Compliance and Functional Testing of IEEE 1451.1 for NCAP-to-NCAP Communications in a Sensor Network

D. Gurkan, *Member, IEEE,* X. Yuan, *Member, IEEE,* D. Benhaddou, *Member, IEEE,* H. Liu, A. Singla, *Student Member, IEEE,* R. Franzl, *Student Member, IEEE,* H. Ma, S. Bhatt, J. Morris, M. Turowski, and F. Figueroa

Abstract

Distributed control in a networked environment is an irreplaceable feature in systems with remote sensors and actuators. Although distributed control was not originally designed to be networked, usage of off-the-shelf networking technologies has become so prevalent that control systems are desired to have access mechanisms similar to computer networks. However, proprietary transducer interfaces for network communications and distributed control overwhelmingly dominate this industry. Unless the lack of compatibility and interoperability among transducers is resolved, the mature level of access (that computer networking can deliver) will not be achieved in such networked distributed control systems. Standardization of networked transducer interfaces will enable devices from different manufacturers to talk to each other and ensure their plug-and-play capability. One such standard is the suite of IEEE 1451 for sensor network communication and transducer interfaces. The suite not only provides a standard interface for smart transducers, but also outlines the connection of an NCAP (network capable application processor) and transducers (through a transducer interface module – TIM). This paper presents the design of the compliance testing of IEEE 1451.1 (referred to as Dot1) compatible NCAP-to-NCAP communications on a link-layer-

independent medium. The paper also represents the first demonstration of NCAP-to-NCAP communications with Dot1 compatibility: a tester NCAP and an NCAP under test (NUT).

A. Introduction

The Instrumentation and Measurement Society's Sensor Technology Technical Committee TC-9 in the Institute of Electrical and Electronics Engineers (IEEE) and the National Institute of Standards and Technology (NIST) have been working to establish a suite of smart sensor interface standards under IEEE 1451 since late 1990s. The objective of the family of standards is to define a set of common communication interfaces for connecting transducers (sensors or actuators) to microprocessor-based systems, instruments, and field networks in a protocol-independent environment [1]. Transducers are used in a wide variety of applications in manufacturing, industrial control, automotive, aerospace, building, and biomedicine. Although transducer applications are very diverse; desirable features such as low-cost, networked, and intelligent are common to all systems. Many control network implementations are currently available, each with their own strengths and weaknesses for specific applications [2]. Typically, one manufacturer's system would deliver a black box solution of networking and access mechanisms as shown in Fig. 1 without any guarantee of interoperability with similar

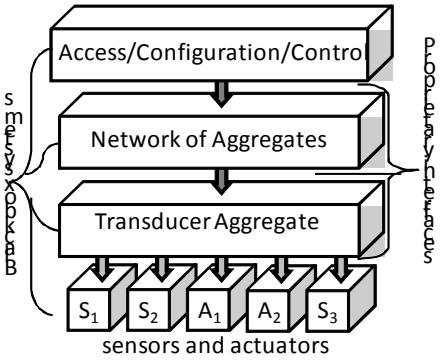


Figure 1. Current proprietary approach is based on complete solutions all the way up to the user access with no room for interoperability between components from different manufacturers.

components made by other manufacturers.

On one hand, interfacing the smart transducers to control networks using a wide variety of standards *requires significant effort* and *upfront expense* to manufacturers. On the other hand, usage of digital communication schemes and networked transducers can *eliminate a large number of lengthy wiring* [3], *enhance access mechanisms to measurements*, *ease configuration capabilities*, and thus *reduce the cost of installation, maintenance, and upgrade of sensor-based systems* [4, 5]. As a result, the emergence of interoperability standards will deliver networking interfaces. And consequently, competition among vendors will be focused on the innovation in component design and manufacturing without defining new network interfaces.

