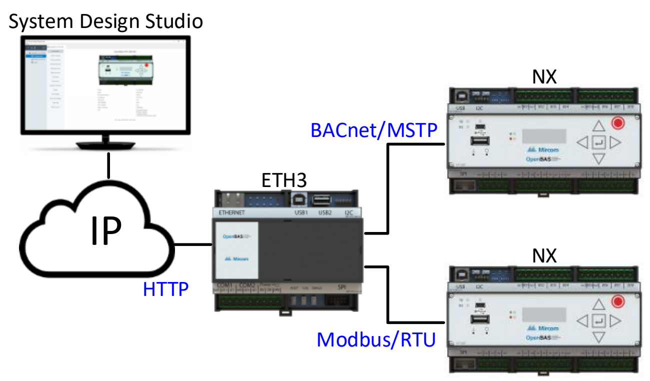
OpenBAS-NWK-ETH3 a true polyglot

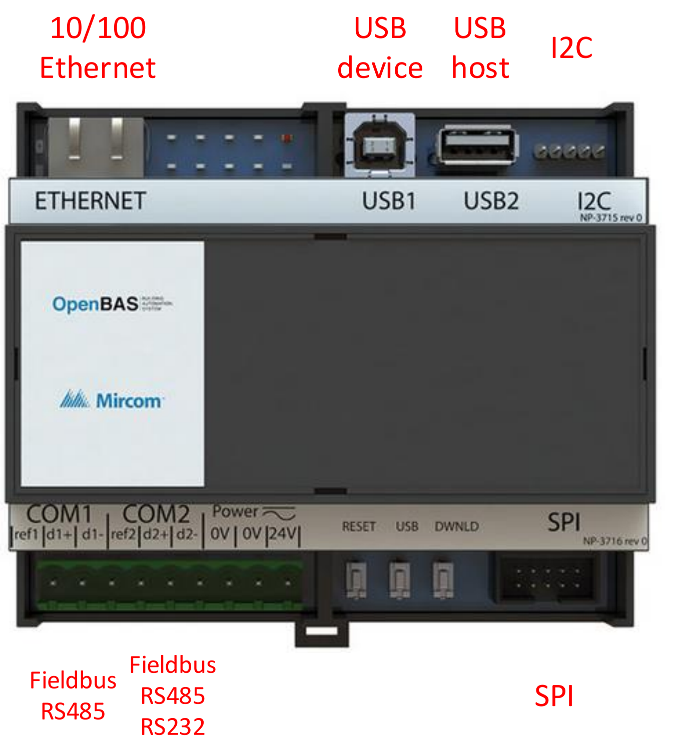
The new firmware **v3.09** of the **OpenBAS-NWK-ETH3** adds **BACnet/MSTP**-tunneling with which you can transparently, using **System Design Studio** (SDS), configure any OpenBAS-NX controller on a BACnet/MSTP network along other 3rd party controllers without any effort.

Furthermore, as it also supports **Modbus/RTU-MEI** tunneling (Modbus Encapsulated Interface) and has two RS485 field buses besides the Ethernet IP port, you can have on a single ETH3 Gateway a second fieldbus network with NX controllers in Modbus/RTU along 3rd party Modbus controllers and also configure them with **SDS**.

This whitepaper provides detailed information on how an Optomux message is transported first over **HTTP** when requested via **SDS** using Ethernet to the **ETH3** and is then encapsulated and tunneled on either BACnet/MSTP or Modbus/RTU RS485 fieldbuses.



