implementation will vary. Readers should read the entire protocol overview section, the section covering the mandatory Protocol Layer, and then each section covering the optional layers to be developed and tested.

1.4 Protocol Scope

Within the ITC messaging system, the Class D protocol is intended for use where vendor and railroad interoperability is required. At the time of this writing, Class D is planned for use between applications and the ITC Application Gateways to be found in railroad back offices, locomotives, and waysides.

1.5 References

- IETF RFC 1122 Requirements for Internet Hosts—Communication Layers
- IETF RFC 5246 The Transport Layer Security (TLS) Protocol Version 1.2

2.0 OVERALL DESCRIPTION

The Class D transport protocol is intended for applications requiring reliable point-to-point message delivery. To meet this goal, the protocol is built upon the reliable, ordered delivery of the TCP/IP protocol. It adds upon TCP/IP support for such features as message acknowledgement, message persistence, link loss detection, and link fail-over.

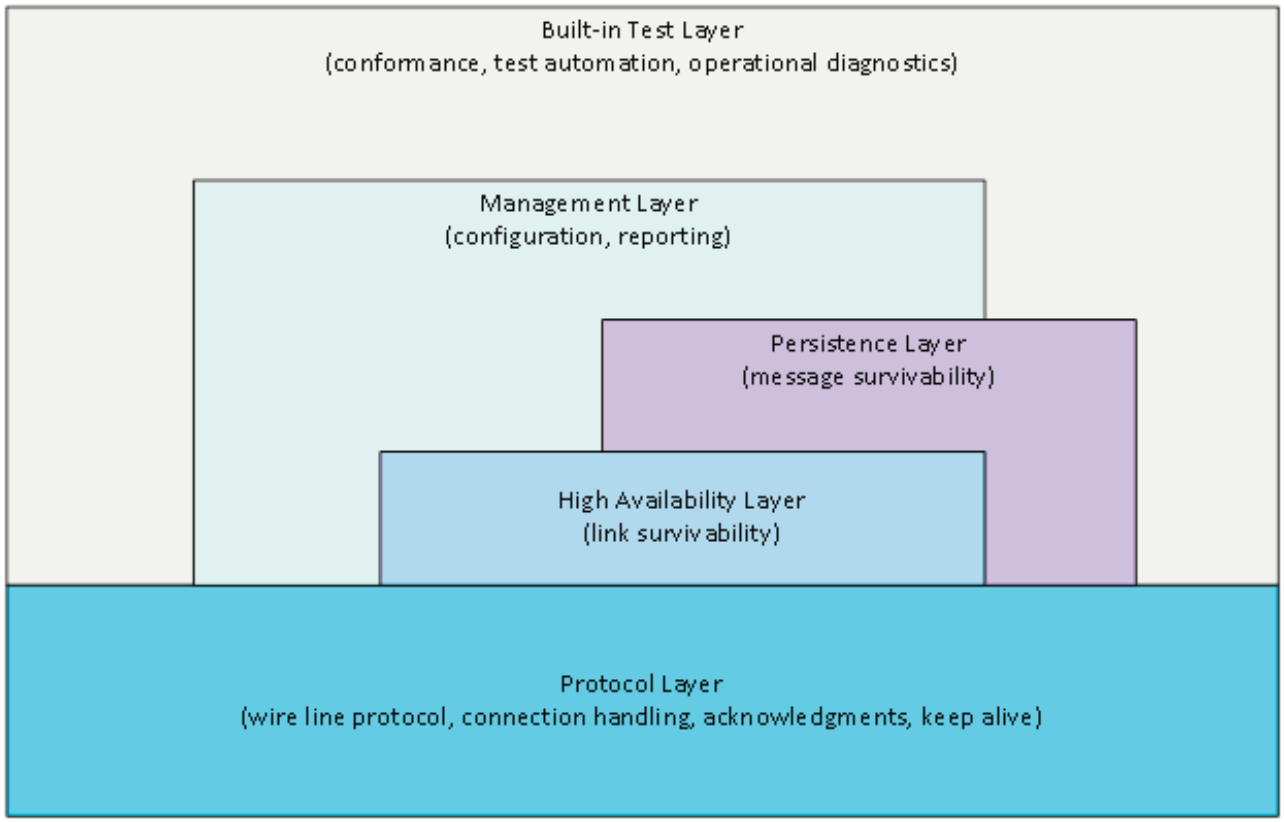

Fig. 2.1 Class D protocol layering model

At its core, the Class D protocol defines a required Protocol Layer. This layer defines the low-level protocol structure and expected behaviors concerning establishment of connections, keep-alive messages, and error handling. The Protocol Layer is mandatory for those devices and applications claiming Class D protocol compliance.

In addition to the Protocol Layer, the Class D protocol defines four optional layers. These are the High Availability Layer, the Persistence Layer, the Management Layer, and the Built-in Test Layer.

- The High Availability Layer describes support for survivability of Class D links in the event of link or node failure.
- The Persistence Layer describes support for survivability of Class D messages in the event of link or node failure.
- The Management Layer describes support for remotely configuring Class D links and optional protocol features, remotely querying statistics and state, and notifying external agents when error conditions occur.
- The Built-in Test Layer describes support for testing protocol conformance, test automation, and operational diagnostics.

A packaged implementation of the Class D protocol may include full support for all five layers of the protocol. Devices and applications needing a Class D protocol implementation may already provide some of the features described by the optional layers. For those cases, a packaged implementation may include support only for the mandatory Protocol Layer. This specification imposes no requirement on the set of optional layers that may be provided by an implementation of the Class D protocol. However, there are some expectations regarding the interaction between the optional layers. For example, an implementation that includes the High Availability Layer and the Management Layer must allow management of the High Availability Layer functions.

2.1 Class D Protocol Layer

The Class D Protocol Layer is the only mandatory layer defined by the Class D Messaging specification. Any device or application claiming compliance with the Class D protocol must at a minimum implement the Protocol Layer of Class D.

The Protocol Layer defines the low-level protocol structure and expected behaviors concerning establishment of connections, keep-alive messages, and error handling. It also defines configurable features such as message acknowledgment and transport layer security (TLS).

Note: Wherever the term *IP address* is used throughout this specification, it is assumed that a DNS-compatible name can also be used.

2.1.1 Link Initiation

Before communication can begin, two Class D nodes must identify each other and establish a link. The current specification defines only one link initiation profile—the configuration-based link initiation profile. With this profile, each node must be configured with the link attributes it needs to establish a connection to another node. Interdependencies between Class D nodes with respect to link attributes are defined by the respective link attribute requirements.

2.1.1.1 Link Security

Class D nodes may provide support for the Transport Layer Security (TLS) 1.2 protocol as defined by RFC 5246. The TLS protocol allows applications to communicate in a way that is designed to prevent eavesdropping, tampering, or message forgery. TLS support is optional; however, if TLS support is provided by an implementation, it must satisfy the requirements given in this specification.

2.1.2 Message Transport

Once a link is established, two nodes can exchange messages. This specification defines a set of message transport functions that a node must support. These functions include the following:

- Message transmission
- Message acknowledgement
- Error reporting

2.1.3 Link Management

As previously mentioned, the Class D protocol is designed to provide reliable point-to-point message delivery with measures to ensure link and message survivability. While some of this capability is allocated to the optional High Availability Layer and Persistence Layer, much of it is provided as part of the mandatory Protocol Layer. The mandatory functions include the following:

- Link monitoring, achieved through the use of keep-alive messages
- Link loss detection, achieved through the use of keep-alive messages
- Link reestablishment, achieved through reconnection logic
- Link termination

2.2 Class D High Availability Layer

The optional High Availability Layer provides support for survivability of Class D links. It describes the required behavior of a Class D node using alternate links to recover from link or node failure. The functions added by the High Availability Layer complement the required Protocol Layer, but they do not replace it. Class D implementations providing support for the High Availability Layer must still satisfy all requirements of the Protocol Layer.

2.3 Class D Persistence Layer

The optional Persistence Layer provides support for survivability of Class D messages in the event of link or node failure. It adds requirements for securing messages at a receiving node before discarding them on the transmitting node. The functions added by the Persistence Layer complement the required Protocol Layer, but they do not replace it. Class D implementations providing support for the Persistence Layer must still satisfy all requirements of the Protocol Layer.

2.4 Class D Management Layer

The optional Management Layer provides support for remotely and securely configuring Class D links and optional protocol features, remotely querying statistics and state, and notifying external agents when error conditions occur. A Class D node that implements the Management Layer must, at a minimum, provide the ability to manage the functions of the Protocol Layer. If other optional layers are supported by the Class D node, the Management Layer must provide the ability to manage the functions of those layers as well.

2.5 Class D Built-In Test Layer

The optional Built-in Test Layer provides support for testing protocol conformance, test automation, and operational diagnostics. A Class D node that implements the Built-in Test Layer should, at a minimum, provide the ability to test and diagnose the functions of the Protocol Layer. If other optional layers are provided with the Class D node, the Built-in Test Layer must provide the ability to test and diagnose the functions of those layers as well.

