iCC 1997

4th international CAN Conference

in Berlin (Germany)

Sponsored by

Motorola Semiconductor
NEC Electronics (Europe)
Siemens Semiconductors

Organized by

CAN in Automation (CiA)
international users and manufacturers group
Am Weichselgarten 26
D-91058 Erlangen
Phone +49-9131-69086-0
Fax +49-9131-69086-79
Email:headquarters@can-cia.de
URL: http://www.can-cia.de

Mr. Don Pieronek

Advancements in Device-Level Networking

Abstract

The primary focus within the standards organization during the first two years of existence has been traditional control system configurations consisting of a single client (master) and up to sixty three (63) servers (slaves) on a single subnet. There are numerous reasons for this focus, much of which is related to the initial content of the specifications, the availability of product, the learning curve of member companies, and so forth. Open systems require a concise definition of interfaces and behaviors, all of which may not be reflected within even the best of specifications. Providing concise and sufficient levels of detail has been one of the primary goals of the standards organization. Additionally, Special Interest Groups (SIGs) have focused on defining and refining additional application objects for various types of devices. These include simple I/O points, various sensors, actuators and even specialized subsystems like motion control.

A goal of many of the participating vendors is an "Open Control System", whose major beneficiary is the end user and system integrator. A small vendor may develop niche market products which may be configured and diagnosed using other vendors' tools.

Definition of Terms

The Open DeviceNet™ Vendor Association (ODVA) strives to clarify and extend its object oriented architecture & protocols. One goal is to create implementation independent product interfaces, thus providing system interoperability while allowing different vendor products to be interchanged.

Prior to describing the technologies of ODVA, a short definition of terms is useful.

Network: **A** collection of one or more subnets. A domain is defined to be a system containing multiple networks.

Subnet: **A** collection of nodes using a common protocol and shared media access arbitration. A subnet may have multiple physical segments and may contain repeaters. A subnet may contain up to sixty four (64) nodes. Additionally, a segment is a collection of nodes connected to a single uninterrupted section of physical media.

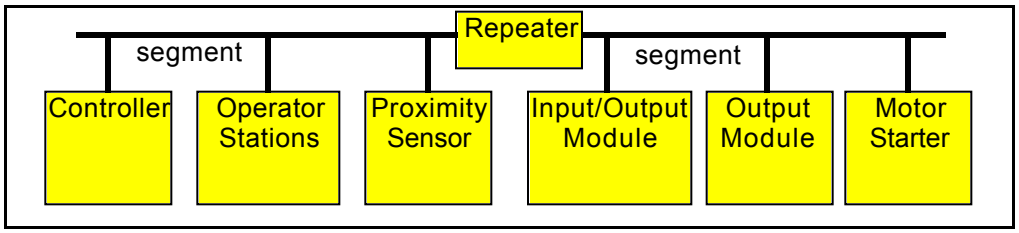

Figure 1: Subnet of Nodes

Node: A collection of objects which communicate over a subnet, and arbitrates using a single Media Access Controller Identifier (MAC ID). A physical device may contain one or more nodes.

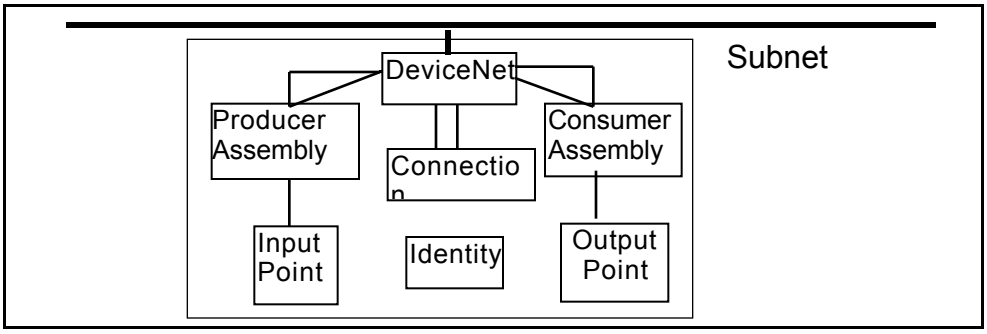

Figure 2: Objects within a Node

Object Classes

Nodes contain various object classes. Within the standard, a node shall contain a minimum of three objects; DeviceNet Object, Identity Object, and Connection Object. All other objects are optional. The standard defines public objects and allows vendor specific objects. Each object class inherits a common set of class attributes and instance attributes. Each class has a unique Class ID. Each class defines the required, optional and conditional attributes. When a vendor implements a given class, all required attributes must be implemented. It is a product decision whether or not the optional or conditional attributes are also implemented. Each class defines a set of services, which are either required, optional or conditional. Each service of a given class defines its request protocol and all possible response protocols. A set of common services are defined which are independent of class, as well as allowing class specific services.

