BSR/ASHRAE Addendum an to
ANSI/ASHRAE Standard 135-2012

Public Review Draft

Proposed Addendum an to Standard 135-2012, BACnet® - A Data Communication Protocol for Building Automation and Control Networks

Second Public Review (March 2014)
(Draft shows Proposed Changes to Current Standard)

[This foreword and the “rationales” on the following pages are not part of this standard. They are merely informative and do not contain requirements necessary for conformance to the standard.]

FOREWORD

The purpose of this addendum is to present a proposed change for public review. These modifications are the result of change proposals made pursuant to the ASHRAE continuous maintenance procedures and of deliberations within Standing Standard Project Committee 135. The proposed changes are summarized below.

In the following document, language to be added to existing clauses of ANSI/ASHRAE 135-2012 and Addenda is indicated through the use of *italics*, while deletions are indicated by ~~strikethrough~~. Where entirely new subclauses are proposed to be added, plain type is used throughout. Only this new and deleted text is open to comment at this time. All other material in this addendum is provided for context only and is not open for public review comment except as it relates to the proposed changes.

135-2012*an*-1 Add COBS-Encoded MS/TP Frames

Rationale

BACnet now supports higher baud rates for MS/TP. Increasing the frame size will improve throughput. Adding the capability to carry full ethernet-sized frames will open up future possibilities to carry foreign protocol frames.

However, increasing the size of MS/TP payloads beyond the current 501-octet maximum without also replacing the CRC-16-CCITT with a 32-bit CRC will result in an unacceptable increase in the probability of undetected bit errors.

There have been significant advances in coding theory since MS/TP was initially specified. In particular, the design space of CRCs for particular applications has been well explored. The probability of undetected errors (P_{ud}, bit errors passed up to the client) is affected by a combination of CRC length, CRC polynomial, bit error rate, and data word (payload) length.

Among the most recent (and readable) published surveys on the evaluation of CRCs are three papers by Dr. Philip Koopman, formerly of United Technologies, and now on the faculty at Carnegie Mellon University [1][2][3]. Since the CRC-16-CCITT is widely used, many studies use it as a basis for comparison. If a picture is worth a thousand words, then the figure below demonstrates that, all other factors being equal, increasing the code word length (the covered data plus the CRC field) by a factor of three, from 512 octets (4096 bits) to 1536 octets (12288 bits) increases the probability of undetected errors by approximately two orders of magnitude (i.e., the effect is non-linear).

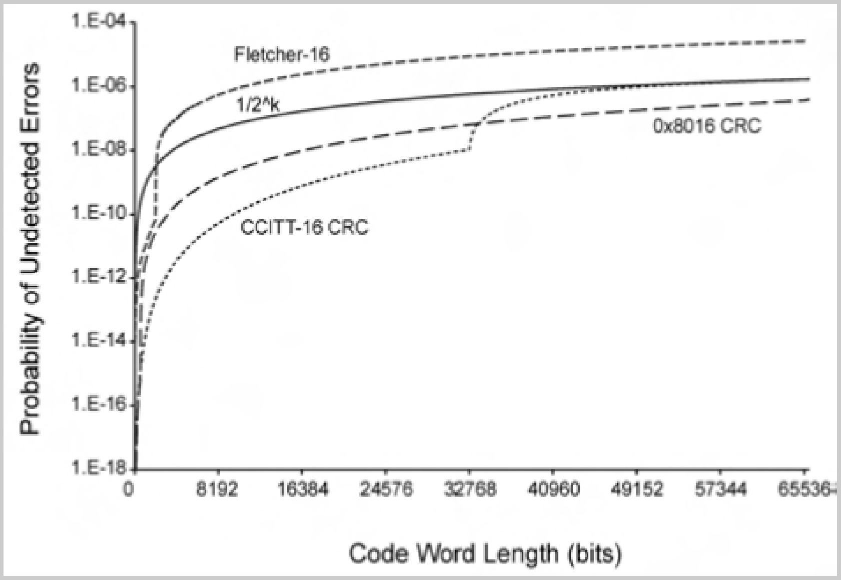


Fig. 10 in [3]. Probability of undetected errors for a 16-bit Fletcher checksum and two 16-bit CRCs at a BER of 10^{-5}. The data values for the Fletcher checksum are the mean of 10 trials using random data.

When we consider that nodes sending large frames are likely to utilize the recently approved higher baud rates (which may further decrease bit error rate margin) the combined effect may be even greater. The large frame discussion should focus on how to a) incorporate a larger CRC into the present MS/TP standard and b) which CRC polynomial should be selected.

Solution Summary:

This proposal can be summarized as a more robust alternative to the present MS/TP design, with BACnet data field lengths between 502 – 1497 octets and a larger (32-bit) CRC that affords greater protection from undetected bit errors. The new frame types proposed here are used to indicate frames that are covered by a larger CRC. The recommended 32-bit polynomial is CRC-32K (Koopman), which has better error detection properties than the IEEE 802.3 Frame Check Sequence polynomial for data lengths up to 114,663 bits [1].

Lastly, MS/TP has not previously defined a method to escape the X'55' that is used to delimit frames. In some cases, this can lead to loss of frame synchronization and dropped frames. Extending the data field would make this problem worse. Therefore, this proposal uses an octet stuffing algorithm called Constant Overhead Byte Stuffing (COBS) to encode the data and CRC-32K fields of extended length frames and thereby eliminates preamble sequences from those fields.

[1] IEEE/IFIP International Conference on Dependable Systems and Networks (DSN 2002), *32-Bit Cyclic Redundancy Codes for Internet Applications*, P. Koopman.

[2] IEEE/IFIP International Conference on Dependable Systems and Networks (DSN 2004), *Cyclic Redundancy Code (CRC) Polynomial Selection for Embedded Networks*, P. Koopman and T. Chakravarty

[3] IEEE Transactions on Dependable and Secure Computing, Vol. 6, No. 1, January – March 2009, *The Effectiveness of Checksums for Embedded Control Networks*, T. Maxino and P. Koopman

[Append to **Clause 5.2.1.2**, p. 20]

...

The value determined by the above constraints will be designated the maximum-transmittable-length. Note that maximum-transmittable-length will in general not be a constant unless minimum values are used for each constraint.

See Clause 19.X for the preferred approach for determining the maximum APDU conveyable by the internetwork.

[Change **Table 6-1**, p. 53]

Table 6-1. Maximum NPDU Lengths When Routing Through Different BACnet Data Link Layers

Data Link Technology	**Maximum NPDU Length**
ISO 8802-3 ("Ethernet"), as defined in Clause 7	1497 octets
ARCNET, as defined in Clause 8	501 octets
MS/TP, as defined in Clause 9	~~501~~ *1497* octets
Point-To-Point, as defined in Clause 10	501 octets
LonTalk, as defined in Clause 11	228 octets
BACnet/IP, as defined in Annex J	1497 octets
ZigBee, as defined in Annex O	501 octets

[Change **Clause 9.1.1.2**, p. 79]

9.1.1.2 Semantics of the Service Primitive

...

Each source and destination address consists of the logical concatenation of a medium access control (MAC) address and a link service access point (LSAP). For the case of MS/TP devices, since ~~MS/TP~~ *the data link interface* supports only the BACnet network layer, the LSAP is omitted and these parameters consist of only the device MAC address.

The 'data' parameter specifies the link service data unit (LSDU) to be transferred by the MS/TP entity. *There is sufficient information associated with the LSDU for the MS/TP entity to determine the length of the data unit.*

...

[Change **Clause 9.1.1.4**, p. 79]

9.1.1.4 Effect on Receipt

Receipt of this primitive causes the MS/TP entity to attempt to send the NPDU using unacknowledged connection-less mode procedures. *BACnet NPDUs less than 502 octets in length are conveyed using BACnet Data Expecting Reply or BACnet Data Not Expecting Reply frames (see Clause 9.3). BACnet NPDUs between 502 and 1497 octets in length, inclusive, are conveyed using BACnet Extended Data Expecting Reply or BACnet Extended Data Not Expecting Reply frames.*

...

[Change **Clause 9.1.2.2**, p. 80]

9.1.2.2 Semantics of the Service Primitive

...

Each source and destination address consists of the logical concatenation of a medium access control (MAC) address and a link service access point (LSAP). For the case of MS/TP devices, since ~~MS/TP~~ *the data link interface* supports only the BACnet network layer, the LSAP is omitted and these parameters consist of only the device MAC address.

The 'data' parameter specifies the link service data unit that has been received by the MS/TP entity. *There is sufficient information associated with the LSDU for the MS/TP entity to determine the length of the data unit. A data unit larger than the maximum supported NPDU shall be silently dropped.*

...

[Change **Clause 9.3**, p. 92]

9.3 MS/TP Frame Format

All frames are of the following format:

Preamble	two octet preamble: X'55', X'FF'
Frame Type	one octet
Destination Address	one octet address
Source Address	one octet address
Length	two octets, most significant octet first
Header CRC	one octet
Data	(present only if Length is non-zero *and frame is a non-encoded type*)
Data CRC	(present only if Length is non-zero *and frame is a non-encoded type*) two octets, least significant octet first
Encoded Data	*(present only if frame is a COBS-encoded type)*
Encoded CRC-32K	*(present only if frame is a COBS-encoded type)* *five octets; CRC-32K generated according to Clause 9.6 and COBS-encoded according to Clause 9.10*
(pad)	(optional) at most one octet of padding: X'FF'

The Frame Type is used to distinguish between different types of MAC frames. Defined types are:

00	Token
01	Poll For Master
02	Reply To Poll For Master
03	Test_Request
04	Test_Response
05	BACnet Data Expecting Reply
06	BACnet Data Not Expecting Reply
07	Reply Postponed
32	*BACnet Extended Data Expecting Reply*
33	*BACnet Extended Data Not Expecting Reply*

Frame Types 8 through *31 and 34 through* 127 are reserved by ASHRAE. *Open standard development organizations wishing to convey network protocols other than BACnet over MS/TP may apply for a Frame Type allocation from the reserved range by contacting the ASHRAE Manager of Standards.*

Frame Types 32 through 127 indicate COBS-encoded frames and shall convey Encoded Data and Encoded CRC-32K fields (see Clause 9.10). All other values for Frame Type indicate a non-encoded frame. Support for COBS-encoded BACnet MS/TP frames is required for BACnet routing nodes and optional for non-routing nodes. Devices that support COBS-encoded BACnet MS/TP frames shall support both BACnet Extended Data Expecting Reply and BACnet Extended Data Not Expecting Reply Frame Types.