2.6 Link Types

There are three types of links for which a Class D link can be constructed: unidirectional send-only link, unidirectional receive-only link, and bidirectional link. The distinction is made based upon the direction that data messages flow across the link relative to the TCP client. The terms *keep alive* and *ACKs & NAKs* in the diagrams below refer to Class D protocol features and not TCP protocol features.

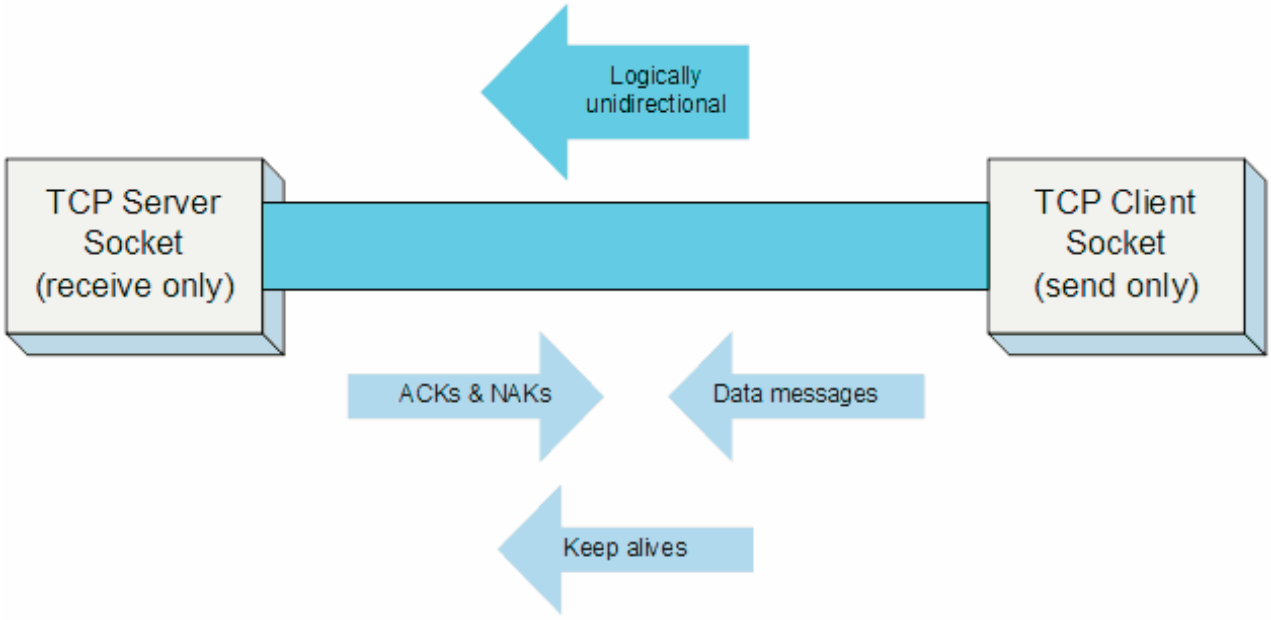

Fig. 2.2 Unidirectional send-only link

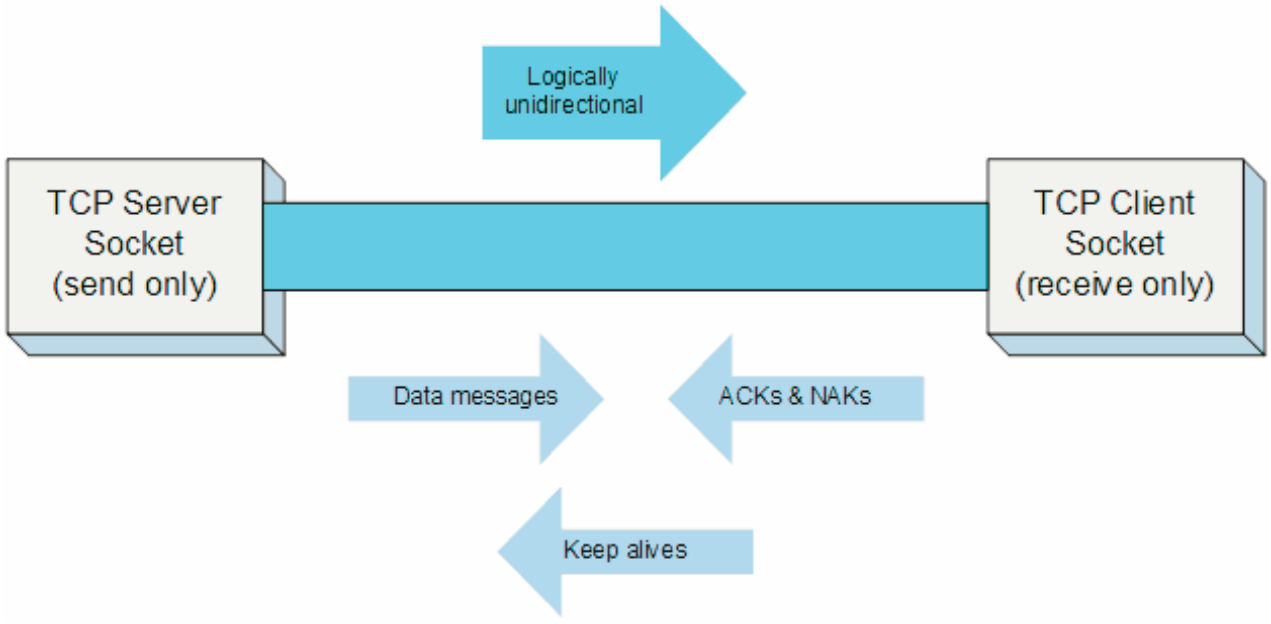

Fig. 2.3 Unidirectional receive-only link

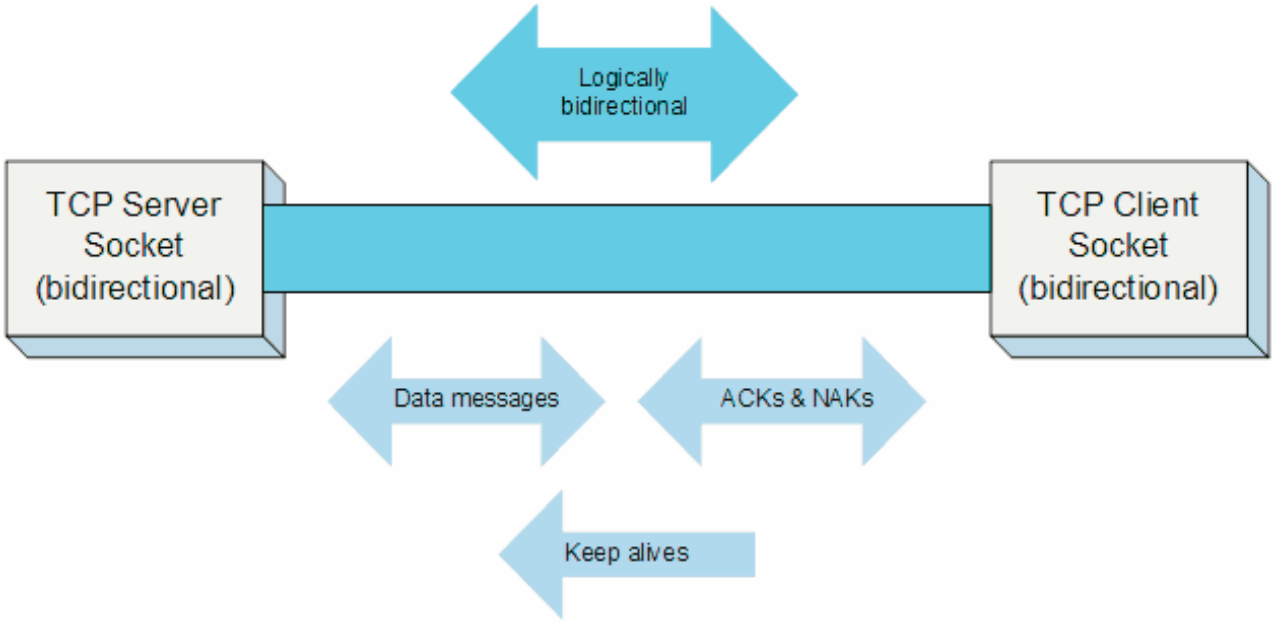

Fig. 2.4 Bidirectional link

2.7 Mapping of Protocol to OSI Layers

When relating the Class D protocol to the OSI model, it is essential to establish the context of the discussion. From the perspective of TCP/IP, which implements behaviors of the network, transport, and to some extent the session OSI layers, the Class D protocol is primarily an implementation of the presentation and application OSI layers. From the perspective of the ITC messaging system, which is how it is typically discussed, the Class D protocol implements the behaviors of the data link OSI layer. That is, it provides the reliability of a link between PTC applications and ITP Application Gateways.

2.8 Relationship to Other Protocols

The Class D protocol is built upon the reliable, ordered delivery of the TCP/IP protocol. It adds upon TCP/IP support for such features as message acknowledgement, message persistence, link loss detection, and link fail-over. Class D does not identify or restrict the protocol type of its payload; therefore, all other protocols can be carried over Class D as standard payload.