If a client knows the classes contained within a node, it also knows the maximum set of attributes, services and behaviors the node supports. A list of these classes may be contained within an optional attribute of the Message Router Object. Currently, the standard does not explicitly support inheritance, since each Class ID is unique. However, from an OO purist's point of view, the standard defines the subclass with the greatest functionality, where pieces of the functionality are optional. This helps preserve the 65535 allowed Class ID's.

Instances

A node may contain up to 65535 instances of each object class. The first instance is always identified as one (1). Instance zero (0) is reserved for dialogs with the class. In other words, a service like "Get Attributes" may be sent to an object class or an object instance.

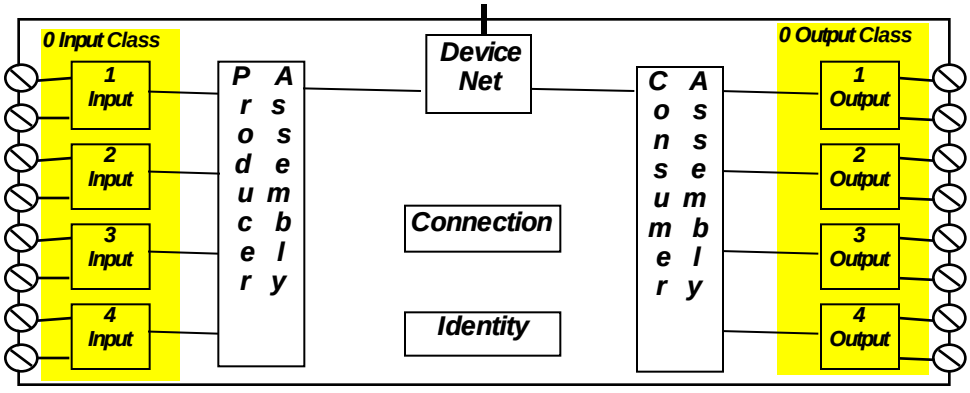

Figure 3: Object Instances

As shown in *Figure 3: Object Instances*, four (4) instances of *Input Point* and *Output Point* are shown. Each instance of *Input Point* provides their *value* attribute to the subnet via a *Producer Assembly* class. A *Consumer Assembly* class provides I/O Message data to each instance of the *Output Point* objects.

Attributes

An object class within a node contains various attributes (properties). The data type of an attribute and its allowed value(s) are defined within the class profile.

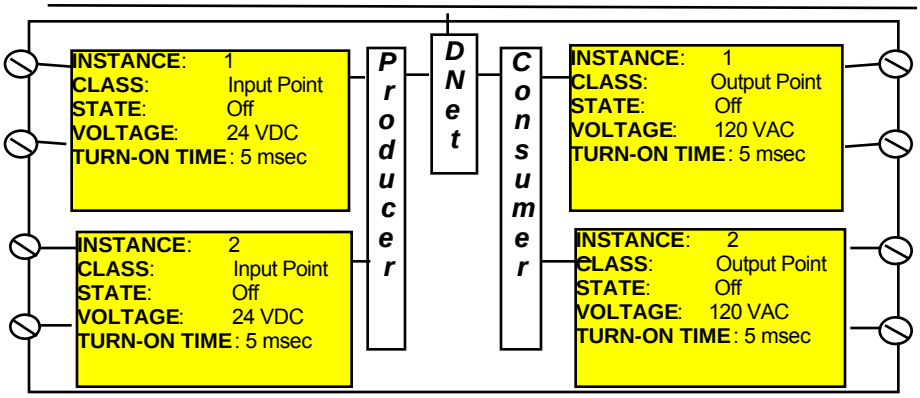

Figure 4: Object Instance Attributes

Figure 4 indicates some attributes of two classes, the *Input Point* and *Output Point*. Although the figure indicates ASCII names for the attributes, each attribute is actually assigned a unique number within each class profile. The standard allows up to 255 instance attributes and 255 class attributes per class.