Frame Types 128 through 255 are available to vendors for proprietary (non-BACnet) frames. Use of proprietary frames might allow a Brand-X controller, for example, to send proprietary frames to other Brand-X controllers that do not implement BACnet while using the same medium to send BACnet frames to a Brand-Y panel that does implement BACnet. Token, Poll For Master, and Reply To Poll For Master frames shall be understood by ~~both proprietary and BACnet~~ *all MS/TP* master nodes.

The Destination and Source Addresses are one octet each. A Destination Address of 255 (X'FF') denotes broadcast. A Source Address of 255 is not allowed. Addresses 0 to 127 are valid for both master and slave nodes. Addresses 128 to 254 are valid only for slave nodes.

For non-encoded frames, the ~~The~~ Length field specifies the length in octets of the Data field *and its value shall be between 0 and 501.*

For non-encoded frames, the ~~The~~ Data and Data CRC fields are conditional on the Frame Type and the Length, as specified in the description of each Frame Type. If the Length field is zero, that is, if both length octets are zero, then the Data and Data CRC fields shall not be present.

~~The length of the Data field shall be between 0 and 501 octets.~~

For COBS-encoded frames, the Length field specifies the combined length of the Encoded Data and Encoded CRC-32K fields in octets, minus two. (Two octets are subtracted to maintain backward compatibility with devices that implement the SKIP_DATA state in their Receive Frame state machine.)

For COBS-encoded frames, the Length field shall be in the range $N_{min\_COBS\_length}$ to $N_{max\_COBS\_length}$ (see Clause 9.5.3), and the Encoded Data and Encoded CRC-32K fields shall always be present.

~~Subclause~~ *Clause* 9.6 and Annex G describe in detail the generation and checking of the Header, ~~and~~ Data CRC, *and CRC-32K* octets.

Clause 9.10 and Annex X describe in detail the COBS-encoding of the Encoded Data and Encoded CRC-32K fields.

[Change **Clause 9.3.9**, renumbering to make room for new frame types, p. 93]

9.3.~~9~~*11* Frame Types 128 through 255: Proprietary Frames

These frames are available to vendors as proprietary (non-BACnet) frames. The first two octets of the Data field shall specify the unique vendor identification code, most significant octet first, for the type of vendor-proprietary frame to be conveyed. The length of the data portion of a ~~Proprietary~~ *proprietary* frame shall be in the range of 2 to 501 octets.

[Insert new **Clauses 9.3.9** and **9.3.10**, p. 93]

9.3.9 Frame Type 32: BACnet Extended Data Expecting Reply

This COBS-encoded frame is used by master nodes to convey the data parameter of a DL_UNITDATA.request whose data_expecting_reply parameter is TRUE and whose data parameter length is between 502 and 1497 octets, inclusive.

9.3.10 Frame Type 33: BACnet Extended Data Not Expecting Reply

This COBS-encoded frame is used by master nodes to convey the data parameter of a DL_UNITDATA.request whose data_expecting_reply parameter is FALSE and whose data parameter length is between 502 and 1497 octets, inclusive.

[Change **Clause 9.5.2**, p. 95]

9.5.2 Variables

A number of variables and timers are used in the descriptions that follow:

...

InputBuffer[]	An array of octets, used to store octets as they are received. InputBuffer is indexed from 0 to InputBufferSize-1. ~~The maximum size of a frame is 501 octets. A smaller value for InputBufferSize may be used by some implementations.~~
InputBufferSize	The number of elements in the array InputBuffer[]. *Routing nodes shall support the maximum NPDU size. Non-routing nodes may support a smaller value. Devices that support COBS-encoded frames shall include additional octets to account for encoding overhead. The overhead is calculated as the maximum supported NPDU size divided by 254 (any fractional part is rounded up to the nearest integer) plus five octets for the Encoded CRC-32K. For example, the InputBufferSize required for a device that supports the maximum BACnet NPDU size is 1497 + 6 + 5 = 1508.*
GoodHeader	*A Boolean flag set to TRUE or FALSE by the CheckHeader procedure (see 9.5.8).*
CRC32K	*Used by devices that support COBS-encoded frames to accumulate the CRC-32K on the Encoded Data field.*
DecodeIndex	*Used by devices that support COBS-encoded frames as an index into InputBuffer during decoding.*
DecodeCount	*Used by devices that support COBS-encoded frames to count down the number of octets remaining to be decoded in the InputBuffer.*
CodeOctet	*Used by devices that support COBS-encoded frames to count down the number of octets remaining to be decoded in the current code block.*
LastCode	*Used by devices that support COBS-encoded frames to buffer the value (1 - 255) of the code octet of the current code block.*
DataOctet	*Used by devices that support COBS-encoded frames to temporarily store the data octet being decoded in the current code block.*

...

[Change **Clause 9.5.3**, p. 96]

9.5.3 Parameters

Parameter values used in the description:

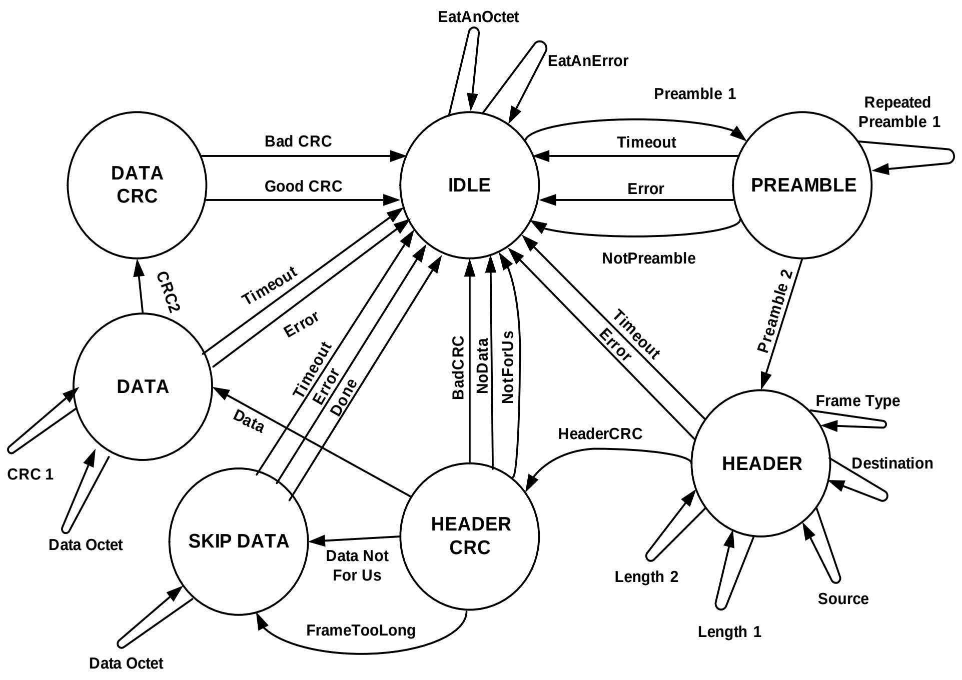
...

$N_{min\_COBS\_type}$	*The first COBS-encoded Frame Type value: 32.*

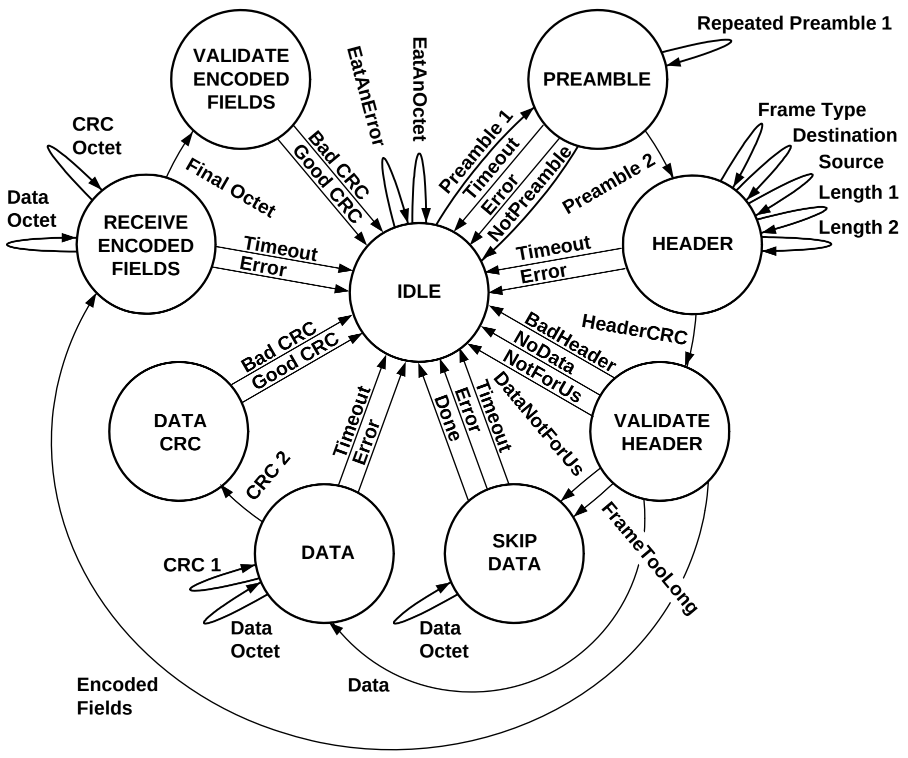
$N_{max\_COBS\_type}$	*The last COBS-encoded Frame Type value: 127.*
$N_{min\_COBS\_length}$	*The minimum valid Length value of any COBS-encoded frame: 5. The theoretical minimum Length is calculated as follows: COBS-encoded frames must contain at least one data octet. The minimum COBS encoding overhead for the Encoded Data field is one octet. The size of the Encoded CRC-32K field is always five octets. Adding the lengths of these fields and subtracting two (for backward compatibility) results in a minimum Length value of five (1 + 1 + 5 - 2). In practice, the minimum Length value is determined by the network-layer client and is likely to be larger (e.g., for BACnet the minimum Length is 502 + 1 + 3 = 506.*
$N_{max\_COBS\_length}$	*The maximum valid Length value of any COBS-encoded frame: 2043. The theoretical maximum Length is calculated as follows: the largest data parameter that any future network client may specify is 2032 octets (this is near the limit of the CRC-32K's maximum error-detection capability). The worst-case COBS encoding overhead for the Encoded Data field would be 2032 / 254 = 8 octets. Adding in the size adjustment for the Encoded CRC-32K results in a maximum Length value of 2032 + 8 + 3 = 2043. In practice, the maximum Length value is determined by the network-layer client and is likely to be smaller (e.g., for BACnet the maximum Length is 1497 + 6 + 3 = 1506.*

[Change **Figure 9-3**, p. 97]

[Existing Figure]



[Revised Figure]



[Change **Clause 9.5.4.3**, p. 98]

...

