The Feasibility Study of DeviceNet over Extended CAN

Kiah Hion Tang and Richard McLaughlin, University of Warwick, United Kingdom

kiah.tang@warwick.ac.uk and r.mclaughlin@warwick.ac.uk

Abstract

This paper discusses the feasibility of porting the existing DeviceNet protocol, which utilises only Standard CAN (11-bit Identifier), onto Extended CAN (29-bit). It highlights on the background, limitations, and problem analysis of the existing DeviceNet protocol over Standard CAN, introducing the new DeviceNet over the Extended CAN concept (DeviceNeX). The advantages and disadvantages of the DeviceNeX protocol will also be discussed in-depth. Finally the issues of Conformance, Physical Layer, and Interoperability Testing with DeviceNeX will be addressed, followed by the conclusion.

1. Introduction

To date, there are two variants of the CAN (Controller Area Network) standard available, i.e.:

- Standard CAN. As defined in the CAN specification 2.0A, it supports 11-bit identifier format. This version is not future proof, i.e. it generates errors when newer format messages (i.e. the Extended CAN frame, as below) is received.
- Extended CAN. As defined in the CAN specification 2.0B, it supports both 11-bit and 29-bit identifier formats. In addition, to minimise the effort of upgrading the existing 2.0A compliance CAN silicon to work with 2.0B, the silicon manufacturers have further defined two variants of 2.0B compliant silicon, namely 2.0B passive and 2.0B active. The passive version recognises the Standard CAN format and ignores the Extended CAN format without flagging errors. The active version recognises both formats.

As the CAN Specification Version 2.0B has been defined since it's first released by Bosch in 1991[i], the number of CAN2.0B compliance silicon are increasing drastically over these years. Data collected by CiA (CAN in Automation, Germany) shows that 81% of the available CAN silicon support the CAN 2.0B active format, 14.5% support the CAN 2.0B passive format, and 4.5% support only the CAN 2.0A format[ii].

DeviceNet was developed at the early stage of CAN, therefore it adopts the Standard CAN format. Due to the shortcomings and limitations of the Standard CAN silicon in the DeviceNet protocol definition, a study was carried out by the authors to investigate the feasibility of porting the DeviceNet protocol to the Extended CAN (29-bit) format.

2. DeviceNet Problem Analysis

DeviceNet has a few limitations against the Standard CAN architecture. In this section, some of the major limitations are discussed, which leads to the definition of the new DeviceNeX protocol.

2.1. 11-bit CAN Identifier Limitations

Due to the nature of CAN, the rule of thumb in designing the CAN higher layer protocol is that, no more than one node should share the same CAN identifier when producing messages. In another words, the CAN identifier must be unique to each node on the same network.

However, due to the protocol definition of DeviceNet and the limited identifiers of the Standard CAN, there are several potential problems could arise. In addition, it also restricts the enhancement of the protocol. Two typical problems are highlighted here.

2.1.1. Duplicate MAC ID Check Message

In DeviceNet, in order to ensure the assigned MAC ID has not been occupied by another node on the same network, a node needs to broadcast the duplicate MAC ID check message twice, at one- second intervals before joining the network.

The format of the Duplicate MAC ID Check message is shown in Figure 1. As can be seen, the CAN identifier field is fixed for each MAC ID. For example, the Duplicated MAC ID Check message transmitted by a node with MAC ID 37 occupies CAN identifier 52F Hex.

In this case, when there is more than one node transmitting the duplicate MAC ID check message at the same time, these messages collide. Due to the identical value in the identifier field and the difference in the data field, error frames will be generated, causing these nodes to re-transmit. The process is likely to repeat until these nodes transition to the Bus Off state where both nodes are not allowed to participate in the network activity.

This situation is quite common in practice, because the default out-of-box settings of DeviceNet are MAC ID 63 at 125 Kbps. This means that these DeviceNet nodes cannot be connected to the network at the same time.

CAN Identifier Field											
10	9	8	7	6	5	4	3	2	1	0	Description
1	0	MAC ID						1	1	1	Duplicate MAC ID Check

Data Field Off Set	7	6	5	4	3	2	1	0
0	Port Number							
1	Vendor ID (16-bit)							
2								
3	Serial Number (32-bit)							
4								
5								
6								

Figure 1 Duplicate MAC ID Check Message

DeviceNet specification Release 2.0 has added a set new messages called the Offline Communication Set. This allows a node that has transitioned to the offline state to communicate with a tool to, for example, change its MAC ID to a new value and re-execute the network access state machine. However, this does present some shortcomings:

- The Offline Communication protocol is complicated and adds significant amount of code to the already-limited memory space within the low-end microcontrollers.
- It inherently implies that a node which has been put to bus off can reset its CAN interface and continue participate on the network activity. This might lead to a very disastrous situation where a node keeps resetting itself and generates error frames due to the failure of its interface.