IEEE 1451 standard suite's goal is to make it easier for transducer makers to develop smart devices that can interface to networks, systems, and instruments using existing and emerging sensor and networking technologies [6, 7]. A network-neutral functional data model for a network-capable application processor (NCAP) has been standardized in IEEE 1451.1 (will be referred to as Dot1 from here on) [8]. (At the time when this work has been finished, Dot1 has been put into draft status by the standardization committee to make revisions based on the overarching IEEE 1451.0's release. Revisions on Dot1 will change some operations. However, the authors believe that the work presented here as testing for compliance will remain still applicable.) Dot1 enables an NCAP to be accessed using an application layer protocol to configure transducers, access/retrieve measurement data, and interface with other network devices (e.g. other NCAPs) on a control network. This paper presents the compliance testing implementation of Dot1-compatible NCAP-to-NCAP communications on a link-layer

independent medium [9, 10, 11]. Testing of STIM modules using a virtual NCAP implementation has been demonstrated in [12]. Complete hardware implementation of STIM and NCAP on a CAN network has been demonstrated [13] with limited control functionality on the Dot1 NCAP. Web services have been implemented as an application on an NCAP framework over .NET with transducer access and without NCAP-to-NCAP communication components [14, 15]. Although there has been also simpler implementations of the NCAP [16], this work represents the first prototype of NCAP-to-NCAP communications with Dot1 compatibility: a tester NCAP and an NCAP under test (NUT). The paper is organized as follows: A brief overview of Dot1 with an introduction to NCAP-to-NCAP communications will be given in Section B. Then, a general overview of compliance testing will be presented. The details of the functional and conformance testing procedure design of Dot1 components will be given in Section D. Results from the implementation of the NCAP testers are presented in section E.

B. IEEE 1451.1 (Dot1) Network Capable Application Processor

IEEE 1451 networked transducer interface standard provides a framework for two levels of plug and play [17]: (1) networking interface to transducers equipped with signal conditioning and data sampling (namely the transducer interface module – TIM); (2) networking interfaces with other network devices, intelligent algorithms, and data transfer protocols (namely the network capable application processor – NCAP). Following the standards, transducers, TIMs, and NCAPs can be manufactured by different vendors while remain interoperable [18]. Fig. 2 illustrates the interoperability goal.

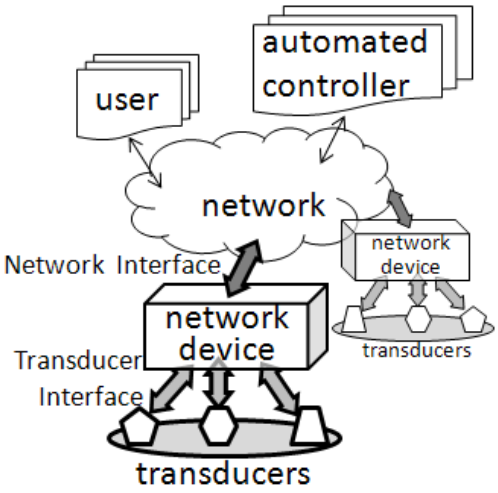


Figure 2. Transducers and network devices can be manufactured by different vendors – as long as there is a common interface, they can interoperate to deliver data to the users or automatic controllers through a network.

IEEE1451.1 is an application layer protocol that defines a common object model to describe how the NCAPs access smart transducer nodes with object interfaces. The objectives of such a model are to: (i) streamline the measurement, reconfiguration, and networking processes; (ii) enable plug-and-play of transducers; (iii) host intelligent algorithms for data fusion/processing before information flows towards the central controller (iv) ensure software portability among different hardware and software technologies. Complying with Dot1 will result in a distributed measurement and control system where NCAPs enable the networking among smart transducers. An NCAP can be connected to one or more number of transducers. Dot1 initiates and manages data transfer from/to transducers, and configures features of the measurement. In this respect, NCAP-to-NCAP communication forms an intelligent control framework for data access and intelligent decision making. NCAP is regarded as a network device with network identification and communication requirements for access and also a gateway to the connected transducers. In this respect, the compliance testing of the Dot1 guarantees the interoperability of two NCAPs from different manufacturers on the same network.