HeaderCRC

If ReceiveError is FALSE and DataAvailable is TRUE and Index is 5,

then set DataAvailable to FALSE; set SilenceTimer to zero; increment EventCount; accumulate the contents of DataRegister into HeaderCRC; *call the CheckHeader procedure defined in Clause 9.5.8;* and enter the VALIDATE_HEADER state.

[Change **Clause 9.5.4.4**, p. 99]

9.5.4.4 VALIDATE_HEADER

In the VALIDATE_HEADER state, the node validates the CRC on the fixed message header *and tests for illegal values in other header fields*.

~~BadCRC~~*BadHeader*

If ~~the value of HeaderCRC is not X '55'~~ *GoodHeader is FALSE,*

then set ReceivedInvalidFrame to TRUE to indicate that an error has occurred during the reception of a frame, and enter the IDLE state to wait for the start of the next frame.

NotForUs

If ~~the value of the HeaderCRC is X '55'~~ *GoodHeader is TRUE* and DataLength is zero and the value of DestinationAddress is not equal to either TS (this station) or 255 (broadcast),

then enter the IDLE state to wait for the start of the next frame.

DataNotForUs

If ~~the value of the HeaderCRC is X '55'~~ *GoodHeader is TRUE* and DataLength is not zero and the value of DestinationAddress is not equal to either TS (this station) or 255 (broadcast),

then set Index to zero and enter the SKIP_DATA state to consume the ~~data~~ *Data* and ~~data~~ *Data* CRC *or Encoded Data and Encoded CRC-32K* portions of the frame.

FrameTooLong

If ~~the value of the HeaderCRC is X '55'~~ *GoodHeader is TRUE* and the value of DestinationAddress is equal to either TS (this station) or 255 (broadcast) and DataLength is greater than InputBufferSize,

then set ReceivedInvalidFrame to TRUE to indicate that a frame with an illegal or unacceptable ~~data length~~ *DataLength* has been received, set Index to zero, and enter the SKIP_DATA state to consume the ~~data~~ *Data* and ~~data~~ *Data* CRC *or Encoded Data and Encoded CRC-32K* portions of the frame.

NoData

If ~~the value of the HeaderCRC is X '55'~~ *GoodHeader is TRUE* and the value of DestinationAddress is equal to either TS (this station) or 255 (broadcast) and DataLength is zero,

then set ReceivedValidFrame to TRUE to indicate that a frame with no data has been received, and enter the IDLE state to wait for the start of the next frame.

Data

If ~~the value of the HeaderCRC is X '55'~~ *GoodHeader is TRUE* and the value of DestinationAddress is equal to either TS (this station) or 255 (broadcast) and DataLength is not zero and DataLength is less than or equal to InputBufferSize *and either this device does not support COBS-encoded frames or FrameType indicates a non-encoded frame,*

then set Index to zero; set DataCRC to X'FFFF'; and enter the DATA state to receive the data portion of the frame.

EncodedFields

If GoodHeader is TRUE and the value of DestinationAddress is equal to either TS (this station) or 255 (broadcast) and DataLength is less than or equal to InputBufferSize and this device supports COBS-encoded frames and FrameType indicates a COBS-encoded frame,

then set Index to zero; set CRC32K to X'FFFFFFFF'; and enter the RECEIVE_ENCODED_FIELDS state to receive the Encoded Data and Encoded CRC-32K fields of the frame.

[Insert new **Clauses 9.5.4.8** and **9.5.4.9**, p. 101]

9.5.4.8 RECEIVE_ENCODED_FIELDS

In the RECEIVE_ENCODED_FIELDS state, the node waits for and decodes the Encoded Data and Encoded CRC-32K fields of a frame according to the procedure described in 9.10.3. If the device does not support COBS-encoded frames, this state is unreachable and should not be implemented.

Timeout

If SilenceTimer is greater than $\mathbf{T_{frame\_abort}}$,

then set ReceivedInvalidFrame to TRUE to indicate that an error has occurred during the reception of a frame, and enter the IDLE state to wait for the start of the next frame.

Error

If ReceiveError is TRUE,

then set ReceiveError to FALSE; set SilenceTimer to zero; set ReceivedInvalidFrame to TRUE to indicate that an error has occurred during the reception of a frame; and enter the IDLE state to wait for the start of the next frame.

DataOctet

If ReceiveError is FALSE and DataAvailable is TRUE and Index is less than DataLength minus 3,

then set DataAvailable to FALSE; set SilenceTimer to zero; accumulate the contents of DataRegister into CRC-32K; save the contents of DataRegister at InputBuffer[Index]; increment Index by 1; and enter the RECEIVE_ENCODED_FIELDS state.

CRCOctet

If ReceiveError is FALSE and DataAvailable is TRUE and Index is greater than or equal to DataLength minus 3 and Index is less than DataLength plus 1,

then set DataAvailable to FALSE; set SilenceTimer to zero; save the contents of DataRegister at InputBuffer[Index]; increment Index by 1; and enter the RECEIVE_ENCODED_FIELDS state.

FinalOctet

If ReceiveError is FALSE and DataAvailable is TRUE and Index is equal to DataLength plus 1,

then set DataAvailable to FALSE; set SilenceTimer to zero; save the contents of DataRegister at InputBuffer[Index]; decode the Encoded Data and Encoded CRC-32K fields according to Clause 9.10.3; and enter the VALIDATE_ENCODED_FIELDS state.

9.5.4.9 VALIDATE_ENCODED_FIELDS

In the VALIDATE_ENCODED_FIELDS state, the node validates the received CRC-32K of the frame. If the device does not support COBS-encoded frames, this state is unreachable and should not be implemented.

BadCRC

If the value of CRC32K is not X'0843323B',

then set ReceivedInvalidFrame to TRUE to indicate that an error has occurred during the reception of a frame, and enter the IDLE state to wait for the start of the next frame.

GoodCRC

If the value of CRC32K is X'0843323B',

then set ReceivedValidFrame to TRUE to indicate the complete reception of a valid frame, and enter the IDLE state to wait for the start of the next frame.

[Change **Clause 9.5.5**, p. 101]

9.5.5 The SendFrame Procedure

The transmission of an MS/TP frame proceeds as follows:

Procedure SendFrame

If Frame Type is less then $N_{min\_COBS\_type}$ or Frame Type is greater than $N_{max\_COBS\_type}$,

then call SendNonEncodedFrame,

else call SendCOBS_EncodedFrame.

9.5.5.1 SendNonEncodedFrame Procedure for Non-Encoded Frame Types

(a) If SilenceTimer is less than $\mathbf{T}_{\mathbf{turnaround}}$, wait ($\mathbf{T}_{\mathbf{turnaround}}$ - SilenceTimer).

...

(k) Disable the transmit line driver *and enable the receiver*.

...

[Insert new **Clause 9.5.5.2**, p. 102]

9.5.5.2 SendCOBS_EncodedFrame Procedure for COBS-Encoded Frame Types

This procedure shall be implemented by devices that support COBS-encoded Frame Types.

(a) COBS-encode the NPDU according to Clause 9.10.2 to generate the Encoded Data field. Set DataLength equal to the length of the Encoded Data field in octets, plus three.

(b) If SilenceTimer is less than $\mathbf{T_{turnaround}}$, wait ($\mathbf{T_{turnaround}}$ - SilenceTimer).

(c) Disable the receiver and enable the transmit line driver.

(d) Transmit the preamble octets X'55', X'FF'. As each octet is transmitted, set SilenceTimer to zero.

(e) Initialize HeaderCRC to X'FF'.

(f) Transmit the Frame Type, Destination Address, Source Address, and Data Length octets. Accumulate each octet into HeaderCRC. As each octet is transmitted, set SilenceTimer to zero.

(g) Transmit the ones-complement of HeaderCRC. Set SilenceTimer to zero.

(h) Initialize CRC32K to X'FFFFFFFF'.

(i) Transmit the Encoded Data octets. Accumulate each octet into CRC32K. As each octet is transmitted, set SilenceTimer to zero.

(j) Compute the ones-complement of CRC32K and order it least significant octet first. COBS-encode the four octets according to Clause 9.10.2 to generate the five-octet Encoded CRC-32K field.

(k) Transmit the Encoded CRC-32K octets. As each octet is transmitted, set SilenceTimer to zero.

(l) Wait until the final stop bit of the last Encoded CRC-32K octet has been transmitted but not more than $\mathbf{T_{postdrive}}$.

(m) Disable the transmit line driver and enable the receiver.

(n) Return.

[Change **Clause 9.5.6.2**, p. 104]

9.5.6.2 IDLE

In the IDLE state, the node waits for a frame.
…
ReceivedDataNoReply

If ReceivedValidFrame is TRUE and DestinationAddress is equal to either TS (this station) or 255 (broadcast) and FrameType is equal to BACnet Data Not Expecting Reply, Test_Response, *BACnet Extended Data Not Expecting Reply,* or a ~~proprietary type~~ *FrameType* known to this node that does not expect a reply,

then indicate successful reception to the higher layers; set ReceivedValidFrame to FALSE; and enter the IDLE state.

ReceivedDataNeedingReply

If ReceivedValidFrame is TRUE and DestinationAddress is equal to TS (this station) and FrameType is equal to BACnet Data Expecting Reply, ~~Test Request~~*Test_Request*, *BACnet Extended Data Expecting Reply,* or a ~~proprietary type~~ *FrameType* known to this node that expects a reply,

then indicate successful reception to the higher layers (management entity in the case of Test_Request); set ReceivedValidFrame to FALSE; and enter the ANSWER_DATA_REQUEST state.

BroadcastDataNeedingReply

If ReceivedValidFrame is TRUE and DestinationAddress is equal to 255 (broadcast) and FrameType is equal to *either* BACnet Data Expecting Reply *or BACnet Extended Data Expecting Reply,* then indicate successful reception to the higher layers; set ReceivedValidFrame to FALSE; and enter the IDLE state to wait for the next frame.

[Change **Clause 9.5.6.3**, p. 105]

9.5.6.3 USE_TOKEN

In the USE_TOKEN state, the node is allowed to send one or more data frames. These may be BACnet Data frames or *other standard or* proprietary frames.

NothingToSend

If there is no data frame awaiting transmission,

then set FrameCount to $\mathbf{N}_{\mathbf{max\_info\_frames}}$ and enter the DONE_WITH_TOKEN state.