Therefore a much better method should be defined to avoid this from happening.

2.1.2. Predefined Master Slave Connection Set

A similar problem is encountered on the Predefined Master/Slave Connection Set, particularly on the messages sent by the master to the slave.

CAN Identifier Field											
10	9	8	7	6	5	4	3	2	1	0	Description
1	0	Destination MAC ID						0	1	0	Master COS/Cyclic Ack
1	0	Destination MAC ID						1	0	0	Master Explicit Request
1	0	Destination MAC ID						1	0	1	Master I/O Poll Command
1	0	Destination MAC ID						1	1	0	G2 Unconnected Request

Figure 2 Predefined Master/Slave Message

As shown in Figure 2, messages sent by the master have the slave's MAC ID within the CAN identifier field. This implies that it is technically possible for more than one device (either another master or tool) to send messages across the same CAN identifiers.

One can argue that if this situation happens, it is a network design problem rather than the protocol, because one slave device can only be mapped to one master. However the authors believe the policy of a "robust protocol" is that a failure situation must be prevented from the protocol, rather than the implementation, point of view.

The reason of placing the slave MAC ID solely within the CAN identifier is because, at the early stage of the DeviceNet life, only three CAN silicon implementations were available, i.e. 82C200 by Philips, 68HC05XX by Motorola, and 82527 by Intel. Among these three CAN silicon implementations, the 82C200 and 68HC05XX are BasicCAN and 82527 is FullCAN.

Although there is no concrete definition about BasicCAN and FullCAN, a BasicCAN silicon implementation usually has only one transmit and one received buffer, whilst FullCAN has several message objects which can be configured to transmit or receive. In addition, the two BasicCAN silicon implementations do not have full masking on the identifier field, only upper 8 bits (ID3 to ID10) can be masked for filtering.

In order to reduce the unwanted interrupts to the slave device (which usually has a slow processor), the destination MAC ID is placed at the upper 8 bits, as seen in Figure 2. In this case, the slave device will be interrupted by messages intended for it only.

Although the idea is good, it does not interoperate with the CAN protocol due to the arbitration mechanism, i.e. it may potential cause error frames.

2.2. Bit Rate

DeviceNet supports three bit rates i.e. 125, 250, and 500 Kbps. Since CAN is a base band technology, these three bit rates cannot co-exist on the same network at a time. Incorrectly set baudrate will lead to network disaster.

DeviceNet provides two options for setting the bit rate, i.e. by hardware switch or software (across DeviceNet). However, it does not prevent any incorrectly set baudrate from disturbing the network. In fact, all CAN based protocols have the same problem.

3. Introduction to DeviceNeX

In this section, the new CIP (Control and Information Protocol) based device level protocol, called DeviceNeX, is introduced. The concepts and features will be explored.

3.1. Dynamic Node Address Allocation

The first feature introduced by DeviceNeX is the dynamic node address allocation. With this feature, the devices, when online, do not carry any addresses. The node address is assigned by the Network Manager (see later section) at run time.

In this case, the node address no longer represents the Medium Access Control (MAC) identifier, but the address that is allocated for a particular operation. Hence, the node address is known as the Operational Address (OA) in DeviceNeX.

This feature effectively allows redundant devices to be connected to the network for safety purpose, which will be presented in the later section of this paper.

3.2. Support Up To 127 Operational Addresses

DeviceNeX supports up to 127 Operational Addresses, as compared to 64 in DeviceNet. This allows wider range of the products to be connected onto a single network. Although in the existing market there is no single-chip CAN transceiver support up to 127 (or more) nodes on one physical network, there are a few CAN bridges or repeaters available which allow a logical network to be segmented into a few physical networks.

3.3. Auto Bit Rate Detection

In DeviceNeX, the auto bit rate detection becomes a requirement. With this requirement, a node, when initialise, must detect the correct bit rate used on the network and configure its interface correctly before joining the network. This must be done without sending error frames onto the network.

There are already a few CAN silicon that are capable of detecting the bit rate without disturbing the network, such as SJA1000 from Philips Semiconductors. For CAN silicon that cannot perform the auto bit rate detection without jeopardising the network, other methods must be utilised to prevent the error frames from being broadcasted onto the network during the auto bit rate detection period[iii].