3.0 CLASS D PROTOCOL LAYER

This section defines the requirements for implementing the Protocol Layer of the Class D protocol.

The Class D Protocol Layer is the only mandatory layer defined by the Class D Messaging specification. Any device or application claiming compliance with the Class D protocol must, at a minimum, implement the Protocol Layer of Class D.

The Protocol Layer defines the low-level protocol structure and expected behaviors concerning establishment of connections, keep-alive messages, and error handling. It also defines configurable features such as message acknowledgment and transport layer security (TLS).

3.1 Link Attributes

Table 3.1 lists attribute names used to describe the configuration of a Class D link. It is strongly recommended that the same names be used when implementing the protocol. Subsequent sections of this document provide requirements pertaining to the use and meaning of each link attribute.

Table 3.1 Basic link attributes (page 1 of 2)

Attribute Name	Client		Server	
	Required	Values	Required	Values
TCP role	Yes	client	Yes	server
Mode	No	send only receive only bidirectional (default)	No	send only receive only bidirectional (default)
Local address	No	*IP address* *DNS name* any (default)	No	*IP address* *DNS name* all (default)
Remote address	Yes	*IPv4 address* *DNS name*	No	*IPv4 address* *DNS name* none (default)
Local port	No	*IP port* any (default)	Yes	*IP port*
Remote port	Yes	*IP port*	No	(not applicable)
Log traffic	No	no (default) yes	No	no (default) yes
Keep-alive interval	Yes	0 <= *x* <= 60000 milliseconds	Yes	0 <= *x* <= 60000 milliseconds
Keep-alive ACK timeout	Yes	0 <= *x* <= 60000 milliseconds	No	(not applicable)
Data ACK enabled	Yes	yes no	Yes	yes no
Data ACK timeout	Yes[a/]	1 < *x* <= 60000 milliseconds	Yes[a/]	1 < *x* <= 60000 milliseconds
Data NAK retry limit	Yes	0 <= *x* <= 10	Yes	0 <= *x* <= 10
Retransmit delay	No	0 <= *x* <= 10000 milliseconds 0 = none (default)	No	0 <= *x* <= 10000 milliseconds 0 = none (default)
Connection attempt timeout	Yes	0 < *x* <= 60000 milliseconds	No	(not applicable)
Connection delay	Yes	0 < *x* <= 60000 milliseconds	No	(not applicable)
Connection retry limit	No	–1 <= *x* <= 10000 –1 = forever (default)	No	(not applicable)
Reconnection limit	No	–1 <= *x* <= 10000 –1 = forever (default)	No	(not applicable)
TLS enabled[b/]	No	yes no (default)	No	yes no (default)
TLS private key[b/]	Yes[c/]	*private key*	Yes[c/]	*private key*

Table 3.1 Basic link attributes (page 2 of 2)

Attribute Name	Client		Server	
	Required	Values	Required	Values
TLS public key certificate[b/]	Yes[c/]	*public key certificate*	Yes[c/]	*public key certificate*
TLS encryption method[b/]	Yes[c/]	*encryption method*	Yes[c/]	*encryption method*
TLS root certificate[b/]	Yes[c/]	*root certificate*	Yes[c/]	*root certificate*
TLS CRL source[b/]	No	CDP local none (default)	No	CDP local none (default)
TLS local CRL[b/]	Yes[d/]	*local CRL location*	Yes[d/]	*local CRL location*

[a/] Required only when "Data ACK enabled" is set to "yes."
[b/] TLS support is optional. If TLS is provided, these attributes must be supported.
[c/] Required when "TLS enabled" is set to "yes."
[d/] Required when "TLS CRL source" is set to "local."

3.2 Requirements

A Class D implementation providing the mandatory Protocol Layer shall meet the following requirements:

r[1] All network communication for Class D shall be implemented using TCP as specified in IETF RFC 1122.

3.2.1 Link Initiation

r[2] A Class D node shall support the ability to configure one or more communication links.

r[3] For each link, a node shall use and support the configuration of one TCP socket.

r[4] For each link, a node shall accept configuration of the following:

r[4.1] TCP role. This required attribute indicates whether the node shall act as a TCP client or TCP server. For each link, one node shall be configured as a TCP server and one node shall be configured as a TCP client.

r[4.1.1] Server: The node shall manage the link using a TCP server socket that listens for connections.

r[4.1.2] Client: The node shall manage the link using a TCP client socket that establishes connections with server sockets.

r[4.2] Mode. This optional attribute indicates the direction in which data messages are permitted for the link. For each link, either both nodes shall be configured as bidirectional or one node shall be configured as send only and the other node shall be configured as receive only.

r[4.2.1] Send only: The node shall send data messages on the link, and it shall not receive data messages on the link. The node shall listen for acknowledgments and negative acknowledgments to sent data messages according to its *data ACK enabled* attribute. The node shall send keep-alive messages if its TCP role is configured as client and it is configured to do so.

r[4.2.2] Receive only: The node shall listen for data messages on the link, and it shall not send data messages on the link. The node shall send acknowledgments and negative acknowledgments for received data messages according to its *data ACK enabled* attribute. The node shall send keep-alive messages if its TCP role is configured as client and it is configured to do so.

r[4.2.3] Bidirectional: The node shall send and receive data messages on the link. The node shall send acknowledgements and negative acknowledgments for received messages according to its *data ACK enabled* attribute. The node shall send keep-alive messages on the link if its TCP role is configured as client and it is configured to do so.

r[4.2.4] If no mode is specified, the default configuration shall be bidirectional.

r[4.3] Local address. This optional attribute indicates the address to which the node shall be bound for the link. It shall be configurable and expressed as an IPv4 address or DNS name. If no local address is specified, the default configuration for a TCP client node shall be *any*, meaning that the system can bind the node to any available address. If no local address is specified, the default configuration for a TCP server node shall be *all*, meaning that the system shall bind the node to all available addresses (i.e., network interfaces).

r[4.4] Remote address. For a TCP server node, this attribute is optional and indicates the address from which the TCP client node shall connect to it. It shall be configurable and expressed as an IPv4 address, a DNS name, or *none* if the TCP server will not validate the TCP client address. If no remote address is specified, the default configuration for a TCP server node shall be *none*. For a TCP client node, this attribute is required and indicates the address of the TCP server node to which it shall connect. It shall be configurable and expressed as an IPv4 address or DNS name.

r[4.5] Local port. This attribute indicates the TCP port to which the node shall be bound for the link. It shall be required for a TCP server node. It shall be configurable and expressed as an integer greater than 1024 and less than or equal to 65535. If no local port is specified, the default configuration for a TCP client node shall be *any*, meaning that the system can bind the node to any available port.

r[4.6] Log traffic. This optional attribute indicates whether or not link traffic should be logged for debugging purposes. If this attribute is set to *yes*, all link traffic shall be logged by the node in a human-readable format. If no value is specified, the default configuration shall be *no*, meaning that link traffic shall not be logged. Binary data shall be expressed using a hexadecimal representation. Each log entry shall include a time stamp. The logging format and mechanism are not specified. Associated log files shall be maintained such that they do not grow without bound.

r[4.7] Keep-alive interval. For a TCP client node, this required attribute indicates the rate at which keep-alive messages shall be sent. For a TCP server node, this required attribute indicates the rate at which keep-alive messages shall be expected to be received. It shall be configurable and expressed in milliseconds with a valid range of 0 to 60,000 (60 seconds). A link's TCP server node should be configured with a greater value than its TCP client node to allow for network propagation of keep-alive messages. A value of 0 shall indicate that no keep-alive messages are sent or expected. The *keep-alive interval* attribute should be configured as 0 on both ends of a link if Class D keep-alive operation is to be disabled.

r[4.8] Data ACK enabled. This required attribute indicates whether or not acknowledgments (and negative acknowledgments) shall be sent or expected in response to data messages. It shall be configurable and expressed as *yes* or *no*. The *data ACK enabled* attribute must be configured the same on both ends of a link.

r[4.9] Data ACK timeout. This attribute indicates how much time shall be allowed to elapse between the sending of a data message and the receipt of an acknowledgement message. It is required for a node if *data ACK enabled* is set to *yes*. It shall be configurable and expressed in milliseconds with a valid range of 1 to 60,000 (60 seconds). If the *data ACK enabled* attribute is set to *no,* this attribute shall have no meaning.

r[4.10] Data NAK retry limit. This required attribute indicates how many times a sending node shall attempt to send a message for which it receives a negative acknowledgment. It shall be configurable and expressed as an integer with a valid range of 0 to 10. The error code associated with a negative acknowledgment shall determine whether or not retries are to be made. If the error code indicates that no retries are to be made, this attribute shall be ignored.

r[4.11] TLS enabled. If TLS is implemented, this optional attribute indicates whether or not the node shall enforce the Transport Layer Security protocol for link authentication and encryption. It shall be configurable and expressed as either *yes* or *no*. If no value is specified, the default configuration shall be *no*.