SendNoWait

If the next frame awaiting transmission is of type Test_Response, BACnet Data Not Expecting Reply, *BACnet Extended Data Not Expecting Reply,* a ~~proprietary type~~ *FrameType known to this node* that does not expect a reply, ~~or a frame of type Data Expecting Reply with a DestinationAddress that is equal to 255 (broadcast)~~ *or has a DestinationAddress that is equal to 255 (broadcast) and a FrameType equal to either BACnet Data Expecting Reply or BACnet Extended Data Expecting Reply,*

then call SendFrame to transmit the frame; increment FrameCount; and enter the DONE_WITH_TOKEN state.

SendAndWait

If the next frame awaiting transmission is of type Test_Request, ~~BACnet Data Expecting Reply,~~ a ~~proprietary type~~ *FrameType known to this node* that expects a reply, ~~or a frame of type Data Expecting Reply with a DestinationAddress that is not equal to 255 (broadcast)~~ *or has a DestinationAddress that is not equal to 255 (broadcast) and a FrameType equal to either BACnet Data Expecting Reply or BACnet Extended Data Expecting Reply,*

then call SendFrame to transmit the data frame; increment FrameCount; and enter the WAIT_FOR_REPLY state.

[Change **Clause 9.5.6.4**, p. 105]

9.5.6.4 WAIT_FOR_REPLY

In the WAIT_FOR_REPLY state, the node waits for a reply from another node.

ReplyTimeout

If SilenceTimer is greater than or equal to $\mathbf{T}_{\mathbf{reply\_timeout}}$,

then assume that the request has failed. Set FrameCount to $\mathbf{N}_{\mathbf{max\_info\_frames}}$ and enter the DONE_WITH_TOKEN state. Any retry of the data frame shall await the next entry to the USE_TOKEN state. (Because of the length of the timeout, this transition will cause the token to be passed regardless of the initial value of FrameCount.)

InvalidFrame

If SilenceTimer is less than $\mathbf{T}_{\mathbf{reply\_timeout}}$ and ReceivedInvalidFrame is TRUE,

then there was an error in frame reception. Set ReceivedInvalidFrame to FALSE and enter the DONE_WITH_TOKEN state.

ReceivedReply

If SilenceTimer is less than $\mathbf{T}_{\mathbf{reply\_timeout}}$ and ReceivedValidFrame is TRUE and DestinationAddress is equal to TS (this station) and FrameType is equal to Test_Response, BACnet Data Not Expecting Reply, *BACnet Extended Data Not Expecting Reply,* or a ~~proprietary type~~ *FrameType known to this node* that indicates a reply,

then indicate successful reception to the higher layers; set ReceivedValidFrame to FALSE; and enter the DONE_WITH_TOKEN state.

ReceivedPostpone

If SilenceTimer is less than $\mathbf{T}_{\mathbf{reply\_timeout}}$ and ReceivedValidFrame is TRUE and DestinationAddress is equal to TS (this station) and FrameType is equal to Reply Postponed,

then the reply to the message has been postponed until a later time. Set ReceivedValidFrame to FALSE and enter the DONE_WITH_TOKEN state.

ReceivedUnexpectedFrame

If SilenceTimer is less than $\mathbf{T}_{\mathbf{reply\_timeout}}$ and ReceivedValidFrame is TRUE and either

(a) DestinationAddress is not equal to TS (the expected reply should not be broadcast) or

(b) FrameType has a value other than Test_Response, BACnet Data Not Expecting Reply, *BACnet Extended Data Not Expecting Reply,* or ~~proprietary reply frame~~ *a FrameType known to this node that indicates a reply,*

then an unexpected frame was received. This may indicate the presence of multiple tokens. Set ReceivedValidFrame to FALSE, and enter the IDLE state to synchronize with the network. This action drops the token.

…

[Change **Clause 9.5.6.9**, p. 109]

9.5.6.9 ANSWER_DATA_REQUEST

The ANSWER_DATA_REQUEST state is entered when a BACnet Data Expecting Reply, *BACnet Extended Data Expecting Reply,* a Test_Request, or a ~~proprietary frame~~ *FrameType known to this node* that expects a reply is received.

Reply

If a reply is available from the higher layers within $\mathbf{T}_{\mathbf{reply\_delay}}$ after the reception of the final octet of the requesting frame (the mechanism used to determine this is a local matter),

then call SendFrame to transmit the reply frame and enter the IDLE state to wait for the next frame.

…

[Change **Clause 9.5.7.2**, p. 110]

9.5.7.2 IDLE

In the IDLE state, the node waits for a frame.

…

ReceivedUnwantedFrame

If ReceivedValidFrame is TRUE and either

(a) DestinationAddress is not equal to either TS (this station) or 255 (broadcast) or

(b) DestinationAddress is equal to 255 (broadcast) and FrameType has a value of BACnet Data Expecting Reply, ~~Test Request~~*Test_Request*, or a ~~proprietary type~~ *FrameType* known to this node that expects a reply (such frames may not be broadcast) or

(c) FrameType has a value of Token, Poll For Master, Reply To Poll For Master, Reply Postponed, or a ~~standard or proprietary frame type~~ *FrameType* not known to this node,

then an unexpected or unwanted frame was received. Set ReceivedValidFrame to FALSE, and enter the IDLE state to wait for the next frame.

ReceivedDataNoReply

If ReceivedValidFrame is TRUE and DestinationAddress is equal to either TS (this station) or 255 (broadcast) and FrameType is equal to BACnet Data Not Expecting Reply, Test_Response, or a ~~proprietary type~~ *FrameType* known to this node that does not expect a reply,

then indicate successful reception to the higher layers, set ReceivedValidFrame to FALSE, and enter the IDLE state.

ReceivedDataNeedingReply

If ReceivedValidFrame is TRUE and DestinationAddress is equal to TS (this station) and FrameType is equal to BACnet Data Expecting Reply, ~~Test Request~~*Test_Request*, or a ~~proprietary type~~ *FrameType* known to this node that expects a reply,

then indicate successful reception to the higher layers (management entity in the case of Test_Request), set ReceivedValidFrame to FALSE, and enter the ANSWER_DATA_REQUEST state.

[Change **Clause 9.5.7.3**, p. 110]

9.5.7.3 ANSWER_DATA_REQUEST

The ANSWER_DATA_REQUEST state is entered when a BACnet Data Expecting Reply, a Test_Request, or a ~~proprietary frame~~ *FrameType known to this node* that expects a reply is received.

Reply

If a reply is available from the higher layers within $\mathbf{T_{reply\_delay}}$ after the reception of the final octet of the requesting frame (the mechanism used to determine this is a local matter),

then call SendFrame to transmit the reply frame, and enter the IDLE state to wait for the next frame.

…

[Add new **Clause 9.5.8**, p. 111]

9.5.8 The CheckHeader Procedure

This procedure checks for invalid header fields and sets the value of GoodHeader accordingly.

If the value of HeaderCRC is not X '55',
or the value of SourceAddress is 255 (broadcast) ,
or the value of FrameType is less than $N_{min\_COBS\_type}$ and DataLength is greater than 501,
or the value of FrameType is greater than $N_{max\_COBS\_type}$ and DataLength is greater than 501,
or the value of FrameType is greater than or equal to $N_{min\_COBS\_type}$ and
    the value of FrameType is less than or equal to $N_{max\_COBS\_type}$ and

DataLength is less than $N_{min\_COBS\_length}$,
or the value of FrameType is greater than or equal to $N_{min\_COBS\_type}$ and
the value of FrameType is less than or equal to $N_{max\_COBS\_type}$ and
DataLength is greater than $N_{max\_COBS\_length}$,

then set GoodHeader to FALSE, else set GoodHeader to TRUE.

[Change **Clause 9.6**, p. 111]

9.6 Cyclic Redundancy Check (CRC)

…

ISO 8802-3, ARCNET, and many other standard and proprietary communications systems use more robust CRCs. Mathematically, a CRC is the remainder that results when a data stream (such as a frame) taken as a binary number is divided modulo two by a generator polynomial. The proof of the error-detecting properties of the CRC and the selection of appropriate polynomials are beyond the scope of this document. *Annex G describes the implementation of the CRC algorithms in software.*

9.6.1 Frame Header CRC

The MS/TP frame header uses the polynomial

…

9.6.2 Data CRC

The MS/TP ~~data~~ *Data* CRC uses the ~~CRC-CCITT~~ *CRC-16-CCITT* polynomial

…

9.6.3 CRC-32K

COBS-encoded MS/TP frames use the CRC-32K (Koopman) polynomial

$$\begin{aligned} G(X) &= X^{32} + X^{30} + X^{29} + X^{28} + X^{26} + X^{20} + X^{19} + X^{17} + X^{16} + X^{15} + X^{11} + X^{10} + X^{7} + X^{6} + X^{4} + X^{2} + X + 1 \\ &= (X + 1)(X^{3} + X^{2} + 1)(X^{28} + X^{22} + X^{20} + X^{19} + X^{16} + X^{14} + X^{12} + X^{9} + X^{8} + X^{6} + 1) \end{aligned}$$

In operation, at the transmitter, the initial content of the CRC-32K register of the device computing the remainder of the division is preset to all ones. The register is then modified by division by the generator polynomial G(x) of the Encoded Data field (see Clause 9.10.2). The ones-complement of the resulting remainder is ordered least-significant octet first and then COBS-encoded to generate the five-octet Encoded CRC-32K field.

At the receiver, the initial content of the CRC-32K register of the device computing the remainder of the division is preset to all ones. The register is then modified by division by the generator polynomial G(x) of the Encoded Data field octets of the incoming message before they are decoded. The Encoded CRC-32K field is then decoded (see 9.10.3) and the register is modified by division by the generator polynomial G(x) of the four decoded octets. In the absence of transmission errors, the resultant remainder will be

0000 1000 0100 0011 0011 0010 0011 1011 (x^0 through x^{31}, respectively).

NOTE: The initialization of the CRC-32K register to all ones and the complementing of the register before transmission prevent the CRC-32K from having a value of zero if the covered field is all zeros.

~~Annex G describes the implementation of the CRC algorithms in software.~~

[Change **Clause 9.7.1**, p. 112]

9.7.1 Routing of *BACnet* Messages from MS/TP

When a *BACnet* network entity with routing capability receives from a directly connected MS/TP data link a NPDU whose 'data_expecting_reply' parameter is TRUE and the NPDU is to be routed to another *BACnet* network according to the procedures of Clause 6, the network entity shall direct the MS/TP data link to transmit a Reply Postponed frame before attempting to route the NPDU. This allows the routing node to leave the ANSWER_DATA_REQUEST state and the sending node to leave the WAIT_FOR_REPLY state before the potentially lengthy process of routing the NPDU is begun.