This requirement effectively reduces the possibility of bringing the network down when incorrect bit rate it set.

In addition, the elimination of the fixed node address and bit rate settings implies that neither I/O ports nor none-volatile memory is required to store these settings, hence free up more resources for other applications.

3.4. Connection and Connectionless Messaging

Recall that DeviceNet is a connection-based protocol, i.e. prior the communication, a connection must be established between two (or more) nodes.

In DeviceNeX, the connectionless message is also supported, i.e. a device can send a connectionless message (explicit) to another without having to establish the connection.

Any services that will modify the parameters of the server end points are not allowed. At the first release of this protocol, the only service applicable to the connectionless message is Get_Attribute_Single

In this case, any nodes can send the request to obtain the information (for e.g. the Vendor ID, serial number, etc) of a particular device without going through the unnecessary connection establishment process.

4. DeviceNeX Concept

4.1. Hello Message

Similar to DeviceNet, all DeviceNeX devices are required to transmit a special message twice before going online. In DeviceNeX this message is called the “Hello Message”. Unlike the “Duplicate MAC ID Check Message” in DeviceNet, this message does not cause the device to go offline.

29-bit CAN Identifier field																												
28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	1	Source OA							R	Serial Number 0								Serial Number 1							

CAN Data Field Offset	Description	
0	Serial Number 2	
1	Serial Number 3	
2	Vendor ID Low	
3	Vendor ID High	
4	Product Code Low	
5	Product Code High	
6	Major Revision	
7	Reserved	Physical Ports Number

Figure 3 Hello Message

As can be seen in Figure 3, the Hello Message contains the universal unique identifier (i.e. Vendor ID, Product Identification, Major Revision, and Serial Number of a device) of a particular device.

The default out-of-box Source OA is 127. The rule of the Hello Message is as follow:

- When broadcasting with OA 127, no response is expected. The device, after completing two Hello Messages, transition to the “Stand-By” state.
- When broadcasting with OA other than 127, response may be received from another node with the same OA. In this case, the request node reinitialises itself with the default OA (127) and rejoin the network by executing another set of the Hello Message. Otherwise the node transitions to the “Online” state.

When this message is broadcasted, the Network Manager will receive and store it into its database for future use.

4.2. Mini-who Request Message

Occasionally the Network Manager needs to know the devices that are currently at the ***Standby*** State and not assigned with any OA (i.e. have default OA of 127). This is achieved by broadcasting a message that is identical to the Hello Message, except that the message is a Remote Frame, i.e. the CAN Remote Transmit Request (RTR) bit is set.

Upon the detection of the Mini-who request, all devices with OA 127 trigger the transmission of the Hello Message.

29-bit CAN Identifier field																												
28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	1	Destination OA = 127							0	Serial Number 0								Serial Number 1							

Figure 4 Mini-who Request Message

The Network Manager (or any producers of the Mini-who Message) must place the lower two bytes of its unique serial number into the CAN Identifier field, to avoid the collision error.

A device can set up its mask-and-match register as shown in Figure 5. This enables a device that is at the ***Standby*** State to receive only the Hello Request Message and the ROAM message. The huge reduction of the interrupt rates allows the device to put itself to the low-power standby mode, thus reduces the unnecessary energy consumption.

29-bit CAN Identifier field

28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	X	1	1	1	1	1	1	1	0	Don't Care															

Figure 5 Mini-who Request Message

4.3. ROAM Message

The Reallocate Operational Address Message (ROAM) is used to reallocate the OA of a target device from and only from 127 to other values. There is no response (success or failure) to the ROAM request

If the target OA is not 127, then the Connection based Explicit Message must be invoked to reallocate the OA. In another word, this message is valid only when the target device has OA 127.

The complete ROAM Request format is similar to the Hello Message, except it contains the target node's identification code, as shown in Figure 6.

29-bit CAN Identifier field																												
28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	0	Destination OA = 127							0	Target Serial Number 0								Target Serial Number 1							

CAN Data Field Offset	Description	
0	Target Serial Number 2	
1	Target Serial Number 3	
2	Target Vendor ID Low	
3	Target Vendor ID High	
4	Target Product Code Low	
5	Target Product Code High	
6	Target Major Revision	
7	0	Requested OA

Figure 6 ROAM

Upon the success change of its OA, the target device reinitialises itself with the new OA, and broadcasting two Hello Messages with the new OA before transition to the ***Online*** State. The Network Manager can then connect to the target device across the new OA by sending the Connection request Message.