First let us take a look at the ports available on the **ETH3**:



The following table provides a detailed description of each port use:

Port	Protocols supported
10-100 Ethernet	BACnet/IP on port 47808 Modbus/TCP on port 502 Telnet on port 23 with SQL for Arduino or Raspberry PI HTTP on port 80 SMTP on ports 25 and 587 SNMP on port 161 FTP on port 20
COM1 RS485	BACnet/MSTP with tunneling Modbus/RTU with MEI tunneling as master or slave Optomux as master or slave N2-Open as master or slave SQL-Arduino-Raspberry PI Mircom fire-panels
COM1 RS485 or RS232	Modbus/RTU with MEI tunneling as master or slave Optomux as master or slave N2-Open as master or slave SQL-Arduino-Raspberry PI Mircom fire-panels
SPI	Gateway to NX controllers
I2C	Gateway to NX peripherals
USB	Used as device or host for web page or massive data storage

The updated firmware provides BACnet/MSTP support only on **COM1** as it is highly resource intensive.

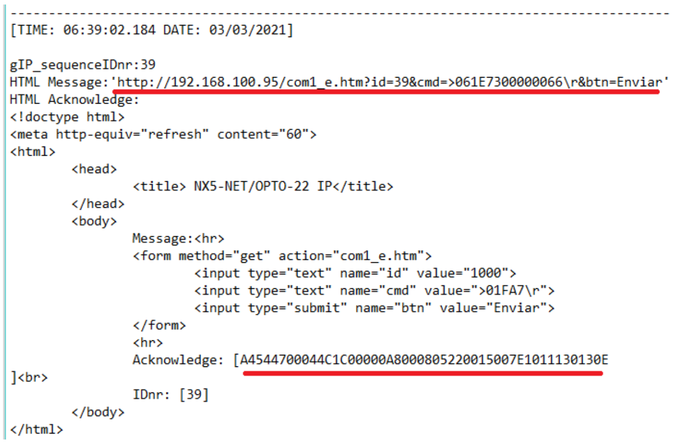
First to convey an Optomux message using HTTP, **SDS** generates a URL that provides information on how the message should be routed, four different URL encodings allow any of the following routes to be used internally on the ETH3:

URL	Use
opto22.htm	ETH3/SPI_NX bridging
com1_e.htm	ETH3/COM1 bridging or tunneling
com2_e.htm	ETH3/COM2 bridging or tunneling
eth_22.htm	ETH3 internal database

This is a typical URL that is requesting to send an Optomux request to an NX controller that is connected in **COM1** that is configured to use a BACnet/MSTP network using MAC 'X06':

http://192.168.100.95/**com1_e.htm**?id=39&cmd=>061E7300000066\r&btn=Enviar

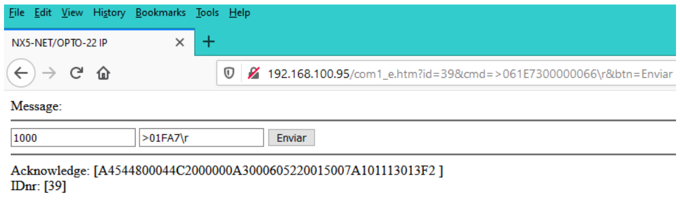
The **URL encoding** is shown below as provided on the LOG file for IP communication, highlighting both the request and the response or acknowledge:



```
--------------------------------------------------------------------------------
[TIME: 06:39:02.184 DATE: 03/03/2021]

gIP_sequenceIDnr:39
HTML Message:'http://192.168.100.95/com1_e.htm?id=39&cmd=>061E7300000066\r&btn=Enviar'
HTML Acknowledge:
<!doctype html>
<meta http-equiv="refresh" content="60">
<html>
        <head>
                <title> NX5-NET/OPTO-22 IP</title>
        </head>
        <body>
                Message:<hr>
                <form method="get" action="com1_e.htm">
                        <input type="text" name="id" value="1000">
                        <input type="text" name="cmd" value=">01FA7\r">
                        <input type="submit" name="btn" value="Enviar">
                </form>
                <hr>
                Acknowledge: [A4544700044C1C00000A8000805220015007E1011130130E
]<br>
                IDnr: [39]
        </body>
</html>
```

The nice thing is that you can also use any browser to do BACnet or Modbus tunneling without any effort, simply copy and paste the URL and place it in any browser's URL field and you will automatically get an Optomux response with the acknowledge encoded as shown below by the ETH3's integrated CGI (**Common Gateway Interface**):