C. Overview of Compliance and Functional Testing

Implementations of a standard typically vary among manufacturers. This makes testing for standard's compliance a crucial component of the standardization process. Compliance with the standards guarantees that the components will operate seamlessly even if they are built by different manufacturers. Typically, a special test instrument connects to a device under test (DUT) to verify its compatibility. For example, the range of resistance for a resistor would be tested using a sensitive ohmmeter (namely, a measurement device as a tester) which can be specified in the standard. A networked device can be tested using the software platform and the governing protocols.

Particularly, an interoperability test of a component in a sensor network platform requires a network connection, a communication protocol agreement between the tester and the component, and a set of operations that provides a specific level of compliance. Interoperability in hardware is based on connection types such as serial (USB, RS-232), parallel, and Ethernet, etc. In software and networking, interoperability is realized when components can speak to each other. The DUT in an interoperability test should work with other manufacturer's devices on the same setup. The compliance to the standard for interoperability has two main thrusts: functional compliance such as the operational states; and conformance compliance such as the format of communication through the network as outlined in a framework among the manufacturers. Although a software-level test of function calls is possible, operational states alone are not sufficient to achieve the interoperability goals. If the DUT cannot perform operations through its communication with other compliant units, a standalone operational unit will not demonstrate interoperability. In this respect, the two compliance thrusts are complementary and

should be tested simultaneously. However, not all test failures can be differentiated between compliance to a protocol versus an operational error.

In the case of an interoperability standard such as the IEEE 1451 standards suite, the network interface with the rest of the sensor network and the control functions with transducers are performed by the network-capable application processor (NCAP). According to the standard, two levels of interoperability are designed for an NCAP to achieve a plug-and-play system of transducers on a sensor network: NCAP-to-TIM and NCAP-to-NCAP. This paper is focused on the NCAP-to-NCAP communications. The operational states of an NCAP are tested through functional compliance. The conformance is tested with respect to a pre-defined format for the application layer packets. More details on the packet format will be given in section D.

D. Testing Procedures of NCAP-to-NCAP Communications in Dot1

Dot1 defines neither an application layer packet format nor a byte-level physical layer format. They are both defined by the manufacturer and left to the specific requirements of the application. For example, the application for this paper involves transmission through a TCP/IP (Transmission Control Protocol/Internet Protocol) network using XDR (external data representation [19]) as a data format language. The operational transmissions related to the transducers, configuration of sensor data retrieval methods, and NCAP identifications are handled through the application layer packets. Packet formats are defined accordingly with addressing information in a header, followed by the operation identifier, and the result of an operation. While addressing, operation identifiers, and data types are compliant to Dot1, the position and format of this

information in a packet depends on the implementation [20]. Dot1 defines the data types of addressing elements such as the "object id", "object tag", and "object dispatch address." Operation identifications such as the "GetTEDS", "SetTEDS", and "GoActive" are also listed in Dot1 as unsigned integer data type with 16-bit length. The ranges of operation id's are listed in [8: Table 7, page 55]. The operational state machine of an NCAP Block and the Function Block enable various operations to be functional or non-functional as shown in Fig. 3. An operation can be functional in that it will return correct results for the performed operation. It can be non-functional if it returns a "return code" equivalent to "unavailable operation." This paper presents the reference implementation from the revised on-the-wire document [20]. Section D.1 will describe functional testing procedures for all operational states of an NCAP. Then, section D.2 will explain the conformance testing of an NCAP according to packet formats and the specific implementation.