Behavior

An object's behavior is determined by its current state and the occurrence of events. Each object is described by a profile which defines the object's states, events that cause transitions between the states, and any actions taken during the state transitions.

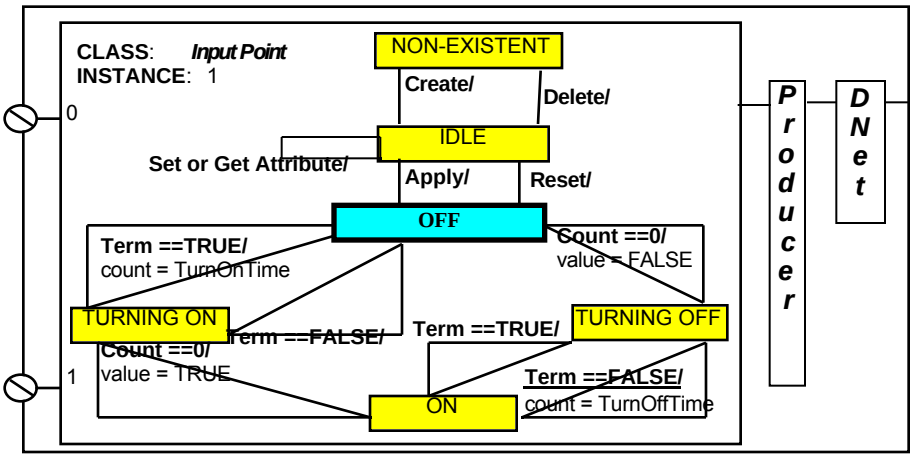

Figure 5: Object Instance Behavior

The *Input Point* in Figure 5 supports both internal and external services which trigger transitions. Assume the instance shown is unused by a client and would therefore be in the "Non-Existent" state. If a client sent a "Create" request message to the *Input Point* class, instance one would transition to the "Idle" state. The node would then reply with "instance one (1)". While in this state the client would be allowed to set the various attributes of this instance. Upon completion of the appropriate configuration, the client would send an "Apply Attributes" request message to instance one. Assuming all attributes are valid, the instance would transition to the "OFF" state. At this point, the state of the *Input Point* is determined by the state of its input terminals.

Message Protocols

Various message protocols currently defined within the specifications include:

1. Explicit Messages,
2. I/O Messages,
3. Duplicate MAC ID Messages,
4. Unconnected Messages,
5. Heartbeat Message, and
6. Recover Faulted Node Messages.

Explicit Messages are generally used to interrogate, configure, and perform diagnostics on nodes. These interpreted messages direct services to a specified instance of an object class. The following protocol is used:

> *Fragmentation Flag[**0**], Transaction ID, MAC ID, Service, Class, Instance, Service Data*

Although the CAN hardware supports an eight (8) byte data packet, the protocol supports fragmentation and re-assembly, allowing messages of any length to be conveyed. When fragmentation is required, the following protocol is used;

> *Fragmentation Flag[**1**], Transaction ID, MAC ID, Fragment Type, Fragment Count, Service, Class, Instance, Service Data*

I/O Messages of less than eight bytes in length do not contain a specific protocol. These messages are analogous to compiled systems, where the nodes have prior knowledge as to the structure and meaning of the data. I/O Messages may be “fixed” or “configurable”. As with the Explicit Message protocol, I/O Messages also support fragmentation and re-assembly, allowing messages of any length to be conveyed. The production of I/O Messages may be triggered by the receipt of a poll request, a bit strobe request, a cyclic timer event, or a change in state of an application object within the node. The setting of an attribute of the connection object determines the I/O Message behavior.

Duplicate MAC ID Messages are used when a node powers up on a subnet and is attempting to gain ownership of an address (MAC ID). The protocol and its respective media access state machine guarantee that only one node may reside at a given address on the subnet. This message contains a nodes’ Vendor ID and Serial Number, which is unique within the system.