File Edit View History Bookmarks Tools Help
NX5-NET/OPTO-22 IP
192.168.100.95/com1_e.htm?id=39&cmd=>061E7300000066\r&btn=Enviar
Enviar

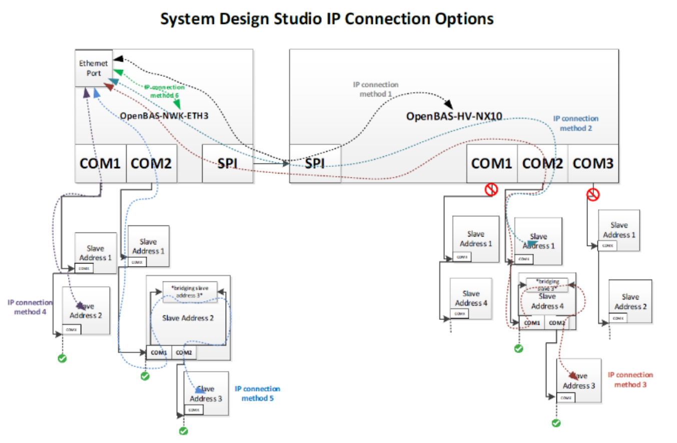
Message:

1000 >01FA7\r

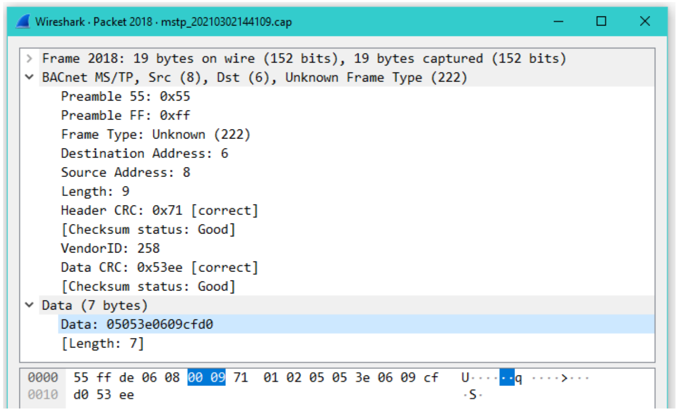
Acknowledge: [A4544800044C2000000A3000605220015007A101113013F2]
IDnr: [39]

A piece of cake!

The following diagram taken from **SDS v1.1.0** manual, shows the internal routings according to the URL:



Using Wireshark network analyzer software, we can easily decode the information present on the BACnet/MSTP network as depicted below:



```
> Frame 2018: 19 bytes on wire (152 bits), 19 bytes captured (152 bits)
v BACnet MS/TP, Src (8), Dst (6), Unknown Frame Type (222)
    Preamble 55: 0x55
    Preamble FF: 0xff
    Frame Type: Unknown (222)
    Destination Address: 6
    Source Address: 8
    Length: 9
    Header CRC: 0x71 [correct]
    [Checksum status: Good]
    VendorID: 258
    Data CRC: 0x53ee [correct]
    [Checksum status: Good]
v Data (7 bytes)
    Data: 05053e0609cfd0
    [Length: 7]

0000  55 ff de 06 08 00 09 71  01 02 05 05 3e 06 09 cf   U·····q ····>···
0010  d0 53 ee                                           ·S·
```

As we can see, the BACnet/MSTP tunneling employs the proprietary MSTP frame **222**, thus if we dissect the MSTP frame we will have the following:

The 0x55, 0xFF preamble followed by the frame type 222.

If we read the BACnet standard on section **9.3 MS/TP Frame Format** we find that:

"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 all MS/TP master nodes."

Then follows the Destination and source MAC addresses followed by the payload's data length (if any, otherwise it is set to zero) and finally the header's CRC checksum.