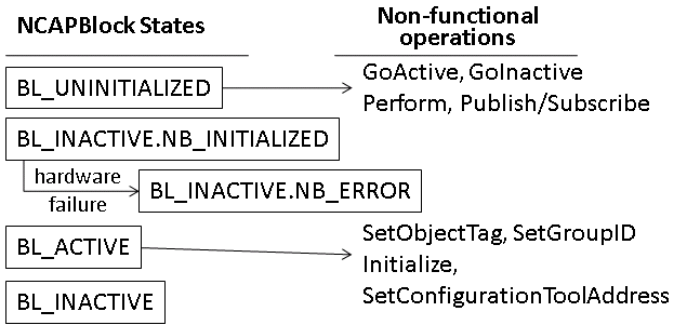
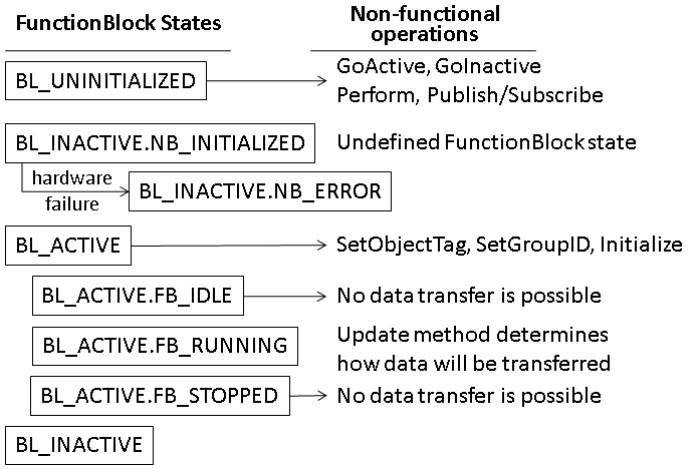


Figure 3. (a) NCAP block states and the corresponding non-functional operations. When NCAP block is active, addressing operations such as SetObjectTag is not operational to prevent conflicts.



(b) Function block states and the non-functional operations. Data transfer is possible through the specified update method only when the NCAP block is active and the function block is running.

D.1. Functional Testing of Dot1

Table 1 lists the functional phases and corresponding operations of an NCAP. As a network device, NCAP is expected to plug-and-play: that is, when connected to a network, it can identify itself to other networked devices through announcements. A dynamic announcement configuration guarantees that the NCAP will announce itself immediately after connection to a network. On the other hand, static configuration waits for a request and then sends back its identification in an announcement packet. These processes fall under the *node discovery phase*. Once an NCAP has been introduced to the network, its network visible objects can be accessed by directly addressing the NCAP. Therefore, announcement-related messages are multicasted using publication keys and UDP (user datagram protocol) whereas network visible objects of a particular NCAP are requested through a point-to-point connection, via TCP (transport control protocol). The Node Discovery section in Table 1 lists the multicasted publication and subscription messages in upper case and point-to-point client/server messages in lower case.

Remaining functional phases address the NCAP through client/server messages except a multicasted publication of data readings from a sensor during *data collection phase*.

Table 1. Functional phases of an NCAP and a sample of corresponding operations. Note: PSK stands for publish-subscribe key.

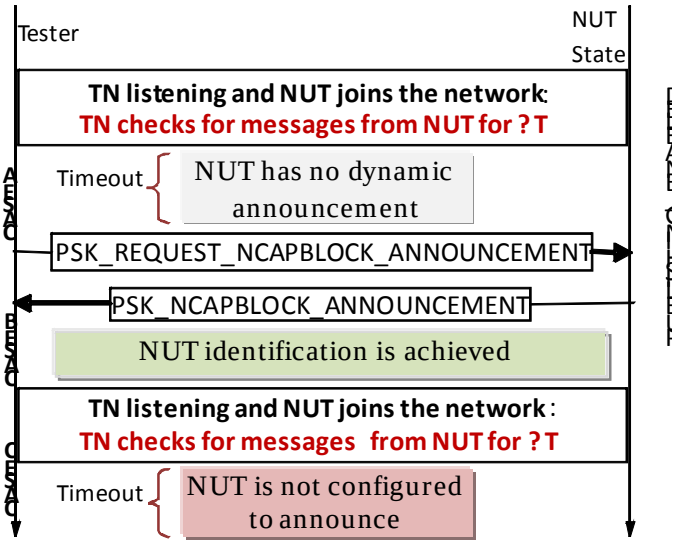
Node Discovery: PSK_NCAPBLOCK_ANNOUNCEMENT, PSK_REQUEST_NCAPBLOCK_ANNOUNCEMENT, PSK_FORCE_NCAPBLOCK_ANNOUNCEMENT, IgnoreRequestNCAPBlockAnnouncement, RespondToRequestNCAPBlockAnnouncement
Object Identification: GetNCAPManufacturerID, GetNCAPModelNumber, GetClassID, GetBlockManufacturerID, GetBlockModelNumber, GetBlockVersion
Network Configuration: GetNetworkVisibleServerObjectProperties, GetClientPortProperties, SetClientPortServerObjectBindings
Transducer Configuration: Get/SetSamplingFreq, Get/SetUpdateFreq, Get/SetUpdateMethod, Get/SetOperationMode, GetNumberChannels, Get/SetTEDS
Data Collection: Update/GetChannelData, PSK_PHYSICAL_PARAMETRIC_DATA, Start, Clear
System Shutdown: GoInactive