r[4.12] TLS private key. If TLS is implemented, this attribute indicates the private key assigned to this node for the link. If TLS is enabled for the link, the private key shall be used by the node to identify itself to the remote node. This attribute shall be required if *TLS enabled* is set to *yes*. If *TLS enabled* is not set to *yes*, this attribute shall have no meaning.

r[4.13] TLS public key certificate. If TLS is implemented, this attribute indicates the public key certificate assigned to this node. This node shall provide the public key certificate when negotiating with the remote node. This attribute shall be required if *TLS enabled* is set to *yes*. If *TLS enabled* is not set to *yes*, this attribute shall have no meaning.

r[4.14] TLS encryption method. If TLS is implemented, this attribute indicates the method for encrypting data that shall be used by the node when sending packets on this link. A NULL method shall be supported, which shall disable encryption. This attribute shall be required if *TLS enabled* is set to *yes*. If *TLS enabled* is not set to *yes*, this attribute shall have no meaning.

r[4.15] TLS root certificate. If TLS is implemented, this attribute indicates the public key certificate of the Certificate Authority that signed the public key certificate of the remote node. The node shall use the TLS root certificate to validate the public key certificate received from the remote node. This attribute shall be required if *TLS enabled* is set to *yes*. If *TLS enabled* is not set to *yes*, this attribute shall have no meaning.

r[4.16] TLS CRL source. If TLS is implemented, this optional attribute indicates how the node should locate the Certificate Revocation List to be used when validating the connecting node's public certificate. A value of *CDP* indicates that the node shall refer to the CDP given in the connecting node's public certificate. A value of *local* indicates that the node shall refer to the locally stored CRL. A value of *none* indicates that the node shall not validate the connecting node's public certificate against a CRL. If no value is specified, the default configuration shall be *none*.

r[4.17] TLS local CRL. If TLS is implemented, this attribute indicates the Certificate Revocation List stored locally on the node. This attribute shall be required if *TLS CRL source* is set to *local*. If *TLS CRL source* is not set to *local*, this attribute shall have no meaning.

r[5] For each TCP client link, a node shall accept configuration of the following additional parameters:

r[5.1] Remote port. For a TCP client node, this attribute is required and indicates the TCP server port to which it shall connect. It shall be configurable and expressed as an integer greater than 1024 and less than or equal to 65535. This attribute shall have no meaning for a TCP server node.

r[5.2] Keep-alive ACK timeout. For a TCP client node, this required attribute indicates how long the node shall wait for an acknowledgment to a keep-alive message before terminating the link. It shall be configurable and expressed in milliseconds with a valid range of 0 to 60,000 (60 seconds). This attribute shall have no meaning for a TCP server node.

r[5.3] Connection attempt timeout. For a TCP client node, this required attribute indicates how much time shall be allowed to elapse while making a single attempt to establish a connection (i.e., open a TCP/IP socket). It shall be configurable and expressed in milliseconds with a valid range of 0 to 60,000 (60 seconds). This attribute shall have no meaning for a TCP server node.

r[5.4] Connection delay. For a TCP client node, this required attribute indicates how much time the node shall wait before retrying a connection after the previous attempt failed. It shall be configurable and expressed in milliseconds with a valid range of 0 to 60,000 (60 seconds). This attribute has no meaning for a TCP server node.

r[5.5] Connection retry limit. For a TCP client node, this optional attribute indicates how many times an attempt shall be made to establish a connection in the event that connection attempts are failing. It shall be configurable and expressed as an integer with a valid range of –1 to 10,000. A value of –1 shall mean retry forever. This attribute has no meaning for a TCP server node. If no value is specified, the default configuration shall be –1.

r[5.6] Reconnection limit. For a TCP client node, this optional attribute indicates the number of times that the client shall attempt to reconnect the link before giving up. It shall be configurable and expressed as an integer with a valid range of –1 to 10,000. A value of –1 shall mean retry forever. If no value is specified, the default configuration shall be –1. The number of reconnection attempts is not reset between successful reconnections, only at system startup. This attribute has no meaning for a TCP server node.

r[5.7] When a TCP client node reaches the connection retry limit, the node shall do the following:

r[5.7.1] Log a descriptive error message containing the time the limit was reached, the link identifier, and the nature of the last connection error; and

r[5.7.2] Request that the Management Layer, if implemented, send an alert to the system management function containing the same information that was logged for the event.

r[5.8] When a TCP client node reaches the reconnection limit, the node shall do the following:

r[5.8.1] Log a descriptive error message containing the time the limit was reached and the link identifier; and

r[5.8.2] Request that the Management Layer, if implemented, send an alert to the system management function containing the same information that was logged for the event.

Note: The following examples are intended to clarify the TCP client link attributes associated with connecting to another Class D node. Assume that a Class D client node is configured as follows*:

Connection attempt timeout (CAT): 5,000 milliseconds
Connection delay (CD): 2,000 milliseconds
Connection retry limit (CRL): 2
Reconnection limit (RL): 1
Remote address**: address$_1$, address$_2$
Example 1. The following is a possible sequence of events for the node:

- Attempt connection to address$_1$
- Abandon attempt after waiting 5,000 milliseconds with no response
- Wait 2,000 milliseconds before the next attempt
- Attempt connection to address$_2$
- Abandon attempt after waiting 5,000 milliseconds with no response
- Wait 2,000 milliseconds before next attempt
- Attempt connection to address$_1$
- Abandon attempt after waiting 5,000 milliseconds with no response
- Give up on link; the CRL allows for only two connection retries

Example 2. The following is a possible sequence of events for the node:

- Attempt connection to address$_1$
- Connection to address$_1$ is successful
- Connection is terminated because of a missed acknowledgment
- Attempt connection to address$_2$
- Abandon attempt after waiting 5,000 milliseconds with no response
- Wait 2,000 milliseconds before next attempt
- Attempt connection to address$_1$
- Connection to address$_1$ is successful
- Connection is terminated because of a missed acknowledgement
- Give up on link; RL allows for only one reconnection retry

* The given link attributes are not typical and are meant to simplify the example.

** This example uses two remote addresses as supported by a Class D High Availability Layer implementation. For a Protocol Layer implementation only, a single address is used and the connection retries are made against the same address.

r[6] For each link not meeting the link configuration requirements established by this document (e.g., missing required attributes, invalid attribute values, etc), the node shall do the following:

r[6.1] Refrain from attempting to initiate the link;

r[6.2] Log a descriptive error message containing the time the error occurred, the link identifier, and the nature of the error; and

r[6.3] Request that the Management Layer, if implemented, send an alert to the system management function containing the same information that was logged for the event.

r[7] If a TCP client link is configured with a *keep-alive interval* that is smaller than or equal to the *keep-alive ACK timeout*, the node shall do the following:

r[7.1] Refrain from attempting to initiate the link;

r[7.2] Log a descriptive error message containing the time that the error occurred, the link identifier, and the nature of the error; and

r[7.3] Request that the Management Layer, if implemented, send an alert to the system management function containing the same information that was logged for the event.

r[8] If a node is initiating a TCP client link, the node shall open a TCP socket to the configured IPv4 address or DNS name and port.

r[9] If a node is initiating a TCP client link and the time to open a TCP socket exceeds the configured *connection attempt timeout* value, the node shall do the following:

r[9.1] Abort the attempt to open the socket;

r[9.2] Log a descriptive error message containing the time that the error occurred, the link identifier, the nature of the error, and the recovery action taken by the node; and

r[9.3] Request that the Management Layer, if implemented, send an alert to the system management function containing the same information that was logged for the event.

r[10] If a node is initiating a TCP client link and it aborts the attempt, the node shall retry initiating the link until the configured *connection retry limit* value is exceeded. If the *connection retry limit* is configured as –1, the node shall attempt to connect indefinitely. The node shall log a descriptive error message indicating the beginning of each connection attempt.

r[11] A node initiating a TCP client link shall wait the amount of time specified by the *connection delay* configuration parameter before retrying a connection.

r[12] If a node is initiating a TCP server link, the node shall listen on the configured TCP *local port* for a client connection.

r[13] A node shall disable the use of the Nagle algorithm on TCP connections.

Note: On many TCP implementations, this is performed by specifying the TCP no-delay option for the socket.

r[14] If a node terminates a link because of processing errors (e.g., missed ACK, missed keep-alive, unrecoverable NAK), the node shall do the following:

r[14.1] Attempt to reestablish the link and continue operation until the number of such attempts has exceeded the configured *reconnection limit* value. If the *reconnection limit* value is –1, then the node shall attempt to reestablish the link forever. The number of attempts is counted from the time the link was initiated at system startup. It is not reset after a connection is reestablished;

r[14.2] Log a descriptive error message containing the time the error occurred, the link identifier, the nature of the error, and the recovery action taken by the node; and

r[14.3] Request that the Management Layer, if implemented, send an alert to the system management function containing the same information that was logged for the event.

r[15] When terminating a link in response to an error condition, a TCP server node shall close the socket to the TCP client. The listening server socket shall not be terminated and shall continue listening for reconnection attempts.