Then after the header, comes the DATA section, that ahead or it contains a WORD indicating the Vendor ID of that Frame, any of the following can be used for NX controllers as taken form the ASHRAE's BACnet site:

http://www.bacnet.org/VendorID/index.html

505	Mircom Group of Companies	Jason Falbo	25 Interchange Way Toronto, ON L4K 5W3 Canada

828	SAMDAV	Ricardo Galnares	Cerrada de Lerdo 6-E Distrito Federal, CP D.F. 10580 Mexico

The DATA section contains the Optomux message that uses the same compression method that Modbus **MEI** (**Modbus Encapsulated Interface**) employs to allow code reuse in both the ETH3 and NX firmware and is explained below in detail:

If we take a look at the document:

Modbus_Application_Protocol_V1_1a.pdf that can be downloaded from the **www.modbus.org** web page we will find in the following section:

6.19 Function code 43 (0x2B) Encapsulated Interface Transport

"Function Code 43 and its MEI Type 14 for Device Identification is one of two Encapsulated Interface Transport currently available in this Specification. The following function codes and MEI Types shall not be part of this published Specification and these function codes and MEI Types are specifically reserved: 43/0-12 and 43/15-255.
The MODBUS Encapsulated Interface (MEI)Transport is a mechanism for tunneling service requests and method invocations, as well as their returns, inside MODBUS PDUs.
The primary feature of the MEI Transport is the encapsulation of method invocations or service requests that are part of a defined interface as well as method invocation returns or service responses."

This is a typical Modbus MEI packet:

Request

Function code	1 Byte	**0x2B**
MEI Type*	1 Byte	0x0E
MEI type specific data	n Bytes	

* MEI = MODBUS Encapsulated Interface

Response

Function code	1 Byte	**0x2B**
MEI Type	1 byte	0x0E
MEI type specific data	n Bytes	

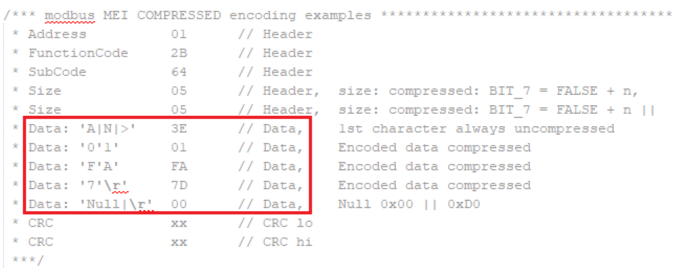
Error

Function code	1 Byte	**0xAB : Fc 0x2B + 0x80**
MEI Type	1 Byte	0x0E
Exception code	1 Byte	01, 02, 03, 04

Because both **BACnet** and **Modbus** employ binary data as compared to **Optomux** that uses ASCII encoded plain text, to save space and make the tunneled data shorter and faster to be conveyed, an **encoding / decoding** is implemented by the ETH3 that **compresses / decompresses** the tunneled Optomux message before being placed into the BACnet or Modbus frames and is as follows:

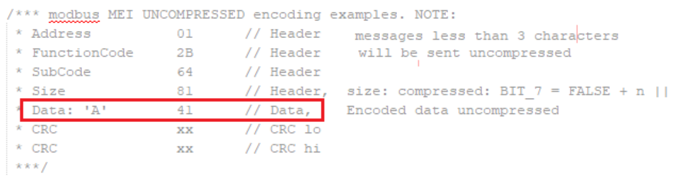
In modbus the data part of the tunneled Optomux message is compressed if only hexadecimals characters '0' thru 'F' are present on the message, the first character of the message used as the START flag: '>' is sent uncompressed, the '\r' END character is encoded as either a 0xD character on the lower nibble of the last byte and then the last byte will be encoded as 0x00 or if the number of characters are even, then it will be encoded as a 0x0D byte at the end.