Functional testing is accomplished by the tester which assumes the role of a subscriber when NCAP under test (NUT) should publish or the role of a client when NUT should provide server functions such as a data transfer. Each operation is tested for its validity, corresponding state of the NCAP device, and configuration updates.

D.1.I. Publication Announcement Test I & II

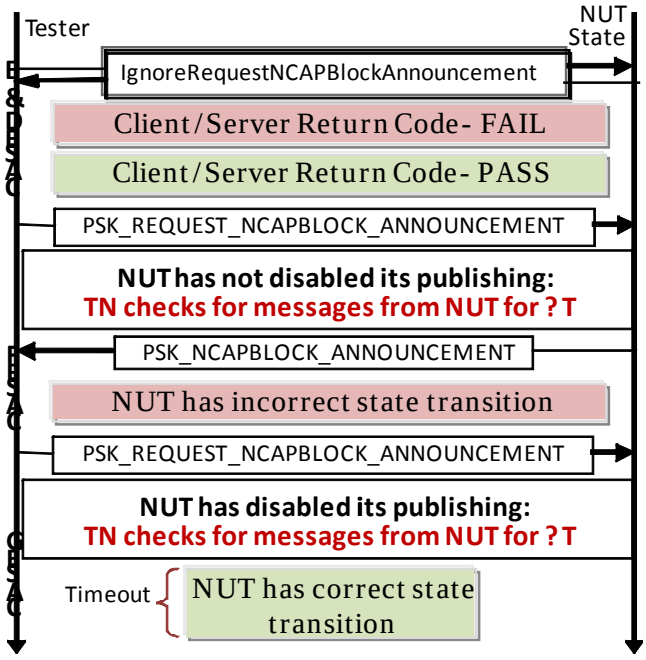
Dynamic versus static announcement configuration is tested first when the NCAP is connected to the network. Fig. 4 shows a time diagram of how tester node waits for a ΔT amount of time for an automatic announcement. Case A reports no dynamic announcement if none is received after timeout. A request is sent by the tester assuming that the NUT's state is Publishing_Enabled. If an announcement response is received within the expiration of ΔT, NUT identification is achieved. If not, Case C reports that the NUT is not configured to announce. After the NCAP is identified, a client-server

message is sent to disable the publishing state of the NUT. Afterwards, it is tested for verification – if the NCAP still publishes in response to a request for announcement, NUT fails the state transition test.



(a)

Figure 4. (a) Publications are expected for ΔT time – when expired, a request for announcement is sent. When successful, NUT identification is received.



(b)

(b) NUT states are tested using state enabler and disabler operations.

Second phase of this publication announcement test forces the state of the NUT to change back and forth between enabled and disabled states for publications through publisher/subscriber and client-server messages such as the PSK_FORCE_-NCAPBLOCK_ANNOUNCEMENT and IgnoreRequestNCAPBlockAnnouncement, respectively.

D.1.II. Client-Server Communication

Once NCAP on a network has been identified through announcements, its objects can be addressed through client/server communications to perform operations such as polling of transducers, setting sampling rates, etc. in addition to retrieving all properties of the network visible objects of an NCAP. The server responses contain specific return codes listed in Dot1 which are verified by the tester. In addition, server responses return the specific properties requested by the client – these properties are also verified by the tester. Hence, a PASS is issued by the tester only when both return codes and properties are received correctly, as shown in Fig. 5.