4.4. Connection Request/Response Message

The Connection Request/Response Message is invalid for OA of 127. This means in order to establish connections with a node, the OA of both parties must not be 127.

The Connection Request Message has the following format.

CAN Data Field Offset	Description		
0	Service Code = 0x4B (Connection Request)		
1	P	Group Select	Source Message ID
2	Target Connection Instance ID (Conditional)		
3			
4	Target Expected Packet Rate (Conditional)		
5			
6	Target Watchdog Timeout Action (Conditional)		

Figure 7 Request

As seen in Figure 7, the Connection Request Message contains three conditional parameters, i.e. the Target Connection Instance ID, Expected Packet Rate, and Watchdog Timeout Action attributes.

This means that the Connection Request Message Body can either be two bytes or seven bytes in length.

When the Connection Request Message Body is two bytes in length (i.e. no Connection Instance ID, Expected Packet Rate, or Watchdog Timeout Action values are specified); the requested connection, if successful, transitions to the Configuring State with default attribute values. This allows the connection to be reconfigured (via the explicit message connection) prior to transition to the Established State.

When the Connection Request Message Body is seven bytes in length (i.e. all Connection Instance ID, Expected Packet Rate, and Watchdog Timeout Action values are specified), the requested connection, if successful, bypassing the Configuring State and transitions to the Established State, using the default attribute value.

This implies that the establishment of an explicit message connection is not required prior to the establishment of the implicit message. This removes the unnecessary overhead for connection establishment, and is very useful for the Predefined Connection set.

The success Connection Response format is shown in Figure 8.

CAN Data Field Offset	Description		
0	Service Code = 0x4B (Connection Request)		
1	P	Actual Group	Source Message ID
2	Reserved		Message Body Format
3	Connection Instance ID or Actual Expected Packet Rate Value (Conditional)		
4			

Figure 8 Response

The Message Body Format attribute indicates the size of class and instance ID, as in DeviceNet.

The content of offset Byte 3 and Byte 4 vary with the request message. If the request message is targeted on the Predefined Connection Set, then this field contains the Actual Expected Packet Rate value. Otherwise it contains the Connection Instance ID.

4.5. Predefined Connection Sets

The Predefined Connection Set in DeviceNeX defines a set of default connection behaviours (by manipulating the attributes' value) without reconfiguring the connections. This is not the same as the Predefined Master/Slave Connection Set in DeviceNet, since in DeviceNeX all devices are operating in the peer-to-peer mode.

The Predefined Connection Set simplifies the Connection Establishment process as well as reducing the need for the device resources, which is very critical in the embedded environment. The predefined connections are shown in Figure 9.

Connection Instance ID	Description
1	Explicit Message Connection
2	Standard Polled Implicit Connection (11-bit)
3	Extended Polled Implicit Connection (29-bit)
4	Standard Broadcast Implicit Connection (11-bit)
5	Extended Broadcast Implicit Connection (29-bit)
6	Standard Multicast Implicit Connection (11-bit)
7	Extended Multicast Implicit Connection (29-bit)
8	Standard Change Of State Implicit Connection (11-bit)
9	Extended Change Of State Implicit Connection (29-bit)
10	Standard Cyclic Implicit Connection (11-bit)
11	Extended Cyclic Implicit Connection (29-bit)
12	Standard Multicast Implicit Connection (11-bit)
13	Extended Multicast Implicit Connection (29-bit)
14 to 127	Reserved for future use
128 to 255	Dynamic Connection
256 to 65535	Reserved for future use

Figure 9 Predefined Connection Settings

As can be seen, the Connection Instance Identifiers in the range of 14 and 65535 are reserved for future use.

CAN Data Field Offset	Description		
0	Service Code = 0x4B (Connection Request)		
1	P	Group Select	Source Message ID
2	Target Connection Instance ID (Conditional)		
3			
4	Target Expected Packet Rate (Conditional)		
5			
6	Target Watchdog Timeout Action (Conditional)		

Figure 10 Predefined Connection Request

To differentiate between the Dynamic and Predefined connections, the **P** (Predefined) flag (Bit 8 Offset 1) within the Connection Request/Response Message is set.

When establishing a Predefined Connection Set, the requester specifies the following within the Connection Request Message:

- The target Connection ID

- The Expected Packet Rate
- The Watchdog Timeout Action

If the target device supports the connection requested, a success response will be returned, specifying the actual Expected Packet Rate value. The communication will then start immediately.