BACnet routers directly connected to MS/TP links must be updated to support COBS-encoded BACnet MS/TP frame types (i.e., BACnet Extended Data Expecting Reply and BACnet Extended Data Not Expecting Reply) before any non-routing devices installed on the link can utilize COBS-encoded frames. This does not, however, ensure that all links on the path from a given source to a given destination network have been updated. Before sending large MS/TP frames, a sending device should determine whether the given destination network is reachable using large MS/TP frames. A procedure by which this can be achieved is described in Clause 19.X.

[Change **Clause 9.7.2**, p. 112]

9.7.2 Routing of *BACnet* Messages to MS/TP

When a *BACnet* network entity issues a ~~DL_UNITDATA.request~~ *DL-UNITDATA.request* to a directly connected MS/TP data link, it shall set the 'data_expecting_reply' parameter of the DL-UNITDATA.request equal to the value of the 'data_expecting_reply' parameter of the network protocol control information of the NPDU, which is transferred in the 'data' parameter of the request.

…

[Insert new **Clause 9.10**, p. 113]

9.10 COBS (Consistent Overhead Byte Stuffing) Encoding

The original MS/TP specification did not prevent preamble sequences from appearing in the Data or Data CRC fields. This resulted in lost synchronization or dropped frames under certain circumstances. The Receive Frame state machine was later revised to mitigate this problem, but many deployed MS/TP devices do not have the update.

In order to decrease the likelihood of compatibility problems in environments with a mix of new and legacy devices, all MS/TP devices that support COBS-encoded frames shall COBS-encode the Encoded Data and Encoded CRC-32K fields before transmission and decode these fields upon reception. The result of this encoding is to remove X'55' octets from these portions of the frame, therefore preamble sequences cannot appear "on the wire" except where intended to indicate the beginning of a frame.

9.10.1 COBS Description

This Clause provides a basic description of the COBS encoding. A complete description may be found in "Consistent Overhead Byte Stuffing" by S. Cheshire and M. Baker. A reference to this paper appears in Clause 25.

COBS performs a reversible transformation on a data field to eliminate a single octet value (e.g. X'55') from it. Once eliminated from the data, that octet value may then be safely used in the framing marker (preamble). The most efficient way to accomplish this is to first eliminate all X'00' (zero) octets as described below, then exclusive OR (XOR) the output with the octet selected for removal.

COBS first takes its input data and logically appends a single zero octet. It then locates all the zero octets in the frame (including the appended one) and then divides the frame at these boundaries into one or more zero-terminated chunks. Every zero-terminated chunk contains exactly one zero octet, and that zero is always the last octet of the chunk. A chunk may be as short as one octet (i.e., a chunk containing a single zero octet) or as long as the entire input sequence (i.e., no zeros in the data except the appended one).

COBS encodes each zero-terminated chunk using one or more variable length COBS code blocks. Chunks of 254 octets or less (counting the terminating zero) are encoded as single COBS code blocks. Chunks longer than 254 octets are encoded using multiple code blocks, as described later in this Clause. After a packet's constituent chunks have all been encoded using COBS code blocks and concatenated, the resulting aggregate block of data is completely free of zero octets so zeros can then be placed around the encoded packet to mark clearly where it begins and ends.

A COBS code block consists of a single code octet, followed by zero or more data octets. The number of data octets is represented by the code octet. For codes X'01' to X'FE', the meaning of each code block is that it represents the sequence of data octets contained within the code block, followed by an implicit zero octet. The zero octet is implicit, meaning it is not actually contained within the sequence of data octets in the code block, but it is counted.

These code blocks encode data without adding any overhead. Each code block begins with one code octet followed by *n* data octets, and represents *n* data octets followed by one trailing zero octet. Thus the code block adds no overhead to the data: a chunk (*n*+1) octets long is encoded using a code block (1+*n*) octets long. These basic codes are suitable for encoding zero-terminated chunks up to 254 octets in length, but some of the zero-terminated chunks that make up a packet may be longer than that. To encode these chunks, code X'FF' is defined slightly differently. Code X'FF' represents the sequence of 254 data octets contained within the code block, without any implicit zero. This encoding is illustrated in Figure 9-6 below.

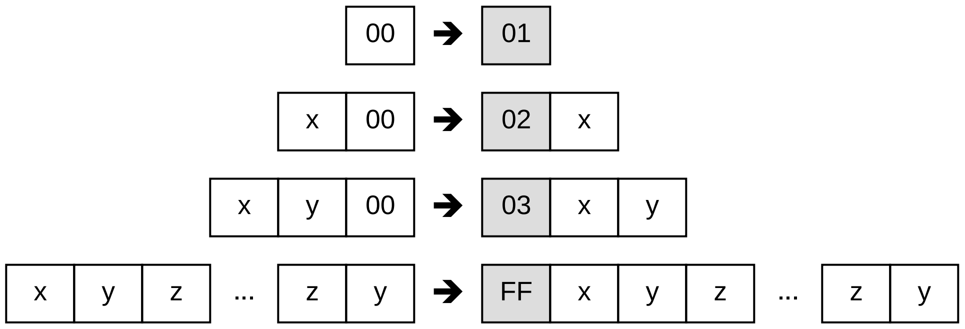


Figure 9-6. COBS Encoding of Chunks into Code Blocks

The two examples in Figure 9-7 show the effect of logically appending a zero (X'00') octet to the input data before encoding. For codes less than 255, a "phantom zero" after the data is used to differentiate between final chunks that end in X'00' and those that do not.

In the upper example, the final chunk of data is the string "Hello" without a terminating zero. A phantom zero is appended to the data and this results in consistent encoding of the final chunk. The phantom zero is discarded by the COBS decoder upon reception and the original data is restored in the receive buffer.

In the lower example, the final chunk of data is again the string "Hello" but this is terminated by a null in the data. Again, a phantom zero is appended to the data. Note the difference after encoding; the result is actually two code blocks transmitted. At the receiver, the phantom zero is discarded and the original data is restored, including the terminating null.

In the case of a final chunk that consists of exactly 254 non-zero octets, a phantom zero SHALL NOT be appended to the encoding since the meaning of code 255 is unambiguous: no implicit zero follows this data.

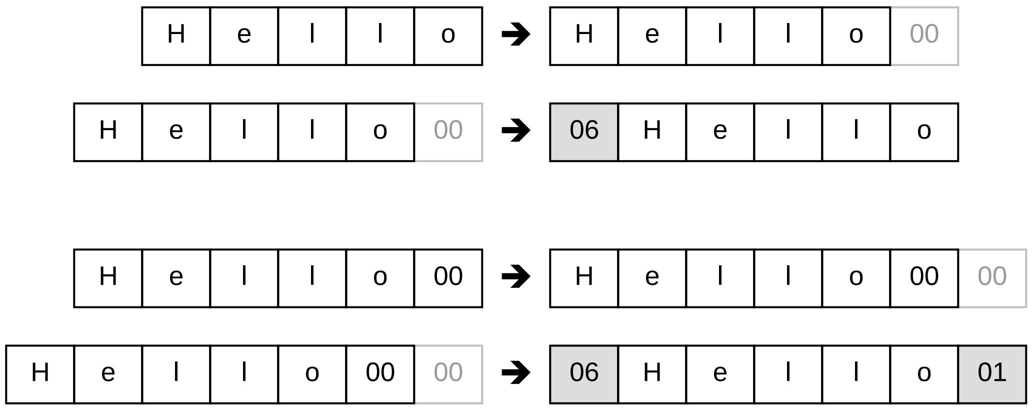

H e l l o → H e l l o 00
H e l l o 00 → 06 H e l l o
H e l l o 00 → H e l l o 00 00
H e l l o 00 00 → 06 H e l l o 01

Figure 9-7. Effect of Logically Appending X'00' to the Final Code Block Before Encoding

It can be seen from this example that COBS encoding always adds at least one octet of overhead. In the worst case, COBS encoding adds one octet of overhead for every 254 data octets (a maximum of six octets for a full NPDU, or less than 0.4%). Annex X describes the implementation of the cobs_encode() and cobs_decode() algorithms in software.

9.10.2 Preparing COBS-Encoded MS/TP Frames for Transmission

The COBS encoding described above eliminates all X'00' octets from the input stream and writes the encoded chunks to output. However, the actual goal is to eliminate MS/TP preamble sequences from the Encoded Data and Encoded CRC-32K fields, which can only be accomplished by eliminating either X'55' or X'FF' from these fields. X'FF' cannot be eliminated as this octet may be optionally appended to the end of any frame, leaving X'55' as the only option. The goal is accomplished by first running the encoding function over the input stream, then XOR'ing every octet in the output stream with X'55'. This modifies every octet in the output stream and transforms all occurrences of X'55' into X'00' (which is an acceptable value in the Encoded Data or Encoded CRC-32K fields).

The logical procedure for preparing COBS-encoded MS/TP frames for transmission is as follows:

(a) Pass the client data through the COBS encoder and save the returned length in DataLength. XOR each of the DataLength octets in the output buffer with X'55' to produce the Encoded Data field.

(b) Initialize the value of CRC32K to X'FFFFFFFF'.

(c) Pass the input stream (Encoded Data field) through CalcCRC32K() and accumulate the result in CRC32K. (It is recommended that this be done as each octet is transmitted to reduce latency.)

(d) Take the ones' complement of CRC32K and if necessary reorder it for transmission least significant octet first.

(e) Pass the modified CRC32K through the COBS encoder and XOR each of the five output octets with X'55' to produce the Encoded CRC-32K field.

(f) Add three to DataLength (the five octet length of Encoded CRC-32K field, minus two octets for backward compatibility with legacy MS/TP devices that implement the SKIP_DATA state in their Receive Frame State Machine). Use this modified value of DataLength as the Length field of the outgoing frame.

9.10.3 Decoding COBS-Encoded MS/TP Frames Upon Reception

The logical procedure for decoding COBS-encoded MS/TP frames upon reception is the inverse of encoding and restores the sender's original NPDU at the receiving node. This procedure assumes the Encoded Data and Encoded CRC-32K octets reside contiguously in InputBuffer in the order received and that the variables Index, CRC32K, DecodeIndex, and DecodeCount are initialized as shown below. Note that this procedure shows CRC32K being calculated over the Extended Data octets before decoding. However, it is recommended this be performed as the octets are received to reduce latency (see Clause 9.5.4.8).

```
Index = 0;
CRC32K = X'FFFFFFFF';
DecodeIndex = 0;
DecodeCount = DataLength - 3;
```

(a) (Process a code octet in the Encoded Data field)

Set the contents of DataOctet to the value of InputBuffer[DecodeIndex]; accumulate the contents of DataOctet into CRC32K; and set CodeOctet equal to the value of DataOctet XOR'd with X'55'.