Below is a **Modbus-MEI** encoded compressed packet.



```
/*** modbus MEI COMPRESSED encoding examples ************************************
 * Address          01       // Header
 * FunctionCode     2B       // Header
 * SubCode          64       // Header
 * Size             05       // Header,   size: compressed: BIT_7 = FALSE + n,
 * Size             05       // Header,   size: compressed: BIT_7 = FALSE + n ||
 * Data: 'A|N|>'    3E       // Data,     1st character always uncompressed
 * Data: '0'1'      01       // Data,     Encoded data compressed
 * Data: 'F'A'      FA       // Data,     Encoded data compressed
 * Data: '7'\r'     7D       // Data,     Encoded data compressed
 * Data: 'Null|\r'  00       // Data,     Null 0x00 || 0xD0
 * CRC              xx       // CRC lo
 * CRC              xx       // CRC hi
 ***/
```

For messages less than 3 bytes or that have other ASCII characters outside of the '0' thru 'F' range that are non-valid Optomux characters, they are sent uncompressed as shown below.



```
/*** modbus MEI UNCOMPRESSED encoding examples. NOTE:
 * Address          01       // Header     messages less than 3 characters
 * FunctionCode     2B       // Header      will be sent uncompressed
 * SubCode          64       // Header
 * Size             81       // Header,   size: compressed: BIT_7 = FALSE + n ||
 * Data: 'A'        41       // Data,     Encoded data uncompressed
 * CRC              xx       // CRC lo
 * CRC              xx       // CRC hi
 ***/
```

This encoding example for Modbus/RTU also applies to BACnet/MSTP using the proprietary frame type 222 (0xDE) and the DATA occupies the rest of the packet.

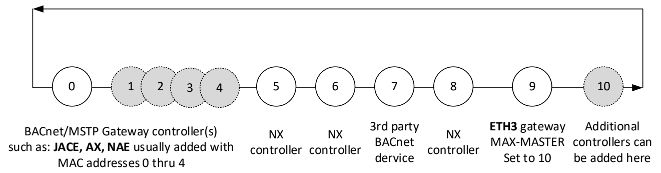
Both protocols use a 16-bit CRC to verify the integrity of the packet, and the tunneled message also contains its own CRC thus the data's integrity is double-fold verified upon arrival as both CRC's are checked.

BACnet/MSTP networking tips:

To improve the operation of (any) **BACnet/MSTP** network, the following points have to be followed, not doing so will degrade the bandwidth and thus the performance of the network:

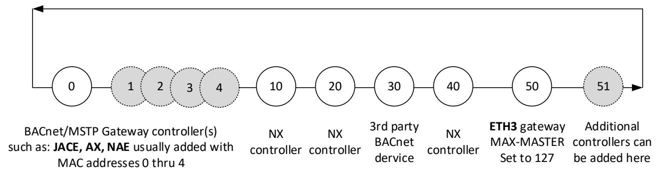
- MAC addresses 0-4 are usually reserved for bus Gateways, remember all **BACnet/MSTP** (Master-Slave-Token-Passing) devices are both true **Masters** and **Slaves.** When a MAC address holds the **Token** it becomes the Master and all other peers are then Slaves, hence it is a true multi-master network. Only devices with "A" functionality, normally used as Gateways employ the largest part of the bandwidth for polling and reading or writing data.
- All other (non-gateway or router) devices can use MAC addresses 5 thru 127.
- Leave **NO** gaps or spaces between MAC addresses to avoid waste maintenance **Poll-For-Master** frames that in average stall the network with a dead time around 20-80 milli seconds for each frame sent when the polled device does not exist or is off-line. Each 50 times that the token is received by any MSTP device on the network, it must do a maintenance poll-for-master cycle looking for added devices to heal broken token sequences. These are completely avoided by using contiguous addresses.
- If possible, assign **ETH3's** fields bus port **COM1** to use the **last MAC address** and set its **MAX-MASTER** to its own MAC address + 1, to still allow for more devices to be added on top of it if when necessary and wasting no unnecessary time.

The following diagram shows a typical good networking scenario.