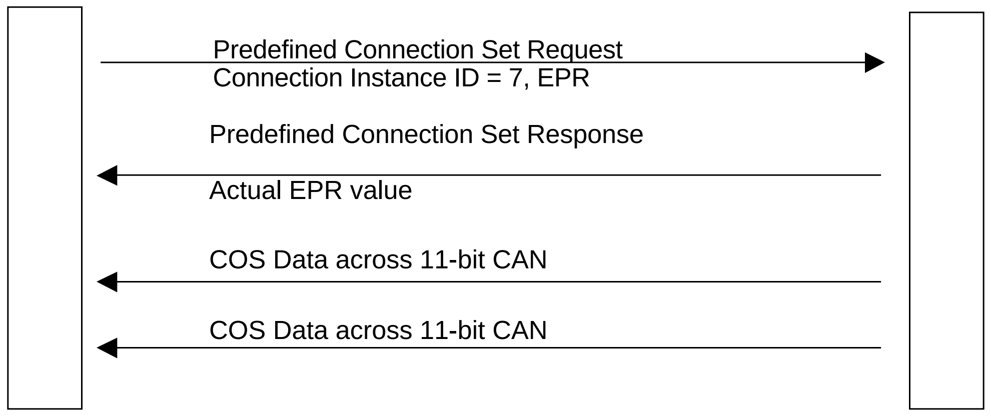
Predefined Connection Set Request
Connection Instance ID = 7, EPR
Predefined Connection Set Response
Actual EPR value
COS Data across 11-bit CAN
COS Data across 11-bit CAN

Figure 11 Predefined Connection Set Establishment Process

As can be seen in Figure 11, the connection establishment process takes only two steps, i.e. one request and one response. This greatly simplified process reduces the network down time by removing unnecessary overheads, thus improves the performance.

Each of these connections can be combined with others or working individually.

5. Implementation Example

In this section, the actual implementation of the DeviceNeX protocol is presented. This includes the setting up of the network, locating, connecting, configuring, and communicating with the devices.

Throughout this section, the network design shown in Figure 12 is assumed.

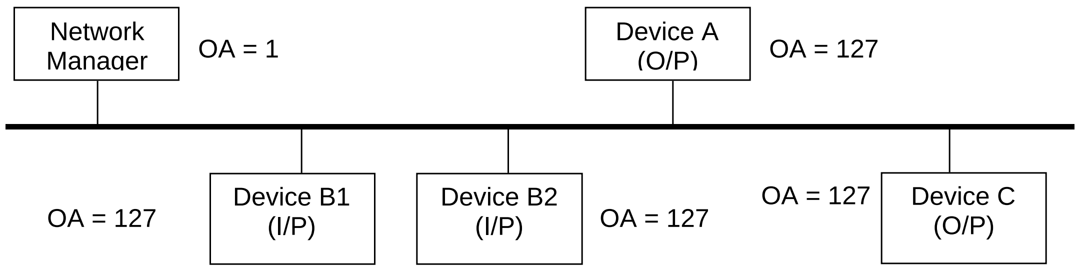

Figure 12 DeviceNeX Implementation Template

B1 and B2 are redundant input devices for safety critical purpose. The input status will be consumed by Device A and Device C.

1-1 Network Set Up

There is no device installation sequence or order required for the network set up; all devices can be installed to the network at any time, including the Network Manager. The only requirements are to configure the Network Manager for the desired network baud rate and OA.

The Network Manager can either be pre-configured with planned EDS information obtained during system planning, or build the a database from the network.

In this implementation example, assuming the Network Manager has been loaded with a pre-configured database.

1-2 Baud Rate Detection

Assuming Device A, B1, B2, and C are installed to the network prior to powering up the network, as shown in Figure 13.