Unconnected Messages are optional and are used to open explicit message connections between two peer nodes.

Heartbeat Message is optional and is used to broadcast a node’s configuration and fault status. It is broadcast periodically or upon a change in state.

Recover Faulted Node Messages are optional and are used to: 1) detect the existence of a faulted node on a subnet, 2) to physically identify the faulted node, and 3) change the address of the faulted node to bring it on line.

Control Paradigm

Various I/O Message behaviors allow multiple control system paradigms to be implemented. These control paradigms include, but are not limited to;

1. single client (master) systems,
2. multiple client (master) systems,
3. peer to peer systems,
4. distributed control systems,
5. redundant systems,
6. any combination of the above, and
7. others yet to be discussed.

The standard allows any combination of centralized, peer, and redundant systems to reside on the same subnet at the same point in time. In fact, the standard divides the 4031CAN identifiers into four groups;

Group 1 allocates sixteen (16) interleaved CAN identifiers for application usage,
Group 2 reserves two Duplicate MAC ID identifiers and allocates six (6) interleaved CAN identifiers for application usage,
Group 3 reserves two Unconnected Message identifiers and allocates five (5) interleaved CAN identifiers for application usage, and
Group 4 is reserved for system administration messages, like Recover Faulted Node Messages.

In general, the Group 2 message identifiers are used for client/server configurations, the Group 1 identifiers are used for peer control I/O Message communications, and the Group 3 messages are generally used for system diagnostics and monitoring applications. One unique feature is the method in which the identifiers are allocated to nodes. Message identifier assignments allow higher priority control messages media access prior to lower priority control messages, thus providing a system tuning capability not possible with other control networks.

Single Client Multicast Subsystems

Although a traditional round robin request/response message dialog may be used to communicate between a client (master) and its server (slave) nodes, an optional multicast protocol may be used to eliminate the overhead required when producing individual messages for each server. As shown in Figure 6, a single multicast message produced by a client may be consumed by multiple servers. Upon consumption the servers extract their information from the message.

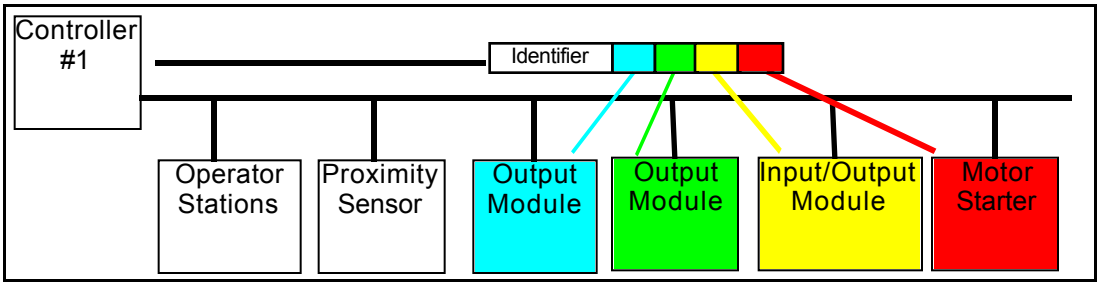

Figure 6: Multicast Master/Slave

If the optional *Dynamic Assembly Object* was implemented within a node, it would be required to consume up to an eight byte message for the multicast configuration versus only a single byte for a polled request.

Multiple Client Subsystems

As shown in Figure 6, a subnet may contain any combination of client nodes and server nodes.

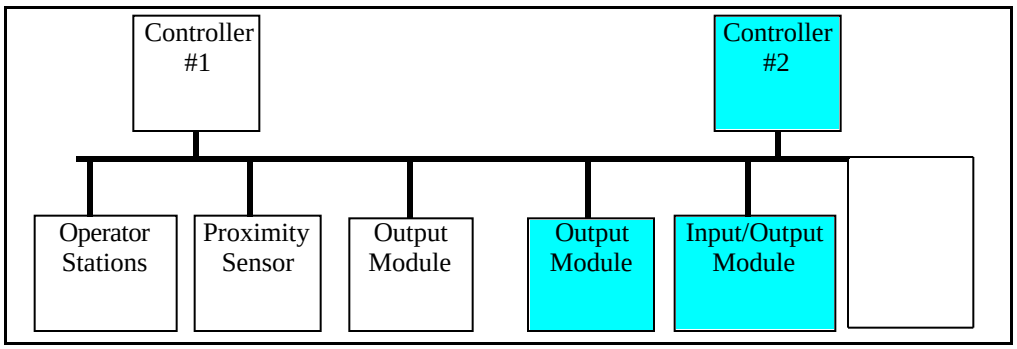

Figure 7: Multiple Master Subnet Configuration

Peer to Peer Subsystems

Various types of systems may be constructed which take advantage of the peer communications capabilities of the network.