0
1
2
3
4
5
6
7
8
9
10
BACnet/MSTP Gateway controller(s) such as: JACE, AX, NAE usually added with MAC addresses 0 thru 4
NX controller
NX controller
3rd party BACnet dervice
NX controller
ETH3 gateway MAX-MASTER Set to 10
Additional controllers can be added here

Here the lower MAC addresses that are usually assigned to gateways in the range 0 to 4, ensure they start the token sequence ahead of any other devices that are present. Then a **no-gap** MAC addressing scheme from 5 thru 8 is assigned. At the end place the ETH3 whose MAC address is set to 9 and has its **MAX MASTER** set to 10 thus it will only look up to address 10 before reverting back to 0 to search for other devices and pass the token.

A poor network addressing scheme is shown below, where gaps exist between the MAC addresses.


0
1
2
3
4
10
20
30
40
50
51
BACnet/MSTP Gateway controller(s) such as: JACE, AX, NAE usually added with MAC addresses 0 thru 4
NX controller
NX controller
3rd party BACnet dervice
NX controller
ETH3 gateway MAX-MASTER Set to 127
Additional controllers can be added here

Here, for example MAC address 10 will periodically every 50 tokens start a maintenance **Poll-For-Master** sequence looking for MAC addresses: 11, 12, 13, 14, 15, 16, 17, 18 and 19 before finally finding MAC address 20.

The same wasteful sequence will again happen for MAC addresses 20, 30, 40 and 50. Even worst for this last one address, as its MAC address is set to 127 which is the maximum allowed MAC address for MSTP devices and every 50 tokens received, it will look for addresses from 51 and all the way thru 127 and then fold back to MAC address 0 all the way to 49 until it finds some device that is present.

And this will repeat forever, for every single node every 50 times the token is received by it!

Once again, but now in big size, the golden rules for MSTP networks are:

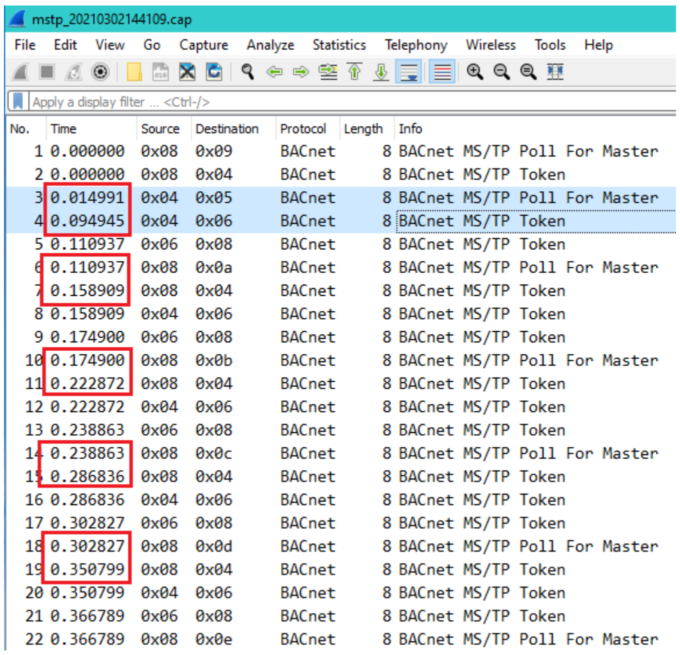
- ⊗ Leave NO-GAPS in MAC addresses.
- ⊗ Set MAX MASTER of the last device on the network to its own MAC address + 1.
- ⊗ Follow good RS485 wiring practices using EOL and BIAS resistors to properly polarize the bus and avoid reflections that will corrupt data.

The following **Wireshark** MSTP capture shows this wasteful sequence in great detail, where each **Poll-For-Master** frame kills around 50-80 milli seconds of network bandwidth in the field bus.