If CodeOctet is equal to zero or greater than DecodeCount,

then set the value of CRC32K to X'FFFFFFFF' to indicate that an error has occurred during the reception of a frame and return;

else increment DecodeIndex by 1; decrement DecodeCount by 1; set LastCode to the value of CodeOctet; and decrement CodeOctet by 1.

If CodeOctet is equal to zero then go to step (c) else go to step (b).

(b) (Process a data octet in the Encoded Data field)

Set the contents of DataOctet to the value of InputBuffer[DecodeIndex]; accumulate the contents of DataOctet into CRC32K; set DataOctet equal to the contents of DataOctet XOR'd with X'55'; save the contents of DataOctet at InputBuffer[Index]; increment Index by 1; increment DecodeIndex by 1; decrement DecodeCount by 1; and decrement CodeOctet by 1.

If CodeOctet is equal to zero then go to step (c) else go to step (b).

(c) If DecodeCount is greater than zero and LastCode is not equal to 255,

then save the value zero at InputBuffer[Index]; and increment Index by 1.

If DecodeCount is greater than zero then go to step (a);

else set DataLength to Index; set the value of DecodeCount to five; and go to step (d).

(d) (Process a code octet in the Encoded CRC-32K field)

Set the contents of DataOctet to the value of InputBuffer[DecodeIndex]; and set CodeOctet equal to the contents of DataOctet XOR'd with X'55'.

If CodeOctet is equal to zero or greater than DecodeCount,

then set the value of CRC32K to X'FFFFFFFF' to indicate that an error has occurred during the reception of a frame and return;

else increment DecodeIndex by 1; decrement DecodeCount by 1; and decrement CodeOctet by 1.

If CodeOctet is equal to zero then go to step (f) else go to step (e).

(e) (Process a data octet in the Encoded CRC-32K field, i.e., CRC1, CRC2, CRC3, or CRC4)

Set the contents of DataOctet to the value of InputBuffer[DecodeIndex] XOR'd with X'55'; accumulate the contents of DataOctet into CRC32K; increment DecodeIndex by 1; decrement DecodeCount by 1; and decrement CodeOctet by 1.

If CodeOctet is equal to zero then go to step (f) else go to step (e).

(f) If DecodeCount is greater than zero,

then accumulate the value zero into CRC32K; and go to step (d);

else return.

[Add new entries to **Clause 25**, p. 772]

25 REFERENCES

...

IEEE/ACM TRANSACTIONS ON NETWORKING, VOL.7, NO.2, APRIL, 1999, *Consistent Overhead Byte Stuffing*, S. Cheshire and M. Baker.

...

IEEE/IFIP International Conference on Dependable Systems and Networks (DSN 2002), *32-Bit Cyclic Redundancy Codes for Internet Applications*, P. Koopman.

...

IEEE/IFIP International Conference on Dependable Systems and Networks (DSN 2004), *Cyclic Redundancy Code (CRC) Polynomial Selection for Embedded Networks*, P. Koopman and T. Chakravarty

...

IEEE Transactions on Dependable and Secure Computing, Vol. 6, No. 1, January – March 2009, *The Effectiveness of Checksums for Embedded Control Networks*, T. Maxino and P. Koopman

...

[Add new entries to **Clause 25**, **Sources for Reference Material** section, p. 773]

...

Cheshire: http://www.stuartcheshire.org/papers/COBSforToN.pdf

...

Koopman: http://www.ece.cmu.edu/~koopman/projects.html#crc

...

[Change **Annex A**, p. 775]

Data Link Layer Options

...

☐ MS/TP master (Clause 9), baud rate(s): ____________________

☐ *MS/TP master supports extended length BACnet frames (BACnet Extended Data Expecting Reply and BACnet Extended Data Not Expecting Reply).*

...

[Insert new **Clause G.3** and subclauses, p. 837]

G.3 Calculation of the Encoded CRC-32K

The equivalence of hardware and software methods for calculating CRCs is demonstrated by the examples shown previously in Clauses G.1 and G.2. There are numerous open source examples of CRC algorithms available, including "A Painless Guide to CRC Error Detection Algorithms." It describes a library that can implement arbitrary CRC systems based on input parameters such as CRC length, polynomial, initial value, shift direction (based on the transmission order of the underlying data link), etc.

G.3.1 Sample Implementation of the CRC-32K in C

An example C language implementation for the CRC-32K (Koopman) polynomial is shown below. Inputs are the octet to be processed and the accumulated CRC value. The function return value is the updated CRC. Note that Koopman expresses CRC polynomial representations in X^{32} ... X^1 order (X^0 is implied because it is always set to X'01'). The polynomial specified in Clause 9.6 is represented in his notation as X'BA0DC66B'. When this is reversed and represented in X^0 ... X^{31} order (X^{32} is implied because it is always set to X'01'), the equivalent polynomial representation becomes X'EB31D82E'.

```c
#include <stdint.h>

/* Accumulate "dataValue" into the CRC in "crcValue".
 * Return value is updated CRC.
 *
 * Assumes that "uint8_t" is equivalent to one octet.
 * Assumes that "uint32_t" is four octets.
 * The ^ operator means exclusive OR.
 */
uint32_t
CalcCRC32K(uint8_t dataValue, uint32_t crcValue)
{
  uint8_t data, b;
  uint32_t crc;

  data = dataValue;
  crc  = crcValue;

  for (b = 0; b < 8; b++) {
    if ((data & 1) ^ (crc & 1)) {
      crc >>= 1;
      crc ^= 0xEB31D82E;  /* CRC-32K polynomial, 1 + x**1 + ... + x**30 (+ x**32) */
    } else {
      crc >>= 1;
    }
    data >>= 1;
  }
  return crc;
}
```

In operation, the CRC-32K register C31-C0 is initialized to all ones. During transmission, the Encoded Data octets are passed through the calculation after encoding but before transmission. The ones' complement of the final CRC-32K register value is then ordered least-significant octet first (i.e., in right to left order) and COBS-encoded according to Clause 9.10.2. At the receiving end, the Encoded Data octets are passed through the calculation before decoding. Then, the received Encoded CRC-32K field is decoded according to Clause 9.10.3 and the resulting octets (the ones' complement of the sender's CRC-32K value) are passed through the calculation. If all the octets are received correctly, then the final value of the receiver's CRC-32K register (termed the "residue") will be the constant X'0843323B'. NOTE: the correct residue for a given polynomial may be determined by passing a string of zeros equal in the length to the size of the CRC register through the calculation.

```
| 0000 | 1000 | 0100 | 0011 | 0011 | 0010 | 0011 | 1011  |
|<- 0 -><- 8 -><- 4 -><- 3 -><- 3 -><- 2 -><- 3 -><- B ->|
```

As a simplified example of usage, consider a non-encoded data sequence X'01', X'22', X'30':

Description	Value	CRC-32K after octet is processed
		X'FFFFFFFF' (initial value)
first data octet	X'01'	X'56381747'
second data octet	X'22'	X'12557D20'
third data octet	X'30'	X'83DD5A41'
		(ones' complement is X'7C22A5BE')
CRC1 (least significant octet)	X'BE'	X'C0D1F128'
CRC2	X'A5'	X'1C708944'
CRC3	X'22'	X'7FC2C8BD'
CRC4 (most significant octet)	X'7C'	X'0843323B'

Thus, the sender would calculate the CRC-32K on the three octets X'01', X'22', X'30' to be X'83DD5A41'. The ones' complement of this is X'7C22A5BE', which is appended to the frame least significant octet first as X'BE', X'A5', X'22', X'7C'.

The receiver would calculate the CRC-32K on the seven octets X'01', X'22', X'30', X'BE', X'A5', X'22', and X'7C' to be X'0843323B', which is the expected result for a correctly received frame.

G.3.2 Sample Implementation of the CRC-32K in Assembly Language

One way to generate an efficient assembly language implementation of the CRC-32K function is to cross-compile the C example above for a given target processor, examine the assembly language listing, and hand optimize register usage, etc. An assembly language implementation of the CRC-32K for the 8-bit AVR[1] instruction set is presented below. The avr-gcc calling conventions are observed and only scratch registers available for use by the compiler are modified.

```
#include <stdint.h>

; uint32_t
; CalcCRC32K(uint8_t dataValue, uint32_t crcValue)
;
; call:
;
;   r24     dataValue
;   r23-r20 crcValue (LSB-MSB)
;
; return:
;
;   r25-r22 updated crcValue (LSB-MSB)
;
CalcCRC32K:
    ldi     r18, 0x01   ; r18 = constant ONE
    ldi     r19, 0x00   ; for (b = 0; ...

_for:
; Calculate and save result of ((data ^ crc) & 1)

    mov     r25, r24    ; data lsb
    eor     r25, r20    ; ^crc msb
    and     r25, r18

; crc >>= 1 in either case

    lsr     r23
```

[1] AVR is a registered trademark of Atmel Corporation.

```
    ror     r22
    ror     r21
    ror     r20

; If result of previous calculation was '1', accumulate feedback

    cp      r25, r18
    brne    _else

; crc ^= 0xEB31D82E; CRC-32K polynomial, 1 + x**1 + ... + x**30 (+ x**32)

    ldi     r25, 0xEB
    eor     r23, r25
    ldi     r25, 0x31
    eor     r22, r25
    ldi     r25, 0xD8
    eor     r21, r25
    ldi     r25, 0x2E
    eor     r20, r25

_else:
; data >>= 1;

    lsr     r24

    add     r19, r18    ; b++;
    cpi     r19, 0x08
    brlt    _for        ; b < 8;

    movw    r24,r22     ; copy 32-bit return value to r25-r22
    movw    r22,r20
    ret
```

G.3.3 Parallel Implementation of the CRC-32K

Other implementations of the CRC-32K algorithm are possible. It can be seen that the new CRC-32K value may be factored into the exclusive OR of two terms. One term is the most significant octet of the prior CRC-32K value. The other term is a function of the least significant octet of the prior CRC-32K exclusive OR'ed with the data value. A lookup table with 256 elements may be used to quickly determine the value of the second term, using the least significant octet of the prior CRC-32K exclusive OR'ed with the data value as an index. This term may then be exclusive OR'ed with the most significant octet of the prior CRC-32K value to form the new CRC-32K value. The contents of the table may be computed using an implementation similar to the one shown in Clause G.3.1. A sample implementation appears below.

```
#include <stdint.h>

void
CreateCRC32Table()
{
  uint16_t data;
  uint32_t crc;
  uint16_t b;

  printf( "static const uint32_t CRC32Table[256] = {" );
  for (data = 0; ; ) {
    if (data % 8 == 0)
      printf("\n");

    crc = data & 0xFF;
    for (b = 0; b < 8; b++) {
      if (crc & 1) {
          crc >>= 1;
          crc ^= 0xEB31D82E;
        } else {
          crc >>= 1;
        }
    }
    printf( "0x%08lX", crc );

    if (++data == 256)
      break;
    printf( ", " );
  }
  printf( "\n};\n" );
}


/* Update running "crcValue" with "dataValue"
 * The crcValue must be initialized to all ones.
 *
 * For transmission, the returned value must be complemented and then
 * sent low-order octet first (i.e., right to left).
 *
 * On reception, if Data ends with a correct CRC, the returned value
 * will be 0x0843323B.  (0000 1000 0100 0011 0011 0010 0011 1011)
 */
uint32_t
LookupCRC32K(uint8_t dataValue, uint32_t crcValue)
{
  crcValue = CRC32Table[(crcValue ^ dataValue) & 0xFF] ^ (crcValue >> 8);
  return (crcValue);
}
```

[Insert new **Annex X**]

ANNEX X - COBS (CONSISTENT OVERHEAD BYTE STUFFING) FUNCTIONS (INFORMATIVE)

(This Annex is not part of this standard but is included for informative purposes only.)