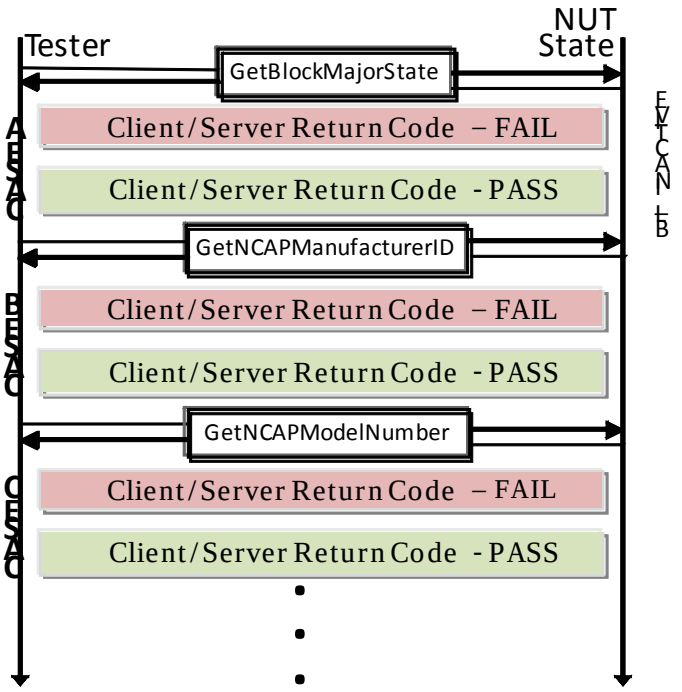
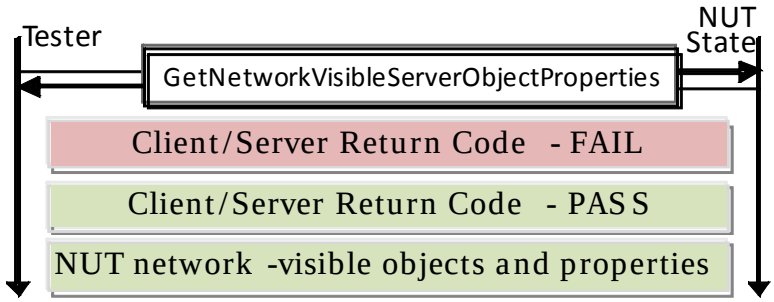


Figure 5. (a) States and other identifications are retrieved for NCAP and the block.



(b) All network visible object properties are retrieved over the network from the NCAP under test.

D.1.III. Get/Set Functionality and Parameter Configurations

This test verifies the active/inactive states and their operational issues. While an NCAP is inactive, Get/Set operations on any object should not be operational. Get/Set messages are sent to the NUT to verify that it responds with a "service unavailable" message. When active, these operations should return the requested data, that is, perform the operation. For example, one of the key features of IEEE 1451 is the definition of Transducer Electronic Data Sheets (TEDS). The TEDS is stored in a memory device attached to the transducer to provide transducer identification, calibration, correction

data, measurement range, and manufacturer-related information, etc. During inactive state, NUT should not send its TEDS to a GetTEDS request, and it should not update its TEDS based on a SetTEDS request. Figure 6 shows an interaction diagram of a section of the test process on inactive and active states for SetTEDS and GetTEDS operations.