3.2.2 Message Transport

3.2.2.1 Message Transmission

All messages transmitted between nodes shall use the following message format with fields occurring in the order given:

Table 3.2 Class D message structure

Field	Size (in bits)	Usage / Notes
Start of message delimiter (a.k.a. STX)	8	A constant 8-bit value used to indicate that a Class D message header and body follow.
Begin Class D Header		
Protocol version (a.k.a. header version)	8	Class D protocol header version.
COMMID (a.k.a. message number)	32	Used to support acknowledgments and distinguish messages.
Message type (a.k.a. message ID)	8	Indicates the type of message carried in the Class D body.
Message version	8	Indicates the version of the message carried in the Class D body.
Data length	32	Indicates the number of bytes making up the Class D body.
End Class D Header and Begin Class D Body		
Message body	*as indicated by data length in the Class D header*	The message body as described by the header.
End Class D Body		
End of message delimiter (a.k.a. ETX)	8	A constant 8-bit value used to indicate the end of a Class D message.

r[16] A node shall use big-endian (most significant byte first) to order multi-byte values within the Class D header.

3.2.2.1.1 Start of Message Delimiter (a.k.a. STX)

This constant value (ASCII 0x02) indicates that what follow are a Class D header and message body. This can be used for checking synchronization with the byte stream. The Class D protocol does not require a node to perform byte stuffing.

r[17] A node shall place the STX value of 0x02 at the beginning of each Class D message immediately preceding the Class D header.

3.2.2.1.2 Protocol Version (a.k.a. Header Version)

This value indicates the version of the Class D protocol used to construct the message. See paragraph 8.0 for more information on possible values.

3.2.2.1.3 COMMID (a.k.a. Message Number)

r[18] For each message sent on a link (with the exception of retransmits), the message shall be assigned a COMMID one greater than the COMMID from the previous message sent on the link. This COMMID is used in generating acknowledgments. The COMMIDs assigned to messages being sent by a node on a link are independent of (1) the COMMIDs assigned to messages being received by the node and (2) the COMMIDs assigned to messages being sent by the node on other links.

r[19] If the COMMID exceeds 4,294,967,295 (the maximum value possible for an unsigned 32-bit value), the COMMID shall roll over to the value of 1.

r[20] The first message on each link shall start with a COMMID of 1. If the link is reestablished, the COMMID shall also restart at 1.

r[21] If a node receives a message with a COMMID of 0 or receives a message on a link with a COMMID that is not one greater than the COMMID of the previous message received on the link (unless the COMMID has rolled over r[19] or the node requested a retransmit of the previous message r[39]), the node shall do the following:

r[21.1] Discard the message;

r[21.2] Terminate the link and attempt to reconnect if configured to do so;

r[21.3] Log a descriptive error message containing the time the error occurred, the link identifier, the nature of the error, and the recovery action taken by the node; and

r[21.4] Request that the Management Layer, if implemented, send an alert to the system management function containing the same information that was logged for the event.

3.2.2.1.4 Message Type (a.k.a. Message ID)

The following Class D message types are defined:

Table 3.3 Class D message types

Message Type	ID
Data message	1
Acknowledgment	2
Negative acknowledgment	3
Keep alive	4
Test message (Protocol Conformance)	30
Test echo request (Protocol Conformance)	31
Test echo response (Protocol Conformance)	32
Echo request (Operational Diagnostics)	40
Echo response (Operational Diagnostics)	41

3.2.2.1.5 Message Version

Note: Message version is maintained for historical purposes. As of this version, it is deprecated and will be removed in a future version of the Class D protocol. See paragraph 8.0 for current message versions.

This value specifies the version of the Class D message (e.g., data message version 2).

3.2.2.1.6 Data Length

The data length value indicates the number of bytes in the message body. The maximum body size for a Class D message shall not exceed 0x7FFFFFFF (2,147,483,647) bytes.

3.2.2.1.7 End of Message Delimiter (a.k.a. ETX)

This constant value (ASCII 0x03) indicates the end of a Class D message. This can be used for checking synchronization with the byte stream. The Class D protocol does not require a node to perform byte stuffing.

r[22] A node shall place the ETX value of 0x03 at the end of each Class D message immediately after the Class D message body.

3.2.2.2 Message Delivery

r[23] A node shall send data messages only on a link that is configured with a mode of either send only or bidirectional.

r[24] All data messages sent by a node shall conform to the Class D message format specified in paragraph 3.2.2.1 as shown below:

Table 3.4 Class D data message

Field	Value
Start of message delimiter (a.k.a. STX)	0x02
Begin Class D Header	
Protocol version (a.k.a. header version)	*variable*
COMMID (a.k.a. message number)	*variable*
Message type (a.k.a. message ID)	1
Message version	*variable*
Data length	*variable*
End Class D Header and Begin Class D Body	
Message body	*variable*
End Class D Body	
End of message delimiter (a.k.a. ETX)	0x03

r[25] A node shall transmit a message on a link only when acknowledgment of the preceding message has been received. The only exceptions to this are when transmitting the first message on a link, when retransmitting a message after a negative acknowledgment has been received, or when the *data ACK enabled* is configured as no, in which case acknowledgements are not sent or expected.

r[26] For each link, a node shall have at most one unacknowledged message (data or keep alive) at any time unless *data ACK enabled* is configured as no, in which case acknowledgments are not sent or expected.

r[27] For each link, if a node does not receive an acknowledgment message for a transmission within the *data ACK timeout* configured for that link, the node shall terminate the link and attempt to reconnect if configured to do so.

r[28] For each link, if a node has retransmitted a message because of negative acknowledgments a number of times that exceed the configured *data NAK retry limit* value, the node shall do the following:

r[28.1] Terminate the link and attempt to reconnect if configured to do so;

r[28.2] Log a descriptive error message containing the time the error occurred, the link identifier, the nature of the error, and the recovery action taken by the node; and

r[28.3] Request that the Management Layer, if implemented, send an alert to the system management function containing the same information that was logged for the event.

r[29] For each link, if a node receives a message that is not delimited by the STX and ETX characters, the node shall do the following:

r[29.1] Terminate the link and attempt to reconnect if configured to do so;

r[29.2] Log a descriptive error message containing the time the error occurred, the link identifier, the nature of the error, and the recovery action taken by the node; and

r[29.3] Request that the Management Layer, if implemented, send an alert to the system management function containing the same information that was logged for the event.

3.2.2.3 Message Acknowledgement

When received, the acknowledgement message indicates that the message whose COMMID is in the body of the acknowledgment message has been received by the other side of the link and no longer needs to be considered for retransmission. The next message, if any, can then be transmitted. Note that this acknowledgment is not an application-level signal.

r[30] A node shall send a Class D acknowledgment message for each message it successfully receives after validating that the message conforms to the format and version of the Class D protocol supported by the node. The only exception to this is when the *data ACK enabled* attribute for the link is set to *no*, in which case acknowledgments are not sent or expected.

r[31] A Class D acknowledgment message sent by a node shall conform to the Class D message format specified in paragraph 3.2.2.1 as shown in Table 3.5:

Table 3.5 Class D acknowledgment (ACK) message

Field	Value
Start of message delimiter (a.k.a. STX)	0x02
Begin Class D Header	
Protocol version (a.k.a. header version)	*variable*
COMMID (a.k.a. message number)	*variable*
Message type (a.k.a. message ID)	2
Message version	*variable*
Data length	4
End Class D Header and Begin Class D Body	
COMMID	*the COMMID of the message being acknowledged*
End Class D Body	
End of message delimiter (a.k.a. ETX)	0x03

r[32] A node shall send Class D acknowledgment messages on the same link that the message being acknowledged was received on.

r[33] A node shall place the COMMID of the message being acknowledged into the COMMID field of the acknowledgment message body. Note that this will likely differ from the COMMID used in the COMMID field of the header.

3.2.2.4 Error Reporting (a.k.a. Negative Acknowledgment)

The negative acknowledgment message indicates that a message has not been accepted by the other end of the link and needs to be considered for retransmission. The body of the negative acknowledgment message contains a COMMID field whose value is the COMMID of the message rejected. Note that this negative acknowledgment is not an application-level signal.

r[34] If a node terminates a link or detects that a link has been terminated, the node shall do the following:

r[34.1] Log a descriptive error message containing the time the link was terminated or the termination as detected, the link identifier, the nature of the error, and the recovery action taken by the node; and

r[34.2] Request that the Management Layer, if implemented, send an alert to the system management function containing the same information that was logged for the event.

r[35] If a node sends a negative acknowledgment message, the node shall do the following:

r[35.1] Log a descriptive error message containing the time the error occurred, the link identifier, the nature of the error, and the recovery action taken by the node; and

r[35.2] Request that the Management Layer, if implemented, send an alert to the system management function containing the same information that was logged for the event.

r[36] If a node receives a message that is delimited by the STX and ETX bytes but does not otherwise conform to the message format specified in paragraph 3.2.2.1, the node shall do the following:

r[36.1] Discard the received message;

r[36.2] Send a negative acknowledgment message conforming to the Class D message format specified in paragraph 3.2.2.1 as shown in Table 3.6 (except when the *data ACK enabled* attribute for the link is set to *no*, in which case negative acknowledgments are not sent or expected); and

r[36.3] Log a descriptive error message containing the time the error occurred, the link identifier, the nature of the error, and the recovery action taken by the node; and

r[36.4] Request that the Management Layer, if implemented, send an alert to the system management function containing the same information that was logged for the event.