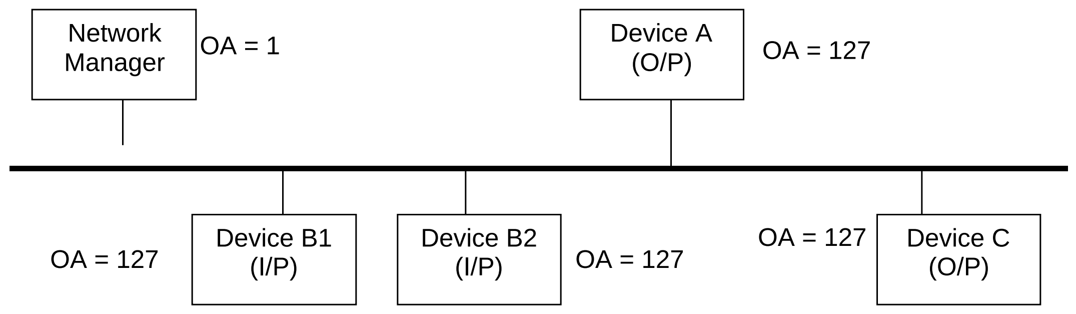

Figure 13 Initial Set Up

After the network power is applied, these devices are in the passive mode, waiting for the bus activity for the detection of the correct bit-rate. As long as the network bit-rate is not detected, these devices remain passive, and they can optionally suspend to the low-power standby mode to reduce the energy lost.

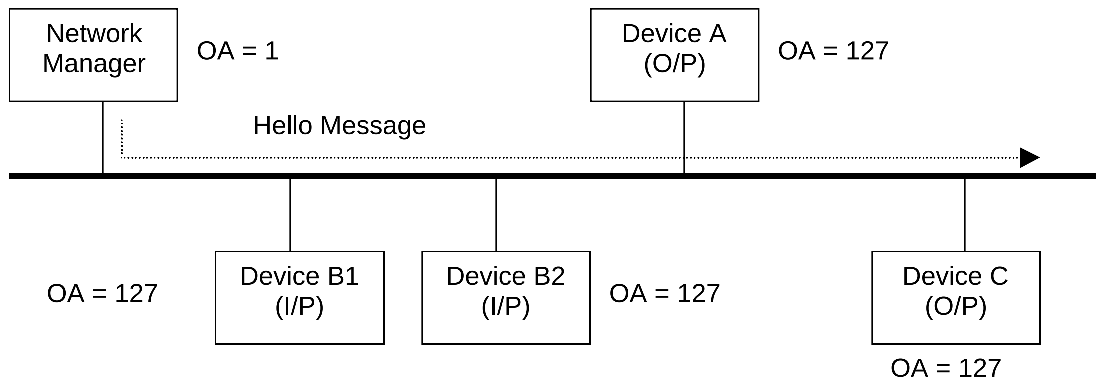

Figure 14 Network Manager joining the network

As seen in Figure 14, as soon as the Network Manager is connected to the network, it broadcasts the Hello Message twice using the pre-assigned OA, in this case, OA=1.

At this stage, some devices on the network may be able to detect the correct bit-rate from the two Hello Messages received. Otherwise more valid CAN messages need to be consumed before the correct bit-rate is successfully detected. This is done at the subsequent process.

1-3 Mini-who

After the Network Manager has successfully performed the network access, it starts to transmit the Mini-who Request Message periodically to locate all devices on the network.

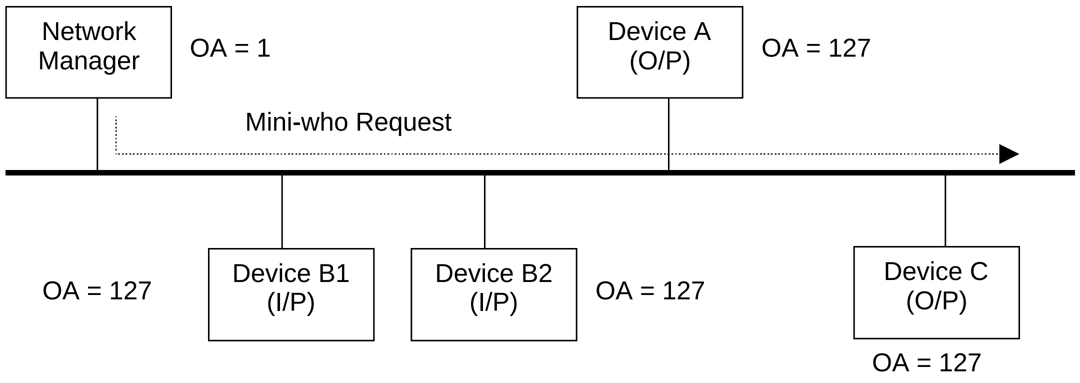

Figure 15 Mini-who Request

This activity injects messages onto the network. It there are devices on the network that have not detected the correct bit-rate successfully, this periodical message allows these device to continuously perform the detection process, until the correct baudrate is detected. After which, these devices will proceed to the next stage, i.e. broadcasting the Hello Message.

1-4 Hello Message

Because Device A, B1, B2, and C initially have OA of 127, they transition to the Standby State after broadcasting twice the Hello Message. At this stage, these devices can suspend themselves to the low-power standby mode until the ROAM message is received, which will be described in the next section.