Frame #	Timing	Difference
3	0.014991	
4	0.094945	0.079954
6	0.110937	
7	0.158909	0.047972
10	0.174900	
11	0.228720	0.053820
14	0.238863	
15	0.286836	0.047973
18	0.302827	
19	0.350799	0.047972

Maximum	0.080
Minimum	0.048
Average	0.056

Image showing the **Poll-For-Master** sequence of different MAC addresses with GAPS.

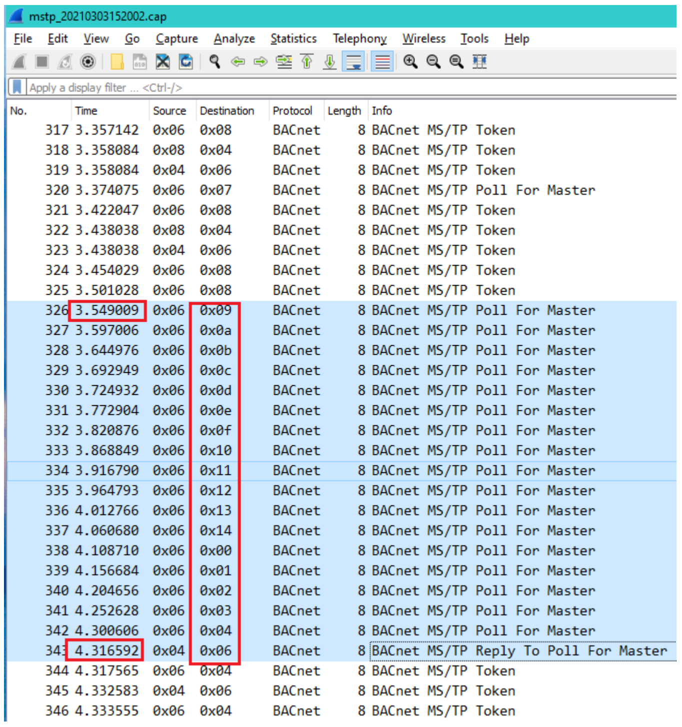

mstp_20210302144109.cap
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
Apply a display filter ... <Ctrl-/>

```
No.  Time      Source Destination Protocol Length Info
  1 0.000000  0x08   0x09        BACnet        8 BACnet MS/TP Poll For Master
  2 0.000000  0x08   0x04        BACnet        8 BACnet MS/TP Token
  3 0.014991  0x04   0x05        BACnet        8 BACnet MS/TP Poll For Master
  4 0.094945  0x04   0x06        BACnet        8 BACnet MS/TP Token
  5 0.110937  0x06   0x08        BACnet        8 BACnet MS/TP Token
  6 0.110937  0x08   0x0a        BACnet        8 BACnet MS/TP Poll For Master
  7 0.158909  0x08   0x04        BACnet        8 BACnet MS/TP Token
  8 0.158909  0x04   0x06        BACnet        8 BACnet MS/TP Token
  9 0.174900  0x06   0x08        BACnet        8 BACnet MS/TP Token
 10 0.174900  0x08   0x0b        BACnet        8 BACnet MS/TP Poll For Master
 11 0.222872  0x08   0x04        BACnet        8 BACnet MS/TP Token
 12 0.222872  0x04   0x06        BACnet        8 BACnet MS/TP Token
 13 0.238863  0x06   0x08        BACnet        8 BACnet MS/TP Token
 14 0.238863  0x08   0x0c        BACnet        8 BACnet MS/TP Poll For Master
 15 0.286836  0x08   0x04        BACnet        8 BACnet MS/TP Token
 16 0.286836  0x04   0x06        BACnet        8 BACnet MS/TP Token
 17 0.302827  0x06   0x08        BACnet        8 BACnet MS/TP Token
 18 0.302827  0x08   0x0d        BACnet        8 BACnet MS/TP Poll For Master
 19 0.350799  0x08   0x04        BACnet        8 BACnet MS/TP Token
 20 0.350799  0x04   0x06        BACnet        8 BACnet MS/TP Token
 21 0.366789  0x06   0x08        BACnet        8 BACnet MS/TP Token
 22 0.366789  0x08   0x0e        BACnet        8 BACnet MS/TP Poll For Master
```

By using a good MAC addressing scheme you will ensure a fast and responsive BACnet/MSTP network.