Table 3.6 Class D negative acknowledgment (NAK) message

Field	Value
Start of message delimiter (a.k.a. STX)	0x02
Begin Class D Header	
Protocol version (a.k.a. header version)	*variable*
COMMID (a.k.a. message number)	*variable*
Message type (a.k.a. message ID)	3
Message version	*variable*
Data length	5
End Class D Header and Begin Class D Body	
COMMID	*the COMMID of the message being acknowledged*
Error code	*refer to Table 3.8, "Class D NAK error codes," for allowed values*
End Class D Body	
End of message delimiter (a.k.a. ETX)	0x03

r[37] A node shall send negative acknowledgment messages on the same link that the message being rejected was received on.

r[38] When sending a negative acknowledgment, a node shall indicate the reason that the negative acknowledgment is being sent by setting the error code field using the error codes shown in Table 3.7:

Table 3.7 Class D NAK error codes

Message Type	Retransmit	Error Code
Class D protocol version not supported	No	1
Invalid message ID or message ID is not supported	No	2
The version number for this message ID is not supported	No	3
Message size is not supported by the receiving node	No	4
Unable to secure message. *This error code shall be sent only by a node implementing the Persistence Layer.*	Yes	5

r[39] If a node receives a negative acknowledgment, it shall do the following:

r[39.1] Retransmit the message or terminate the link according to the value indicated in the *retransmit* column of Table 3.8, "Class D NAK error codes," for the error code encountered. If a retransmission is indicated, the node shall retransmit the message according to the link's *data NAK retry limit* attribute after a delaying for the time specified by the link's *retransmit delay* attribute. If a retransmission is not indicated, the node shall terminate the link and attempt to reconnect if configured to do so;

r[39.2] Log a descriptive error message containing the time the error occurred, the link identifier, the nature of the error, and the recovery action taken by the node; and

r[39.3] Request that the Management Layer, if implemented, send an alert to the system management function containing the same information that was logged for the event.

3.2.3 Link Management

r[40] A Class D node shall be capable of detecting the loss of a link through the use of Class D keep-alive messages and the implementation's use of other methods provided by the TCP/IP stack and operating system.

r[41] If a node detects the loss of a link, the node shall do the following:

r[41.1] Attempt to reconnect according to the link initiation requirements in paragraph 3.2.1;

r[41.2] Log a descriptive error message containing the time the loss was detected, the link identifier, the nature of the error, and the recovery action taken by the node; and

r[41.3] Request that the Management Layer, if implemented, send an alert to the system management function containing the same information that was logged for the event.

r[42] For each link with a configured mode of send only, if a data message is received on the link, a node shall do the following:

r[42.1] Terminate the link;

r[42.2] Log a descriptive error message containing the time the error occurred, the link identifier, the nature of the error, and the recovery action taken by the node; and

r[42.3] Request that the Management Layer, if implemented, send an alert to the system management function containing the same information that was logged for the event.

3.2.4 Link Monitoring

The Class D protocol provides a keep-alive message for each link. This is used to determine if two connected nodes are communicating and operating within a configurable time frame. A node that receives a Class D keep-alive message must respond with an acknowledgment message.

Note: Note that the Class D keep-alive message is distinct from the keep-alive functionality of TCP. Class D keep-alive messages are provided for the case where TCP keep-alive messages are not supported or desired.

r[43] A Class D keep-alive message sent by a node shall conform to the Class D message format specified in paragraph 3.2.2.1 as shown in Table 3.8:

Table 3.8 Class D keep-alive message

Field	Value
Start of message delimiter (a.k.a. STX)	0x02
Begin Class D Header	
Protocol version (a.k.a. header version)	*variable*
COMMID (a.k.a. message number)	*variable*
Message type (a.k.a. message ID)	4
Message version	*variable*
Data length	0
End Class D Header and Begin Class D Body	
End Class D Body	
End of message delimiter (a.k.a. ETX)	0x03

r[44] For each link with a configured TCP role of client, a node shall send a Class D keep-alive message whenever no messages have been sent or received for the duration of the configured keep-alive interval link attribute.

r[45] For each link, a Class D node shall provide the capability to disable Class D keep-alive operation by setting the *keep-alive interval* link attribute to 0.

r[46] For each link, if Class D keep-alive operation is disabled, TCP keep-alive operation shall be enabled for the link.

r[47] For each link, TCP keep-alive operation shall be disabled if Class D keep-alive operation is enabled.

r[48] For each link with a configured TCP role of server, a Class D node shall provide the capability to respond to a keep-alive message by sending a Class D acknowledgment message or negative acknowledgment message.

r[49] For each link with a configured TCP role of client, if a node does not receive acknowledgment of a Class D keep-alive message within the configured *keep-alive ACK timeout*, the node shall do the following:

r[49.1] Terminate the link and attempt to reconnect if configured to do so;

r[49.2] Log a descriptive error message containing the time the error occurred, the link identifier, the nature of the error, and the recovery action taken by the node; and

r[49.3] Request that the Management Layer, if implemented, send an alert to the system management function containing the same information that was logged for the event.

r[50] For each link with a configured TCP role of client, if a node receives a negative acknowledgment to a Class D keep-alive message, the node shall do the following:

r[50.1] Terminate the link and attempt to reconnect if configured to do so;

r[50.2] Log a descriptive error message containing the time the error occurred, the link identifier, the nature of the error, and the recovery action taken by the node; and

r[50.3] Request that the Management Layer, if implemented, send an alert to the system management function containing the same information that was logged for the event.

3.2.5 State Diagrams

The following state diagrams illustrate the behavior of a node for each of the possible node configurations.

3.2.5.1 TCP Client Send Only Socket

TBD

3.2.5.2 TCP Server Send Only Socket

TBD

3.2.5.3 TCP Client Receive Only Socket

TBD

3.2.5.4 TCP Server Receive Only Socket

TBD

3.2.5.5 TCP Server Bidirectional Socket

TBD

3.2.5.6 TCP Client Bidirectional Socket

TBD

3.2.6 Security

Class D protocol security may be provided by the mandatory Protocol Layer using the Transport Layer Security (TLS) 1.2 protocol as defined by RFC 5246. The TLS protocol allows applications to communicate in a way that is designed to prevent eavesdropping, tampering, or message forgery. TLS support is optional; however, if TLS support is provided by an implementation, it must satisfy the requirements given in this specification.

r[51] If TLS is implemented, a node shall support the Transport Layer Security (TLS) 1.2 protocol as defined by RFC 5246.

r[52] If TLS is implemented, a node shall utilize the bilateral connection mode (a.k.a mutual authentication) of the Transport Layer Security protocol such that both ends of a link can be assured with whom they are communicating.

r[53] When authenticating a remote node, a node shall ensure that the remote node's public key certificate is not revoked according to the CRL source configured using the *TLS CRL source* attribute. If *TLS CRL source* is configured as *CDP*, the node shall obtain a current CRL from the CRL Distribution Point identified in the remote node's public key certificate. If *TLS CRL source* is configured as *local*, the node shall use a locally stored CRL. If *TLS CRL source* is configured as *none*, the node shall not validate the remote node's public key certificate against a CRL.

r[54] If TLS is implemented, a node shall allow the configuration of each link with a public key certificate to be used when negotiating with a remote node. This public key certificate shall be provided to the remote node as part of the TLS authentication process.

r[55] If TLS is implemented, a node shall allow the configuration of each link with a *TLS private*

key to be used for identifying itself as the originator of messages sent to a remote node. This private key is matched with the public key certificate provided to the remote node during TLS negotiation.

r[56] If TLS is implemented, a node shall allow the configuration of each link with an encryption method by setting the *TLS encryption method* attribute for the link. A *NULL* encryption method shall be supported, which disables encryption for the link.

r[57] If TLS is implemented, a node shall allow the Transport Layer Security protocol to be enabled or disabled for each link by configuring the *TLS enabled* attribute for the link. A node shall require configuration of link attributes pertaining to TLS only if TLS is implemented and enabled for the link.

4.0 CLASS D HIGH AVAILABILITY LAYER

The optional High Availability Layer provides support for survivability of Class D links. It describes the required behavior of a Class D node using alternate addresses and ports to recover from link or node failure. The functions added by the High Availability Layer complement the required Protocol Layer but they do not replace it. Class D implementations providing support for the High Availability Layer must still satisfy all requirements of the Protocol Layer.

Note: To clarify whether a given Class D node implements the High Availability Layer, the term *HA node* will be used in place of *node*. If a node also provides persistence, the term *persistent HA node* will be used. A link provided by such implementations is referred to as an *HA link* or *persistent HA link*.

4.1 Link Attributes

Table 4.1 lists altered and additional attribute names used to describe the configuration of a Class D HA link. It is strongly recommended that the same names be used when implementing the protocol.