The Network manager, upon receiving the Hello Massages from Device A, B1, B2, and C, performs the database matching against the information contained within the Hello Messages received. If no match is found, it can optionally report to the System Designer in order to build the database of the unknown device.

If match is found, the Network Manager will proceed to the next stage.

1-5 ROAM

Upon matching the devices' information with its database, the Network Manager will subsequently send the ROAM message to change the OA of each of these matched devices according to it database.

In this implementation example, assuming Device A, B2, and C are to be configured.

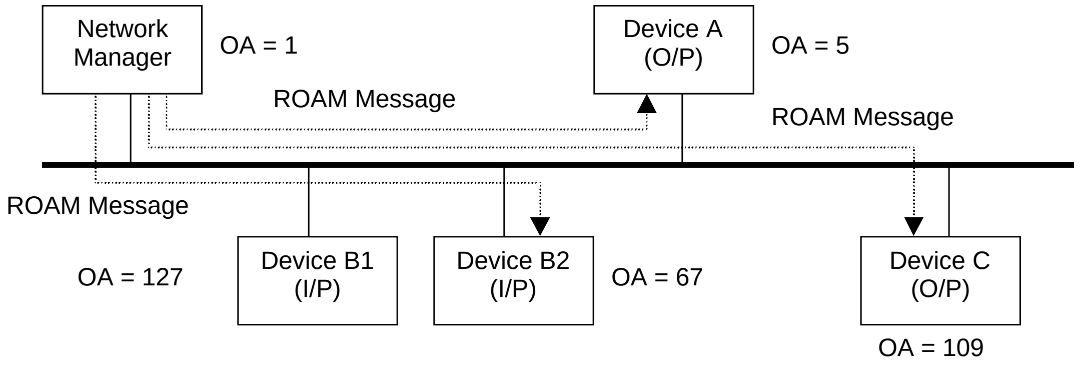

Figure 16 ROAM Message

Because ROAM is point-to-point, the Network Manager configures each of these devices at a time, as shown in Figure 16. After receiving the ROAM request, Device A, B2, and C reinitialise themselves and rejoin the network by broadcasting twice the Hello Message using the newly configured OA, i.e. OA 5, 67, and 109 respectively. After successfully broadcasting the Hello Messages, these devices transition to the Online State, and wait for the connection establishment request message or Connectionless Request messages.

1-6 Connectionless Request

If the Network Manager needs to gather more information about the devices, it can send the requests to the devices via the connectionless port, without having to establish any connections with these devices.

1-7 Connection Request/Response

Upon the detection of the Hello Message broadcast across the new OA, the Network Manager performs the database matching again. If match is found, it will try to establish the connection with the device using the Connection Request Message.

The connection establishment information (such as the produce/consume sizes, paths, etc) of each device is stored in the Electronic Data Sheets (EDS) that have been preloaded to the Network Manager's database. Assuming the following configurations:

Device B2 (input device) will produce a message that will be consumed by both Device A and C.

The Network Manager will establish the connection with Device B2, configure its producing connection instance. After which Device B2 starts to produce its input data across the newly created connection.

The Network Manager then tries to connect with Device A and C, and pass the producing information of Device B2 to them. This informs Device A and C to consume the input data produced by Device B2, as shown in Figure 17.

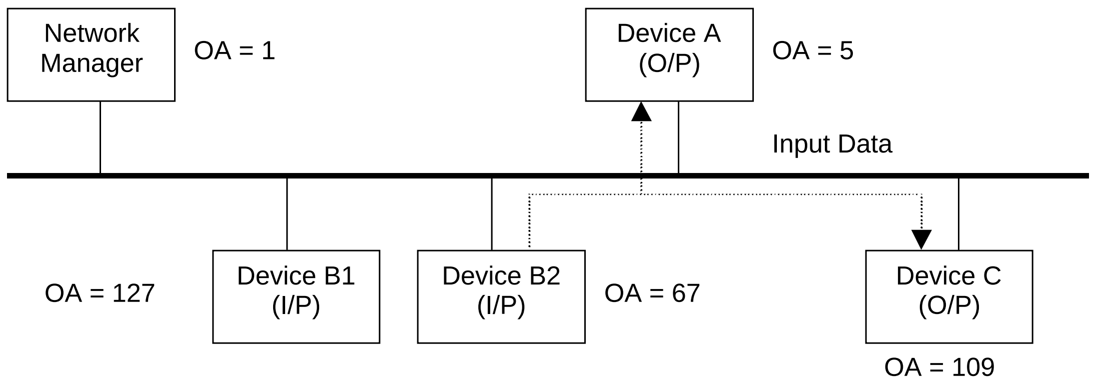

Figure 17 The exchange of I/O data between Device A and C

After the success configuration, Device A and C start to consume data produced by Device B2.