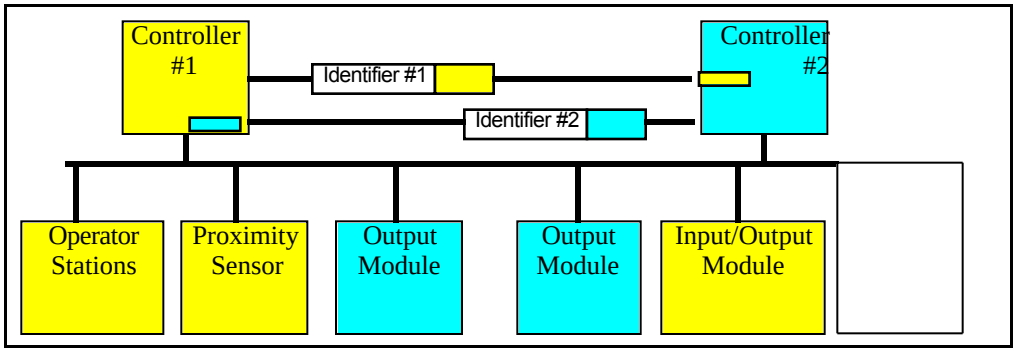

Figure 8: Peer Control Configuration

Figure 8 identifies two nodes, Controller #1 and Controller #2, which exchange control information. The information passed will vary, depending upon the application programs. Common scenarios include machine interlocks, operator information, data logging, and so forth.

Peer communications may also exist between nodes of significantly less functionality than two controllers. Although a common paradigm in the power distribution and control industry today, this previously vendor specific functionality will be one focus of ODVA during the coming year.

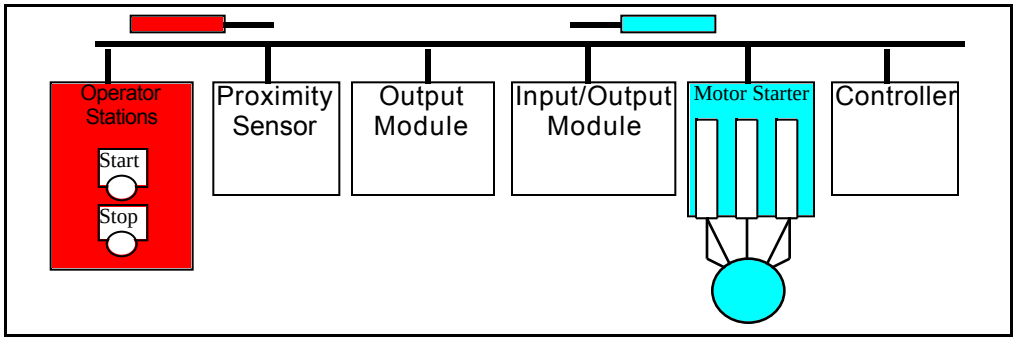

Figure 9: Distributed Control Configuration

Typical scenarios include an Operator Station and Motor Starter, where an I/O Message produced by the Operator Station is consumed by the Motor Starter, and an I/O Message produced by the Motor Starter is consumed by the Operator Station. Although this functionality has been available for years in power distribution networks, an open interface shall be defined consistent with the ODVA standard. Various control paradigms are possible to provide the prior solution.

1. The Operator Station could function as a client to the Motor Starter, or
2. the Motor Starter could function as a client to the Operator Station, or
3. the Operator Station and Motor Starter could function as peers.