Table 4.1 High availability link attributes

Attribute	Values	Definition
Remote address	*variable*	This attribute replaces the same attribute as defined by the Protocol Layer. For a TCP server node, this attribute, if set, indicates the address from which the TCP client node will be connecting to it. It can be expressed as a list of one to eight IPv4 addresses or DNS names, a range of IPv4 addresses, or left blank if the TCP server will not validate the TCP client address. For a TCP client node, this attribute is required and indicates the address of the TCP server node to which it will connect. It can be expressed as a list of one to eight IPv4 addresses or DNS names or a range of IPv4 addresses.
Remote port	*variable*	This attribute replaces the same attribute as defined by the Protocol Layer. For a TCP client node, this attribute is required and indicates the TCP server port to which it will connect. If *remote address* is configured with a list of IPv4 address or DNS names, a *remote port* shall be configured for each list entry. If *remote address* is configured as a range of IPv4 addresses, a single *remote port* value shall be applied to each address in the range. It must be expressed as an integer greater than 1024 and less than or equal to 65535. This attribute has no meaning for a TCP server node.
Idle timeout	0 <= *x* <= 360,000 milliseconds	Indicates the amount of time that the link can be idle (i.e., without data messages) before reporting the lack of activity. Keep-alive messages do not reset this timeout. A value of 0 disables the idle timeout.
Terminate when idle	no (default) yes	Indicates whether the link should be terminated if it is idle longer than allowed by the *idle timeout* attribute.

4.2 Requirements

A Class D implementation providing the optional High Availability Layer shall meet the following requirements:

r[58] When an HA node detects that its HA status has changed, the node shall do the following:

r[58.1] Log a descriptive error message containing the time the error occurred, the link identifier, the nature of the error, and the recovery action taken by the node; and

r[58.2] Request that the Management Layer, if implemented, send an alert to the system management function containing the same information that was logged for the event.

4.2.1 Link Initiation

r[59] An HA node shall support the ability to configure a minimum of 12 communication links.

r[60] An HA node shall accept configuration of the following:

r[60.1] Remote address. For each TCP client link, this attribute is required and indicates the address of the TCP server node to which it will connect. It shall be expressed as a list of one to eight IPv4 addresses or DNS names or a range of IPv4 addresses. For each TCP server link, this attribute, if set, indicates the address from which the TCP client node will be connecting to it. It shall be expressed as a list of one to eight IPv4 addresses or DNS names, a range of IPv4 addresses, or left blank if the TCP server will not validate the TCP client address.

r[60.2] Remote port. For each TCP client link, this attribute is required and indicates the TCP server port to which it will connect. If *remote address* is configured with a list of IPv4 address or DNS names, a *remote port* shall be configured for each list entry. If *remote address* is configured as a range of IPv4 addresses, a single *remote port* value shall be applied to each address in the range. It shall be expressed as an integer greater than 1024 and less than or equal to 65535. This attribute shall have no meaning for a TCP server node.

r[60.3] Idle timeout. For each TCP server link, this attribute indicates the amount of time that the link can be idle (i.e., without data messages) before the node shall report the lack of activity. Keep-alive messages shall not reset this timeout. A value of 0 shall disable the idle timeout.

r[60.4] Terminate when idle. Indicates whether the link shall be terminated if it is idle longer than allowed by the *idle timeout* attribute. If set to *yes* and an *idle timeout* event occurs, the node shall terminate the link and attempt to reconnect if configured to do so.

r[61] For each link, an HA node shall use the addresses indicated by the *remote address* attribute in a round-robin fashion when attempting a connection or reconnection.

r[62] For each TCP server link, an HA node shall permit a connection to only one remote address and port at a time (i.e., primary and backup links may not be active at the same time).

4.2.2 Link Management

r[63] An HA node shall provide support for dynamically configuring all link attributes such that the HA node need not be shut down or cycled (i.e., without interrupting other links).

r[64] An HA node shall provide support for dynamically configuring the following link attributes such that an HA link need not be shut down or cycled (i.e., without interrupting the modified link to make a change: *log traffic, keep-alive interval, keep-alive ACK timeout, data ACK enabled, data ACK timeout, data NAK retry limit, connection attempt timeout, connection delay, connection retry limit, idle timeout, reconnection limit, remote address (when not impacting the active link), remote port (when not impacting the active link)*.

4.2.3 Link Monitoring

r[65] For each link, if an HA node detects that no message has been received on the link after the *idle timeout* link attribute has elapsed, the node shall do the following:

r[65.1] Log a descriptive error message containing the time the error occurred, the link identifier, the nature of the error, and the recovery action taken by the node; and

r[65.2] Request that the Management Layer, if implemented, send an alert to the system management function containing the same information that was logged for the event.

r[66] For each link, if an HA node detects that a message has been received on the link subsequent to an *idle timeout* event, the node shall do the following:

r[66.1] Log a descriptive error message containing the time the error occurred, the link identifier, the nature of the error, and the recovery action taken by the node; and

r[66.2] Request that the Management Layer, if implemented, send an alert to the system management function containing the same information that was logged for the event.

r[67] For each link, an HA node shall provide the capability to configure the termination of the link when the *idle timeout* event occurs.

4.3 State Diagrams

The following state diagrams illustrate the behavior of an HA node for each of the anticipated failure cases.

4.3.1 Remote Link Lost

TBD

4.3.2 Local Link Lost

TBD

5.0 CLASS D PERSISTENCE LAYER

The optional Persistence Layer provides support for survivability of Class D messages in the event of link or node failure. It adds requirements for securing messages at a receiving node before discarding them on the transmitting node. The functions added by the Persistence Layer complement the required Protocol Layer, but they do not replace it. Class D implementations providing support for the Persistence Layer must still satisfy all requirements of the Protocol Layer.

Note: To clarify whether a given Class D node implements the Persistence Layer, the term *persistent node* will be used in place of *node*. If a node also provides high availability, the term *persistent HA node* will be used. A link provided by such implementations is referred to as a *persistent link* or *persistent HA link*.

5.1 Link Attributes

Table 5.1 lists altered and additional attribute names used to describe the configuration of a Class D persistent link. It is strongly recommended that the same names be used when implementing the protocol.

Table 5.1 Persistence link attributes

Attribute	Values	Definition
Persistence enabled	yes (default) no	Enables message persistence for the link.

5.2 Requirements

A Class D implementation providing the optional Persistence Layer shall meet the following requirements:

r[68] For each link, a persistent node shall protect messages such that they cannot be lost during a link recovery or node fail-over.

r[69] A persistent node shall support configuration of the following:

r[69.1] For each link, an option to enable or disable message persistence.

5.2.1 Link Initiation

r[70] For each link, upon reconnecting with a node for which there are persisted outgoing messages, a persistent node shall send all persisted messages prior to sending any other messages.

Note: An implementation of the Class D Persistence Layer likely has logic based on configurable business rules that determines how to handle the situation where system resource limits are met. For the purposes of this requirement, a persistent node shall send all persisted messages that have not been discarded per the business rules mentioned.

5.2.2 Message Transport

r[71] For each link, a persistent node shall persist a message before signaling the Protocol Layer to send a Class D message acknowledgment for the message.

r[72] For each link, a persistent node shall ensure the ability to retransmit an outgoing message until a corresponding acknowledgment message has been received such that the message cannot be lost during a link recovery or node fail-over.

6.0 CLASS D MANAGEMENT LAYER

The optional Management Layer provides support for remotely and securely configuring Class D links and optional protocol features, remotely querying statistics and state, and notifying external agents when error conditions occur. A Class D node that implements the Management Layer must, at a minimum, provide the ability to manage the functions of the Protocol Layer. If other optional layers are supported by the Class D node, the Management Layer must provide the ability to manage the functions of those layers as well.

Note: To clarify whether a given Class D node implements the Management Layer, the term *managed node* will be used in place of *node*. A link provided by such an implementation is referred to as a *managed link*.

6.1 Requirements

A Class D implementation providing the optional Management Layer shall meet the following requirements:

r[73] A managed node shall allow node and link configuration to be performed remotely by the system management function. The system management protocol is not defined.

r[74] A managed node shall allow node and link statistics to be queried remotely by the system management function. The system management protocol is not defined.

r[75] A managed node shall allow node and link statistics to be reset remotely by the system management function. The system management protocol is not defined.

r[76] A managed node shall allow node and link state to be queried remotely by the system management function. The system management protocol is not defined.

r[77] For the mandatory Protocol Layer and any implemented optional layers, a managed node shall provide services to send alerts to the system management function. The system management protocol is not defined.

r[78] A managed node shall provide the ability to remotely read, reset, and write (as appropriate) all options, attributes, statistics, state, and other relevant information pertaining to the optional Class D layers provided by an implementation.

7.0 CLASS D BUILT-IN TEST LAYER

The optional Built-in Test Layer provides support for protocol conformance testing, test automation, and operational diagnostics. A Class D node that implements the Built-in Test Layer should, at a minimum, provide the ability to test and diagnose the functions of the Protocol Layer. If other optional layers are provided with the Class D node, the Built-in Test Layer must provide the ability to test and diagnose the functions of those layers as well.