In this other example we can see that when MAC address 8 is taken off-line, the Poll-For-Master searching for the next MAC address heals the token cycle by polling from MAC addresses 9 thru 20 and then folds back from 0 all the way up until it finds MAC address 4.

At that time, the node with the MAC address 4 replies with the **Reply-To-Poll-For-Master** frame and then finally the token sequence can start again.


mstp_20210303152002.cap
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
Apply a display filter ... <Ctrl-/>

```
No.  Time      Source Destination Protocol Length Info
317 3.357142  0x06   0x08        BACnet        8 BACnet MS/TP Token
318 3.358084  0x08   0x04        BACnet        8 BACnet MS/TP Token
319 3.358084  0x04   0x06        BACnet        8 BACnet MS/TP Token
320 3.374075  0x06   0x07        BACnet        8 BACnet MS/TP Poll For Master
321 3.422047  0x06   0x08        BACnet        8 BACnet MS/TP Token
322 3.438038  0x08   0x04        BACnet        8 BACnet MS/TP Token
323 3.438038  0x04   0x06        BACnet        8 BACnet MS/TP Token
324 3.454029  0x06   0x08        BACnet        8 BACnet MS/TP Token
325 3.501028  0x06   0x08        BACnet        8 BACnet MS/TP Token
326 3.549009  0x06   0x09        BACnet        8 BACnet MS/TP Poll For Master
327 3.597006  0x06   0x0a        BACnet        8 BACnet MS/TP Poll For Master
328 3.644976  0x06   0x0b        BACnet        8 BACnet MS/TP Poll For Master
329 3.692949  0x06   0x0c        BACnet        8 BACnet MS/TP Poll For Master
330 3.724932  0x06   0x0d        BACnet        8 BACnet MS/TP Poll For Master
331 3.772904  0x06   0x0e        BACnet        8 BACnet MS/TP Poll For Master
332 3.820876  0x06   0x0f        BACnet        8 BACnet MS/TP Poll For Master
333 3.868849  0x06   0x10        BACnet        8 BACnet MS/TP Poll For Master
334 3.916790  0x06   0x11        BACnet        8 BACnet MS/TP Poll For Master
335 3.964793  0x06   0x12        BACnet        8 BACnet MS/TP Poll For Master
336 4.012766  0x06   0x13        BACnet        8 BACnet MS/TP Poll For Master
337 4.060680  0x06   0x14        BACnet        8 BACnet MS/TP Poll For Master
338 4.108710  0x06   0x00        BACnet        8 BACnet MS/TP Poll For Master
339 4.156684  0x06   0x01        BACnet        8 BACnet MS/TP Poll For Master
340 4.204656  0x06   0x02        BACnet        8 BACnet MS/TP Poll For Master
341 4.252628  0x06   0x03        BACnet        8 BACnet MS/TP Poll For Master
342 4.300606  0x06   0x04        BACnet        8 BACnet MS/TP Poll For Master
343 4.316592  0x04   0x06        BACnet        8 BACnet MS/TP Reply To Poll For Master
344 4.317565  0x06   0x04        BACnet        8 BACnet MS/TP Token
345 4.332583  0x04   0x06        BACnet        8 BACnet MS/TP Token
346 4.333555  0x06   0x04        BACnet        8 BACnet MS/TP Token
```

The whole process took from frames 326 thru 343 and spanned from 3.549 to 4.316 seconds = **767 milli seconds** therefore, to traverse those 17 nodes to heal the network, an average **Poll-For-Master** time of 45.5 milli seconds was used or better say wasted.

If **MAX MASTER** had been set to **127** instead of **20**, the time to heal a broken network would have taken instead:

9 thru 127 + 0 thru 4

= 122 nodes

= **5.504 seconds.**