Change of State Behavior (Cyclic)

Prior to the definition of Cyclic Change of State (COS), server nodes using the client/server paradigm had to be polled to convey I/O Messages. In general, a client node would produce an output message and consume the servers input response message. Using COS, the time required to read all inputs and write all outputs (scan time) is no longer dependent upon the number of server nodes and the size of their I/O Messages. I/O Messages are only sent when their values change. The current technique states that once a node produces a COS message, it is not allowed to send another message until an internal “production inhibit timer” expires. This is intended to prevent two nodes, whose input values are changing rapidly, from producing messages back to back and consuming the total subnet bandwidth.

New System Features

Some of the features added during 1996 include:

1. Faulted Node Recovery,
2. Device Heartbeat,
3. Selector Object (Redundant Systems),
4. Common Configuration attribute, and
5. Improved Conformance Test Suites and Independent Test Sites.

Faulted Node Recovery

Prior to the definition of this functionality, if a node failed its media access arbitration due to duplicate node addresses residing on the same subnet, it was necessary to physically disconnect a node from the subnet to change its address. An enhancement now allows a faulted node's address to be changed directly over the subnet.

Device Heartbeat

Although various object classes are defined, there was no common way to determine the health of a node. An optional message has been defined which contains detected error information, a configuration value, and the node's state. This broadcast message may be consumed by any node within the system and used to diagnose and repair faults, back up and restore systems, and so forth.

Redundant Subsystems

This functionality was defined so that high reliability systems can be constructed. These systems often contain multiple redundant nodes. Should one node fail, a redundant node will take over. The optional Selector Object defined four redundancy algorithms to meet this market need. Various types of redundant systems are possible using the Selector Object, like *Figure.* Here, the system contains two (2) redundant input nodes.

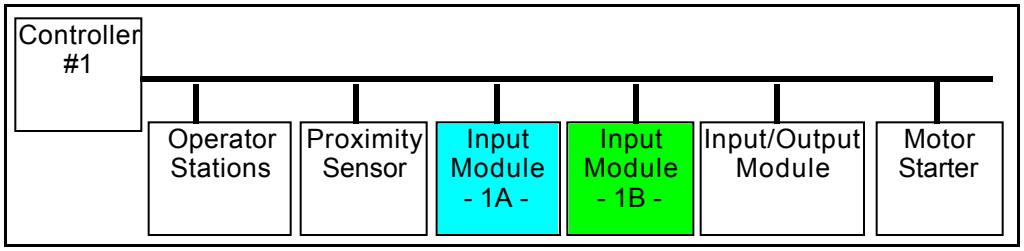

Figure 10: Redundant Input Node System

The client node (Controller #1) contains the Selector Object, which would use input data from either Input Module 1A or Input Module 1B, depending on their health.

Common Configuration Attribute

The specification allows the configuration of any node within the system, by any client within the system. However, should the configurations of client and server be inconsistent, it was difficult to detect. An attribute was added to the Identity Object to allow detection of configuration inconsistencies. A vendor may implement a configuration consistency check using any technique, including:

1. incrementing the attribute whenever a Set Attribute is processed,
2. calculating a checksum on all non-volatile attributes, or
3. calculating a cyclic redundancy check on all attributes.

When using a simple checksum algorithm, if one attribute is changed from four (4) to eight (8) and another attribute is changed from eight (8) to four (4), the checksum would not indicate that the configuration has changed. If the configuration attribute is incremented each time any attribute is changed, modifying and returning an attribute back to its original value will be detected as a change, when in fact the configuration remained the same. This is preferred to the simple checksum since there is at least an indication of a possible configuration change.

Conformance Test Suites and Independent Test Sites

Significant effort has been undertaken to create and extend test plans and related test software to verify conformance with the specifications. As with any test software, it will never be complete, but will continue to be improved. Although various vendors have test plans which exceed those of the standards organization, they are generally donated to create the most robust testing possible. Future tests shall include physical layer tests, power supply tests, numerous timing tests, in addition to extending both object and protocol tests. The goal of the Conformance Tests are to guarantee that ODVA members provide the most robust products available.

Future Direction

Some features under consideration for 1997 include:

1. Node Timing Characterization,
2. Change of State (Event Synchronized),
3. Common Peer Communications interfaces, and
4. Common Control Programming interfaces.

Node Timing Characterization

Customers have discovered that node response times vary greatly. When delays are excessive, system faults often result. Identifying these delays, and understanding their impact on system operation, is currently being addressed. These delays are incurred due to various aspects of a node's implementation.