7.1 Protocol Conformance Requirements

Requirements in this section are not intended to affect interoperability of Class D protocol implementations. They support execution of a common protocol conformance test. Compliance ensures that a common set of tests can be executed on different implementations.

r[79] A node shall support the ability to be configured to establish a test link, to be used for test purposes only, and maintain the link (e.g., send keep-alive messages, acknowledge received messages, and reconnect as configured) until the configuration disables the test link.

r[80] A node shall send and receive protocol conformance messages (i.e., test message, test echo request, and test echo response) only on an established test link.

r[81] If a node receives a protocol conformance message (i.e., test message, test echo request, or test echo response) on a link that is not configured as a test link, the node shall do the following:

r[81.1] Log a descriptive error message containing the time the error occurred, the link identifier, and the nature of the error; and

r[81.2] Discard the message.

r[82] A node shall send and receive data messages only on a link that is not configured as a test link.

r[83] If a node receives a data message on a link that is configured as a test link, the node shall do the following:

r[83.1] Log a descriptive error message containing the time the error occurred, the link identifier, and the nature of the error; and

r[83.2] Discard the message.

r[84] A node shall support the ability to be configured to send between 1 and 50,000 test messages on a test link. The message body of a test message is unspecified.

r[85] A node shall support the ability to be configured to delay the interval between individual test messages from 0 to 60,000 milliseconds.

r[86] A node shall support the ability to be configured to listen for and receive test messages on a test link. Received test messages shall be discarded by the node (i.e., not processed as normal data messages).

r[87] A node shall apply all normal checks for protocol conformance on test messages received on a test link, logging received messages and detected errors if so configured.

r[88] The test message shall conform to the Class D message format specified in paragraph 3.2.2.1 as shown in Table 7.1:

Table 7.1 Class D test message

Field	Value
Start of message delimiter (a.k.a. STX)	0x02
Begin Class D Header	
Protocol version (a.k.a. header version)	*variable*
COMMID (a.k.a. message number)	*variable*
Message type (a.k.a. message ID)	30
Message version	*variable*
Data length	*variable*
End Class D Header and Begin Class D Body	
Message body	*variable*
End Class D Body	
End of message delimiter (a.k.a. ETX)	0x03

r[89] A node shall support the ability to receive a test echo request message on a test link. The test echo request message shall conform to the Class D message format specified in paragraph 3.2.2.1 as shown in Table 7.2:

Table 7.2 Class D test echo request message

Field	Value
Start of message delimiter (a.k.a. STX)	0x02
Begin Class D Header	
Protocol version (a.k.a. header version)	*variable*
COMMID (a.k.a. message number)	*variable*
Message type (a.k.a. message ID)	31
Message version	*variable*
Data length	*variable*
End Class D Header and Begin Class D Body	
Message body	*variable*
End Class D Body	
End of message delimiter (a.k.a. ETX)	0x03

r[90] A node shall support the ability to respond to a test echo request message by sending a test echo response message. The test echo response message body shall contain the same data that was present in the test echo request message body. The test echo response message shall conform to the Class D message format specified in paragraph 3.2.2.1 as shown in Table 7.3:

Table 7.3 Class D test echo response message

Field	Value
Start of message delimiter (a.k.a. STX)	0x02
Begin Class D Header	
Protocol version (a.k.a. header version)	*variable*
COMMID (a.k.a. message number)	*variable*
Message type (a.k.a. message ID)	32
Message version	*variable*
Data length	*same as provided in echo request*
End Class D Header and Begin Class D Body	
Message body	*same as provided in echo request*
End Class D Body	
End of message delimiter (a.k.a. ETX)	0x03

r[91] A node shall support the ability to be configured to log errors detected on a test link. The format of the log is defined by the unit under test but must contain at least the time the error occurred and an indication of the error.

r[92] A node shall support the ability to be configured to log all messages received on the test link.

r[93] A node shall support the ability to be configured to log all messages transmitted on the test link.

r[94] A node shall support the ability to be configured with a starting COMMID other than 1 (i.e., any normally valid COMMID other than 0) on the test link. This is used to test that the COMMID rolls over to 1 without sending an impractical number of messages.

7.2 Operational Diagnostics

r[95] A node shall support the ability to send an echo request message to the connected node to validate node operation. The echo request message shall conform to the Class D message format specified in paragraph 3.2.2.1 as shown in Table 7.4:

Table 7.4 Class D echo request message

Field	Value
Start of message delimiter (a.k.a. STX)	0x02
Begin Class D Header	
Protocol version (a.k.a. header version)	*variable*
COMMID (a.k.a. message number)	*variable*
Message type (a.k.a. message ID)	40
Message version	*variable*
Data length	*variable*
End Class D Header and Begin Class D Body	
Message body	*variable*
End Class D Body	
End of message delimiter (a.k.a. ETX)	0x03

r[96] A node receiving an echo request message shall do the following:

r[96.1] Log a descriptive message; and

r[96.2] Send an echo response message to the node that sent the echo message request.

r[97] A node shall support the ability to respond to an echo request message by sending an echo response message. The echo response message body shall contain the same data that was present in the echo request message body. The echo response message shall conform to the Class D message format specified in paragraph 3.2.2.1 as shown in Table 7.5:

Table 7.5 Class D echo response message

Field	Value
Start of message delimiter (a.k.a. STX)	0x02
Begin Class D Header	
Protocol version (a.k.a. header version)	*variable*
COMMID (a.k.a. message number)	*variable*
Message type (a.k.a. message ID)	41
Message version	*variable*
Data length	*same as provided in echo request*
End Class D Header and Begin Class D Body	
Message body	*same as provided in echo request*
End Class D Body	
End of message delimiter (a.k.a. ETX)	0x03

8.0 VERSION NUMBERS

Note: There is no requirement to maintain compatibility with protocol version 1 of Class D. A future version of Class D will add support for protocol version negotiation, at which point Class D implementations will be required to support protocol version n-1 compatibility.

r[98] The Class D protocol and message version numbers shown in Table 8.1 shall be used as of this revision of the Class D Messaging Specification:

Table 8.1 Class D version number definitions

Field	Version
Class D protocol version	0x02 (2)
Data message	0x02 (2)
Acknowledgment message	0x02 (2)
Negative acknowledgment message	0x02 (2)
Keep-alive message	0x02 (2)
Test message (Protocol Conformance)	0x02 (2)
Test echo request (Protocol Conformance)	0x02 (2)
Test echo response (Protocol Conformance)	0x02 (2)
Echo request (Operational Diagnostics)	0x02 (2)
Echo response (Operational Diagnostics)	0x02 (2)

9.0 CLASS D VERIFICATION TEST PLAN

The Class D Verification Test Plan is intended to validate protocol conformance of an implementation of the Class D protocol. All layers must be validated if they are provided by the implementation under test.

9.1 Prerequisites for the Class D Verification Test Plan

TBD

9.2 Protocol Layer Test Plan

TBD

9.3 High Availability Layer Test Plan

TBD

9.4 Built-in Test Layer Test Plan

TBD

9.5 Management Layer Test Plan

TBD

9.6 Test Plan Procedures

TBD

APPENDIX A
GLOSSARY

Term	Definition
Communication link	Also known as a *link*, this term is used to describe a logical message transport path that has been established between two processes. This transport can be configured in two logical modes: *Unidirectional*—a message transport where data messages flow in only one direction (i.e., one process only sends and the other receives; the receiving process will send acknowledgements to the data). An example where this type of link may be used is for a sensor that is broadcasting data to a subscriber. *Bidirectional*—a message transport where data messages flow in two directions (i.e., both process send and receive data messages). Typical client-server or peer-to-peer interaction is an example where this type of link would be used. The terms *unidirectional* and *bidirectional* describe only the flow of data messages, not the flow of keep-alive messages. Keep-alive messages always flow from the TCP client node to the TCP server node.
Class D node	Also known as *node*, this term is used to describe a process that is at one of the ends of a communication link.
Terminate	This term describes the action of a node closing the TCP/IP socket associated with a link.
Message	This term is used to describe an atomic unit of communication between two Class D nodes. There are several types of messages defined by the Class D protocol including data messages, data acknowledgement messages, and keep-alive messages.
Message acknowledgment	Also known as *ACK*, this term describes the message that a node sends back to another node when it successfully receives a message. The receipt of this message is a signal that the message does not have to be sent again. When sent by a node supporting the Class D High Availability Layer, the acknowledgment also indicates that the received message was persisted such that the sending node can discard it.
Validation	This term describes the process of inspecting a Class D message to ensure that the format (e.g., delimiters, header fields, field order) conforms to this specification and that the header values conform to the defined legal values.
Reconnect	This term describes the action of reestablishing a link to a Class D node in the event that the previous link was terminated. While attempting to reestablish a link, multiple connection attempts may be made. All such connection attempts are considered part of the same reconnection attempt.
Retransmit	This term describes sending a message that is identical to the previously sent message, including the COMMID value in the class D header. Messages may be retransmitted if the sender receives a negative acknowledgement, depending on the reason for the negative acknowledgment.

THIS PAGE LEFT BLANK INTENTIONALLY