The Network Manager then becomes idle, listening and monitoring the connections created, until any timed out communication is detected.

1-8 Fully Auto Device Replacement

Assuming Device B1 is the backup device for B2. Since it has not been configured by the Network Manager, it has OA of 127 and at its Standby Stage, until waken up by the Network Manager.

If Device B2 is physically disconnected from the network or enters its Communication Fault or Bus-Off State, the Network Manager will detect the lost of the message from B2.

The Network Manager can optionally retry the connection establishment with Device B2. If this succeeds, B2 will continuously operating. If the retry attempts fail, the Network Manager will then look into its database to find a replacement device. If there is no replacement found, the Network Manager signals the System Designer about the lost of connection with Device B2.

If any replacement device is found from the database (i.e. Device B1), the Network Manager can optionally broadcast the Mini-who request to locate Device B1. Once Device B1 is located, the Network Manager will send the ROAM message to reconfigure B1's OA to 67, and set up B1's producing connection instance as what has been previously done on Device B2.

After the configuration, B1 starts to produce its input data that is consumed by Device A and C, as shown in Figure 18.

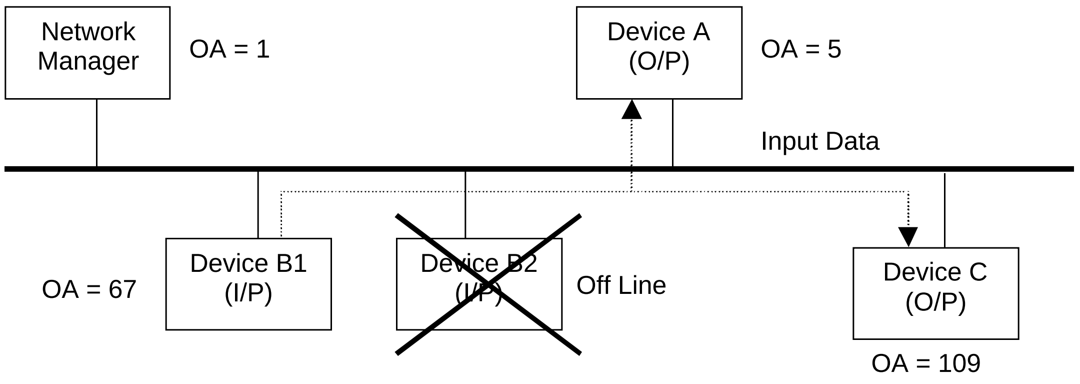

Figure 18 Auto Device Replacement

If Device A and C have infinite connection time (i.e. EPR = 0), they do not need to be reconfigured. Otherwise reconfiguration may be required. The Network Manager can send a request to Device A and C to find out the status of the connection before reconfiguring these devices.

6. Conclusion

This paper introduces the new CIP based protocol DeviceNeX in brief. This protocol is proposed as an economically viable methodology for overcoming the short falls of DeviceNet. It provides the solution to the DeviceNet disadvantages and also provides the following additional advantages:

- Larger node count
- Fully auto device replacement.
- More effective use of the network bandwidth
    - Peer-to-peer communication (no proxy required)
    - More data sent across a single frame

This protocol is defined in such a way that minimum effort is required to transform the already existing DeviceNet devices into DeviceNeX compatible, i.e. only requires firmware changes for many existing DeviceNet products. This is important if existing product vendors and network installations are to adopt such a protocol in the future.

As this is the beginning of this new protocol, there are still many areas to cover, as listed below:

- LED indicators behaviour
- Multiple physical attachments
- Routing technologies
- Repeater, bridge, gateway and router requirements and definitions.
- Remote frame utilisation
- Conformance and interoperability test.

More details can be found from the authors.

References

[i] Robert Bosch GmbH, “Bosch CAN Specification Version 2.0”, 1991.

[ii] Kiah Hion Tang, “A Study pf the DeviceNet Composite Test and the Definition of the DeviceNeX protocol”, Ph.D. thesis, June 2000.

[iii] Chris Quigley et al; "A Low Cost CAN Bus Transducer and Its Application", Proceeding of iCC2000, October 2000.