Although not explicitly shown in Figure 11, delays may be incurred due to various aspects of a node's implementation.

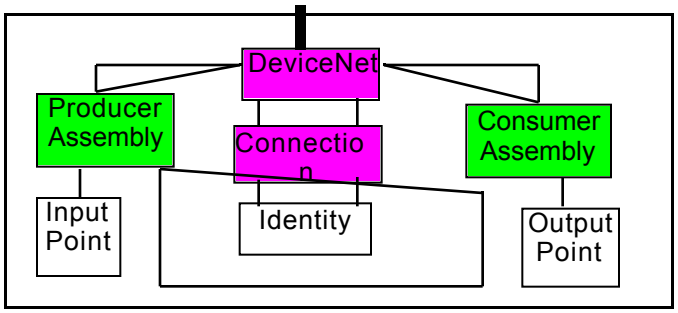

Figure11: Internal Node Timing

Upon receipt of a message, a finite time is required to check and reject a message. This is referred to as the *reject time*. Some nodes perform this message rejection in hardware, using zero CPU time. If the message is: 1) accepted and processed by the node, 2) the media is not busy and 3) the response message is the highest priority message ready to send, it shall be produced on the subnet. The time to service the message and post a response is referred to as the *response time*. There are two categories of response times: one for I/O Messages and one for Explicit Messages.

System faults occur when the message load presented by the subnet exceeds the node's ability to service the messages. The available message service time is based upon various parameters. A message packet may contain zero (0) to eight (8) data bytes. The smaller the message, the faster the delivery rate. Since three baud rates are supported, the arrival rate for the same message increases with the baud rate. Keeping up with the presented subnet traffic is commonly referred to as "drinking from the fire hose".

Many vendors' products operate at all three currently defined baud rates. However, as with any network, these higher baud rates do not generally improve the message processing time. When the baud rate is doubled, a message is delivered in half the time. If the time to accept or reject a message is sufficient at 125K, it may not be sufficient at 250K and/or 500K. If such a scenario exists, messages may be missed. This is a very difficult fault to detect, especially when it occurs under spurious subnet loading scenarios. ODVA is investigating tools and techniques to detect and resolve these real world application issues.

More specifically, at 500K a zero length message consumes 94 microseconds on the subnet, while an eight byte message consumes 222 microseconds. Therefore, if a message is consumed, and an eight byte message is being produced on the subnet, the slave node has 222 microseconds to accept or reject the prior message. If the second message produced is zero bytes in length, the slave node may have only 94 microseconds to accept or reject the prior message. Additionally, explicit or fragmented messages contain a variable within the data portion of the message which must be checked. This further increases CPU accept/reject processing time.

Due to the inability to "drink from the fire hose", test plans shall be developed which will determine the "bandwidth" of nodes using various subnet message scenarios. This information may be published or become optional attributes within a node for use by system configuration tools. Ideally such configuration details will be transparent to the user.

As should be apparent, open systems are constantly striving for "continuous system improvement". Due to competitive pressures, through time, open standards will clearly become the most robust, and most easily diagnosed systems within the automation industry.

System Timing Characterization

System performance is determined by message and node execution delays. Figure 11 lists five nodes down the left column. The rectangles with solid outlines indicate messages produced by the respective node. The shaded areas indicate *application object execution delays*. Although various message orders are possible, the timing diagram indicates one possible "Polled" configuration.

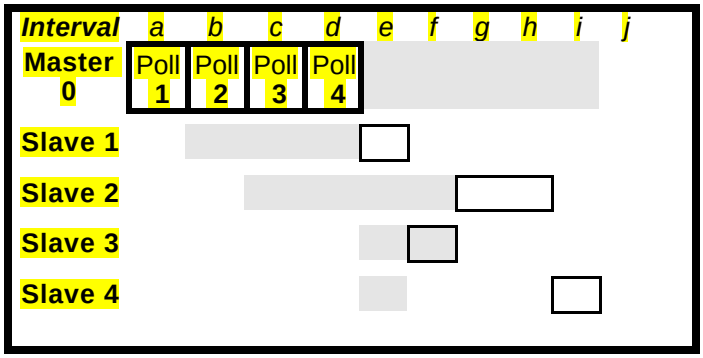

Figure 11: Polled Subnet Timing

Assuming all messages contain two data bytes at 500k, the following table lists the sequence of events and timing for each interval:

Interval	EVENT	TIME
a	Master 0 performs a "Poll request" to Slave 1	126
b	Master 0 performs a "Poll request" to Slave 2	126
c	Master 0 performs a "Poll request" to Slave 3	126
d	Master 0 performs a "Poll request" to Slave 4	126
e	Slave 1 performs a "Poll response" to Master 0	126
f	Slave 3 performs a "Poll response" to Master 0	126
g-h	Slave 2 performs a "Poll response" to Master 0	126
I	Slave 4 performs a "Poll response" to Master 0	126
j	Master 0 performs some housekeeping	0
	Total Scan Time (microseconds)	1008

The timing diagram indicates that a slave responds when ready, not necessarily the order in which it was polled. Additionally, network utilization nearing one hundred percent is possible utilizing this technology. As shown, one millisecond scan times are theoretically possible with this five node system.

Most systems today would minimally exhibit a *response time* delay at *interval j* prior to the next I/O scan. This is where the client (scanner) performs some internal housekeeping. Additionally, a scanner may be configured to leave a short interval here to allow production of diagnostic messages, duplicate MAC ID messages, and so forth.

Cyclic Change of State-Centralized Control

The current Cyclic Change of State technique offers performance improvements over standard polled systems. This improvement will be realized when a subnet contains numerous slave nodes, of which only a few change state within a specified period of time. However, assume a system configuration includes only a few nodes, like the five node system shown previously.

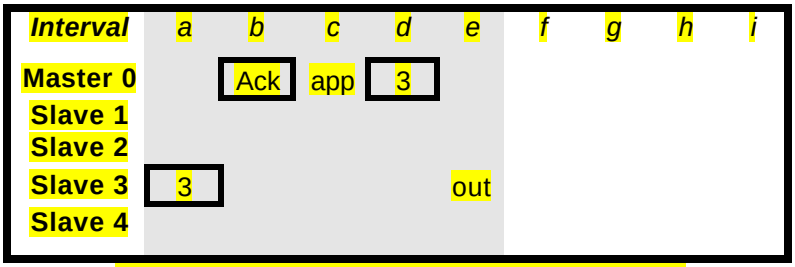

Figure 12: Polled Request/Response Timing

Assume Slave 3 produces an COS input message at *interval a,* to which Master 0 produces an zero length acknowledgment message at *interval b*. The logic program executes during *interval c*, followed by an COS output message at *interval d*, which is processed by Slave 3 during *interval e*, possibly produing a zero length acknowledge message. The user may assume a sub-millisecond response time will always be achieved. However, assume all the slaves are typical micro-controller implementations, having a 10 millisecond internal timer tick. Since such timers are generally used for multiple purposes within a node, like connection time outs, the internal clock runs asynchronous to the node firmware. Therefore should the COS "production inhibit timer" get set to one upon receipt of an I/O Message, just prior to the time tick, it could expire within a matter of a few microseconds. If set just after the internal time tick, it may expire 10 milliseconds later.

Assuming two byte I/O Messages at 500K baud, the time from *interval a* to *e* is;

Interval	EVENT	TIME
a	Slave 3 performs a "COS production" to Master 0	126
b	Master 0 performs a "ack response" to Slave 3	94
c	Master 0 executes logic program during 'ack"	0
d	Master 0 performs a "COS production" to Slave 3	126
e	Slave 3 performs a "ack response" to Master 0	94
	Total Scan Time (microseconds)	440

Figure 13: COS Response Time

The end result is that once a Slave node produces a message, the loop response time will be rapid. However, it may not produce another I/O Message for 10 to 20 milliseconds. The same system using the "polled" configuration guarantees a loop closure time of less than two (2) milliseconds, which is considerably faster! Therefore, for this scenario, until the polled configuration exceeds a scan time of twenty milliseconds, the Cyclic Change of State configuration is of little value. Although costly, if slave nodes had a timer tick interval as low as one millisecond, the Cyclic Change of State configuration would be of value. This fundamental system behavior may not be intuitively obvious to many customers.