MS/TP did not originally specify a method for escaping preamble sequences that might appear in the Data or Data CRC fields. In certain cases, this might result in dropped frames due to loss of frame synchronization. The SKIP_DATA state was subsequently added to the Receive Frame state machine to mitigate this issue (see Clause 9.5.4.7), but did not resolve the problem for earlier implementations. Encoded MS/TP frames eliminate this problem by using Consistent Overhead Byte Stuffing to remove preamble sequences from the transmitted MSDU fields.

MS/TP implementations that support encoded frames use COBS to encode the Encoded Data and Encoded CRC-32K fields before transmission and decode these fields upon reception. COBS is a reversible run-length encoding method that by design removes X'00' octet values from its input. The encoding overhead is at least one octet per field and at most one octet per 254 octets of input, or less than 0.4% (as described in Clause 9.10). A selected octet value may be removed by first passing the input stream through the COBS encoder and then XOR'ing the output with the specified octet. In the case of MS/TP, the X'55' preamble octet is specified for removal.

X.1 Preparing a COBS-Encoded MS/TP Frame for Transmission

Encoding proceeds in two passes over the client data. First, the data is passed through the COBS encoder. Then each octet of the encoded output stream is XOR'd with the MS/TP preamble octet X'55' in order to remove all occurrences of this value from the resulting Encoded Data field (any X'55' octet is transformed into a X'00' octet, which is not a preamble value). Observing that the worst-case overhead for COBS-encoding is one octet per 254 octets of input (or six octets for a 1497 octet NPDU), it is possible to set the location of the output buffer just six octets before the location of the input buffer in memory and perform the encoding nearly in place.

The CRC-32K is then calculated over the Encoded Data field, prepared for transmission as described in Annex G.3, COBS-encoded (which adds one octet of overhead), and the resulting octets are each XOR'd with X'55' to produce the Encoded CRC-32K field. The combined length of the Encoded Data and Encoded CRC-32K, minus two octets for compatibility with legacy MS/TP devices, is placed in the MS/TP header Length field before transmission. Observing that the length of the Encoded CRC-32K field is always five octets, the Length field may be computed after encoding the data (as the length of the Encoded Data field plus three octets) and the CRC-32K field may then be computed and in parallel with data transmission.

As an example, the frame encoding of the null-terminated C string "Hello World\n" is shown below. Before encoding, the client data is:

```
0000: 48 65 6C 6C 6F 20 57 6F 72 6C 64 0A 00                  "Hello World.."
```

After COBS encoding (shown here for clarity), the output stream is:

```
0000: 0D 48 65 6C 6C 6F 20 57 6F 72 6C 64 0A 01               ".Hello World.."
```

Each octet in the COBS-encoded output stream is XOR'ed with X'55':

```
0000: 58 1D 30 39 39 3A 75 02 3A 27 39 31 5F 54               "X.099:u.:'91_T"
```

The length of the resulting Encoded Data field is 14 octets. After adding the constant five octet length of the Encoded CRC-32K field and subtracting two octets for compatibility with legacy MS/TP devices, the resulting Length field is 17 octets. At this point data transmission may begin and each octet in the Encoded Data field is accumulated in the CRC-32K before it is sent.

The resulting CRC-32K value (shown here for clarity) is:

```
000E: B3 8F 28 CA                                              "..(."
```

After taking the ones' complement and arranging in LSB order (see Annex G.3), the value becomes:

```
000E: 35 D7 70 4C                                              "5.pL"
```

After COBS-encoding and XOR'ing each octet in the output stream with X'55', the Encoded CRC-32K field is ready for transmission:

```
000E: 50 60 82 25 19                                           "P`.%."
```

An example C implementation that combines these two passes is shown below. This algorithm is presented as an example and is not intended to restrict the vendor's implementation of COBS. Since the worst-case overhead of the COBS encoding for a maximum size NPDU is six octets, the output pointer `to` in the example below may be set to `(uint8_t *)(from - 6)` and the encoding performed nearly in place.

```
#include <stddef.h>
#include <stdint.h>

#define CRC32K_INITIAL_VALUE (0xFFFFFFFF)
#define MSTP_PREAMBLE_X55 (0x55)

/*
 * Encodes 'length' octets of data located at 'from' and
 * writes one or more COBS code blocks at 'to', removing
 * any 0x55 octets that may present be in the encoded data.
 * Returns the length of the encoded data.
 */
```

```
size_t
cobs_encode (uint8_t *to, const uint8_t *from, size_t length, uint8_t mask)
{
  size_t code_index = 0;
  size_t read_index = 0;
  size_t write_index = 1;
  uint8_t code = 1;
  uint8_t data, last_code;

  while (read_index < length) {
    data = from[read_index++];
    /*
     * In the case of encountering a non-zero octet in the data,
     * simply copy input to output and increment the code octet.
     */
    if (data != 0) {
      to[write_index++] = data ^ mask;
      code++;
      if (code != 255)
        continue;
    }
    /*
     * In the case of encountering a zero in the data or having
     * copied the maximum number (254) of non-zero octets, store
     * the code octet and reset the encoder state variables.
     */
    last_code = code;
    to[code_index] = code ^ mask;
    code_index = write_index++;
    code = 1;
  }
  /*
   * If the last chunk contains exactly 254 non-zero octets, then
   * this exception is handled above (and returned length must be
   * adjusted).  Otherwise, encode the last chunk normally, as if
   * a "phantom zero" is appended to the data.
   */
  if ((last_code == 255) && (code == 1))
    write_index--;
  else
    to[code_index] = code ^ mask;

  return write_index;
}


/*
 * Encodes ’length’ octets of client data located at 'from' and writes
 * the COBS-encoded Encoded Data and Encoded CRC-32K fields at 'to'.
 * Returns the combined length of these encoded fields.
 */
size_t
frame_encode (uint8_t *to, const uint8_t *from, size_t length)
{
  size_t cobs_data_len, cobs_crc_len;
  uint32_t crc32K;
  int i;

  /*
   * Prepare the Encoded Data field for transmission.
   */
  cobs_data_len = cobs_encode(to, from, length, MSTP_PREAMBLE_X55);
  /*
```

```
   * Calculate CRC-32K over the Encoded Data field.
   * NOTE: May be done as each octet is transmitted to reduce latency.
   */
  crc32K = CRC32K_INITIAL_VALUE;
  for (i = 0; i < cobs_data_len; i++) {
    crc32K = CalcCRC32K(to[i], crc32K);   /* See Annex G.3.1 */
  }
  /*
   * Prepare the Encoded CRC-32K field for transmission.
   * NOTE: Assumes a little-endian CPU (otherwise order the
   * octets least-significant first before encoding).
   */
  crc32K = ~crc32K;
  cobs_crc_len = cobs_encode((uint8_t *)(to + cobs_data_len),
                             (const uint8_t *)&crc32K, sizeof(uint32_t),
                             MSTP_PREAMBLE_X55);
  /*
   * Return the combined length of the Encoded Data and Encoded CRC-32K
   * fields. NOTE: Subtract two before use as the MS/TP frame Length field.
   */
  return cobs_data_len + cobs_crc_len;
}
```

X.2 Decoding an Extended MS/TP Frame upon Reception

The frame_decode() function is the inverse of the frame_encode() function shown in the previous section. Note that the octets of the Encoded Data field are accumulated by the crc32K() function before decoding. The Encoded CRC-32K field is then decoded and the resulting CRC-32K octets are accumulated by the crc32K() function. If the result is the expected residue value, then the frame was received correctly.

```
#include <stddef.h>
#include <stdint.h>

#define CRC32K_INITIAL_VALUE (0xFFFFFFFF)
#define CRC32K_RESIDUE (0x0843323B)
#define MSTP_PREAMBLE_X55 (0x55)

/*
 * Decodes 'length' octets of data located at 'from' and
 * writes the original client data at 'to', restoring any
 * 'mask' octets that may present in the encoded data.
 * Returns the length of the encoded data or zero if error.
 */
size_t
cobs_decode (uint8_t *to, const uint8_t *from, size_t length, uint8_t mask)
{
  size_t read_index = 0;
  size_t write_index = 0;
  uint8_t code, last_code;

  while (read_index < length) {
    code = from[read_index] ^ mask;
    last_code = code;
    /*
     * Sanity check the encoding to prevent the while() loop below
     * from overrunning the output buffer.
     */
    if (read_index + code > length) {
      return 0;
    }
    read_index++;
    while (--code > 0) {
      to[write_index++] = from[read_index++] ^ mask;
    }
    /*
     * Restore the implicit zero at the end of each decoded block
     * except when it contains exactly 254 non-zero octets or the
     * end of data has been reached.
     */
    if ((last_code != 255) && (read_index < length)) {
      to[write_index++] = 0;
    }
  }
  return write_index;
}

#define ADJ_FOR_ENC_CRC (5)   /* Set to 3 if passing MS/TP Length field */
#define SIZEOF_ENC_CRC (5)

/*
 * Decodes Encoded Data and Encoded CRC-32K fields at 'from' and
 * writes the decoded client data at 'to'.  Assumes 'length' contains
 * the actual combined length of these fields in octets (that is, the
 * MS/TP header Length field plus two).
 * Returns length of decoded Data in octets or zero if error.
 * NOTE: Safe to call with ’output’ <= ’input’ (decodes in place).
 */
size_t
frame_decode (uint8_t *to, const uint8_t *from, size_t length)
{
  size_t data_len, crc_len;
  uint32_t crc32K;
  int i;

  /*
   * Calculate the CRC32K over the Encoded Data octets before decoding.
   * NOTE: Adjust 'length' by removing size of Encoded CRC-32K field.
   */
  data_len = length - ADJ_FOR_ENC_CRC;
  crc32K = CRC32K_INITIAL_VALUE;

  for (i = 0; i < data_len; i++) {
    crc32K = CalcCRC32K(from[i], crc32K);   /* See Annex G.3.1 */
  }
  data_len = cobs_decode(to, from, data_len, MSTP_PREAMBLE_X55);
  /*
   * Decode the Encoded CRC-32K field and append to data.
   */
  crc_len = cobs_decode((uint8_t *)(to + data_len),
                        (uint8_t *)(from + length - ADJ_FOR_ENC_CRC),
                        SIZEOF_ENC_CRC,
                        MSTP_PREAMBLE_X55);
  /*
   * Sanity check length of decoded CRC32K.
   */
  if (crc_len != sizeof(uint32_t)) {
    return 0;
  }
  /*
   * Verify CRC32K of incoming frame.
   */
  for (i = 0; i < crc_len; i++) {
    crc32K = CalcCRC32K((to + data_len)[i], crc32K);
  }
  if (crc32K == CRC32K_RESIDUE) {
    return data_len;
  } else {
    return 0;
  }
}
```

X.4 Example COBS-Encoded Frame - Who-Has Service

This section shows the header, NPDU, and APDU fields of a simple (Who-Has) BACnet request with a very long name consisting of 19 "A" characters, followed by 19 "B" characters, etc., through 19 "Z" characters. The hexadecimal dump of the corresponding COBS-encoded frame follows.

X'55FF'	MS/TP Preamble
X'21'	Frame Type BACnet Extended Data Not Expecting Reply = 33
X'FF'	Destination Address = 255 (broadcast)
X'01'	Source Address = 1
X'0200'	Length = 512
X'4E'	Header CRC = 78

X'01'	Version=1
X'20'	Control Bit 5: 1 = DNET, DLEN, and Hop Count Present
X'FFFF'	DNET = 65535 (Global broadcast)
X'00'	DLEN = 0 (A value of 0 indicates a broadcast on the destination network)
X'FF'	Hop Count = 255

X'10'	PDU Type=1 (Unconfirmed-Service-Request-PDU)	
X'07'	Service Choice=7 (Who-Has-Request)	
X'3D'	SD Context Tag 3 (Object Name, L>4)	
	X'FE'	Extended Length > 253
	X'01EF'	Extended Length = 495
	X'00'	ISO 10646 (UTF-8) Encoding
	X'41' ... '5A'	"A ... Z" (494 octets of Object Name data)

X'ACF483BD'	CRC-32K

```
0000  55 FF 21 FF 01 02 00 4E  50 54 75 AA AA 5D AA 45
0010  52 68 AB 54 BA AA 14 14  14 14 14 14 14 14 14 14
0020  14 14 14 14 14 14 14 14  14 17 17 17 17 17 17 17
0030  17 17 17 17 17 17 17 17  17 17 17 17 16 16 16 16
0040  16 16 16 16 16 16 16 16  16 16 16 16 16 16 16 11
0050  11 11 11 11 11 11 11 11  11 11 11 11 11 11 11 11
0060  11 11 10 10 10 10 10 10  10 10 10 10 10 10 10 10
0070  10 10 10 10 10 13 13 13  13 13 13 13 13 13 13 13
0080  13 13 13 13 13 13 13 13  12 12 12 12 12 12 12 12
0090  12 12 12 12 12 12 12 12  12 12 12 1D 1D 1D 1D 1D
00A0  1D 1D 1D 1D 1D 1D 1D 1D  1D 1D 1D 1D 1D 1D 1C 1C
00B0  1C 1C 1C 1C 1C 1C 1C 1C  1C 1C 1C 1C 1C 1C 1C 1C
00C0  1C 1F 1F 1F 1F 1F 1F 1F  1F 1F 1F 1F 1F 1F 1F 1F
00D0  1F 1F 1F 1F 1E 1E 1E 1E  1E 1E 1E 1E 1E 1E 1E 1E
00E0  1E 1E 1E 1E 1E 1E 1E 19  19 19 19 19 19 19 19 19
00F0  19 19 19 19 19 19 19 19  19 19 18 18 18 18 18 18
0100  18 18 18 18 18 18 18 18  18 18 18 18 18 1B 1B 1B
0110  1B 1B 1B 1B A4 1B 1B 1B  1B 1B 1B 1B 1B 1B 1B 1B
0120  1B 1A 1A 1A 1A 1A 1A 1A  1A 1A 1A 1A 1A 1A 1A 1A
0130  1A 1A 1A 1A 05 05 05 05  05 05 05 05 05 05 05 05
0140  05 05 05 05 05 05 05 04  04 04 04 04 04 04 04 04
0150  04 04 04 04 04 04 04 04  04 04 07 07 07 07 07 07
```

```
0160  07 07 07 07 07 07 07 07  07 07 07 07 07 06 06 06
0170  06 06 06 06 06 06 06 06  06 06 06 06 06 06 06 06
0180  01 01 01 01 01 01 01 01  01 01 01 01 01 01 01 01
0190  01 01 01 00 00 00 00 00  00 00 00 00 00 00 00 00
01A0  00 00 00 00 00 00 03 03  03 03 03 03 03 03 03 03
01B0  03 03 03 03 03 03 03 03  03 02 02 02 02 02 02 02
01C0  02 02 02 02 02 02 02 02  02 02 02 02 0D 0D 0D 0D
01D0  0D 0D 0D 0D 0D 0D 0D 0D  0D 0D 0D 0D 0D 0D 0D 0C
01E0  0C 0C 0C 0C 0C 0C 0C 0C  0C 0C 0C 0C 0C 0C 0C 0C
01F0  0C 0C 0F 0F 0F 0F 0F 0F  0F 0F 0F 0F 0F 0F 0F 0F
0200  0F 0F 0F 0F 0F 50 F9 A1  D6 E8
```

135-2012*an*-2 Add Procedure for Determining Maximum Conveyable APDU

Rationale

Add the description of a method for determining the maximum conveyable APDU size to a remote network.

[Insert new **Clause 19.X**, p. 613]

19.X Determining Maximum Conveyable APDU

This clause describes a method for determining the maximum conveyable APDU size to a remote network. This size may be affected by the existence of routers in the path to the remote network that are incapable of handling the maximum APDU size of the end nodes. If this method is used by numerous devices, it consumes considerable communications bandwidth and should be used sparingly since normal communications may be disrupted when used excessively. The method should only be performed when necessary and preferably after a randomly chosen delay in order to reduce the quantity of devices that may be using the method simultaneously. Once a node has determined the maximum conveyable APDU length for a remote network, it shall cache the determined value.

During this test, it is important that the client not generate any other messages to the remote network that are larger than the currently outstanding test message being sent. This will ensure that any Reject-Message-To-Network with a "Reject Reason" of 4 (MESSAGE_TOO_LONG) for the destination network applies to the outstanding test message.

A device on the remote network is selected which is capable of receiving a large APDU. It is a local matter as to how the client determines which device on the remote network to perform the test with.

The client first reads the remote peer's Max_APU_Length_Accepted property. This step verifies that the remote peer is online and reachable from the client. If no response is received from the remote peer, a different remote peer needs to be selected and the test restarted.

The client then generates a request of a size equal to the smaller of the local maximum APDU the client supports and the maximum APDU length supported by the remote peer. The preferred message to send is a ConfirmedPrivateTransfer-Request with a 'Vendor ID' of 0, a 'Service Number' of 0, and 'Service Parameters' containing a single application tagged OCTET_STRING with N data octets, where N is the number of octets required to generate an APDU of the desired size. This message is reserved by ASHRAE for this use and as such will not result in a change of state of the remote peer (it will have no net effect on the remote peer's operation).

Table 19.X: Calculating Test Message OCTET STRING Size

Desired_APDU_Size	**Number of OCTET STRING Data Octets (excluding tag octets)**
Up to 263 octets	Desired_APDU_Size - 12
Over 263 octets	Desired_APDU_Size - 14

19.X.1 Example ConfirmedPrivateTransfer Service

This is an example test message using a BACnet confirmed service.

Service =	ConfirmedPrivateTransfer
'VendorID'=	0
'ServiceNumber'=0	
'ServiceParameters'=	(An OCTET STRING with length calculated according to Table 19.X, e.g., 1476 - 14 = 1462)

19.X.2 Encoding for the Example

X'00'		PDU Type=0 (BACnet-Confirmed-Request-PDU, SEG=0, MOR=0, SA=0)
X'05'		Maximum APDU Size Accepted=1476 octets
X'55'		Invoke ID=85
X'12'		Service Choice=18 (ConfirmedPrivateTransfer)
X'09'		SD Context Tag 0 (Vendor ID, L=1)
X'00'		0 (ASHRAE)
X'19'		SD Context Tag 1 (Service Number, L=1)
X'00'		0 (ASHRAE Max Conveyable APDU Test Message)
X'2E'		PD Opening Tag 2 (Service Parameters)
	X'65'	Application Tag 6 (Octet String, L>4)
	X'FE'	Extended Length > 253
	X'05B6'	Extended Length = 1462
	X'00' ... '00'	(1462 octets of octet string data)
X'2F'		PD Closing Tag 2 (Service Parameters)

19.X.3 Procedure

Upon sending such a request, the BACnet device can expect one of the following situations to occur:

1) The remote peer responds. Any response by the remote peer, be it a positive or negative response, indicates that the request was successfully conveyed through the internetwork to the peer.
2) A Reject-Message-To-Network with a “Reject Reason” of 4 for the destination network is received. This indicates that the message cannot be conveyed through the internetwork to the peer because it is too large.
3) A Reject-Message-To-Network with a “Reject Reason” of a value other than 4 for the destination network is received indicating that some other failure is stopping completion of the test.
4) No response is received from the remote peer.

If situation 3 occurs, the test cannot continue and the maximum conveyable APDU cannot be determined at this time.

If situation 2 or 4 occurs, the message was too large. A shorter ConfirmedPrivateTransfer should be sent. Given that BACnet networks have specific maximum conveyable APDU sizes, the next size to check should be the next lower maximum conveyable APDU for the standard BACnet data link types. See Clause 20.1.2.5 for a list of suggested APDU lengths to test.

If situation 4 repeatedly occurs, regardless of the size of test message used, either the remote peer is now offline, the network has become disjoint, or some other catastrophic failure is occurring. The test should be restarted with a different selected remote peer device.

Clients that send messages to determine the maximum conveyable APDU length for remote networks shall cache the determined values so as to not have to repeat the test whenever the information is needed.