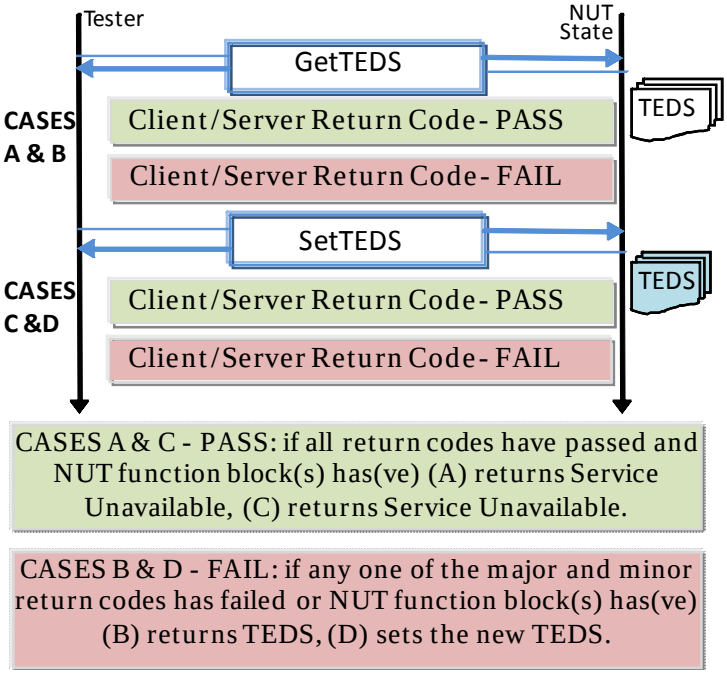


Figure 6. Before NUT is changed to active state (while in BL_INACTIVE state), GetTEDS and SetTEDS operations should return "Service Unavailable" in their return codes. When NUT is in BL_ACTIVE state the cases for FAIL and PASS would be reversed.

During active state, all operations based on Get/Set will be functional. The tests are designed so that each test first retrieves the parameter's current value, sets the parameter to a valid value, and then retrieves it again to verify that it has been updated. Also, some incorrect parameter settings are sent to the NUT to confirm its capability to respond with an "invalid parameter value" message to the tester. Similar testing procedures were designed for the following operations besides SetTEDS/GetTEDS:

i. GetSamplingFreq/SetSamplingFreq: sampling frequency of the sensor readings,

ii. GetUpdateFreq/SetUpdateFreq: the update frequency at which the NCAP would report the readings of a sensor – update frequency cannot be higher than the sampling frequency and similarly, sampling frequency cannot be lower than the update frequency,

iii. GetUpdateMethod/SetUpdateMethod: update method (UM) that the NUT uses to report data to other NCAPs – it can be broadcasted (publish), retrieved in polling (pulled by the client), or interrupted (pushed through by the server) fashion,

iv. GetOperationMode/SetOperationMode: operation mode (OM) is used to configure the nature of data that is transferred – e.g. raw would correspond to voltage values for a thermocouple in the place of the actual temperature readings.

D.1.IV. Data Exchange Test

According to IEEE 1451, NCAPs should be able to exchange sensor readings or control actuators among each other. Generic data exchange or update modes can be defined such as (i) periodic broadcasts of sensor readings (defined in Dot1); (ii) client request, server response, and then client acknowledge (implementation specific); or (iii) client request, server response, and no acknowledgment (implementation specific). NCAPs are tested for different update modes. Sensors report data to an NCAP at their sampling rates. Other NCAPs can retrieve this data at the update frequency. In the case of a periodic broadcast, the update frequency determines the period. During a client/server data exchange, client request overrides the update frequency; that is, the server responds whenever it receives a request even if the update period has expired or not passed yet. Data exchange modes are represented by UM (update method) prefix: for example,

UM_PUBLISH is multicasted at the update frequency whereas UM_SERVER_PULL is a server initiated data request/response mode. Messages to be tested are:

i. PSK_PHYSICAL_PARAMETRIC_DATA: Publication message of data transfer is multicasted with the key for subscribers (PSK stands for publish-subscribe key).

ii. Start/Clear: "Start" indicates the beginning of the data transfer from an NCAP with a transducer to the requesting NCAP. "Clear" stops the transfer.

iii. UpdateChannelData: Transducer communicates through a "channel" according to Dot1. Channel data is requested actively by the client (tester) from the NCAP in a one-time polling fashion. An acknowledgment is sent by the client once data is received successfully. Receipt of ACK by the NCAP (server) triggers more data to be sent.

iv. GetChannelData: channel data is being requested and acknowledged by the tester. Even after the receipt of the ACK, NCAP will expect the client to still send a request (GetChannelData) to continue sending data.

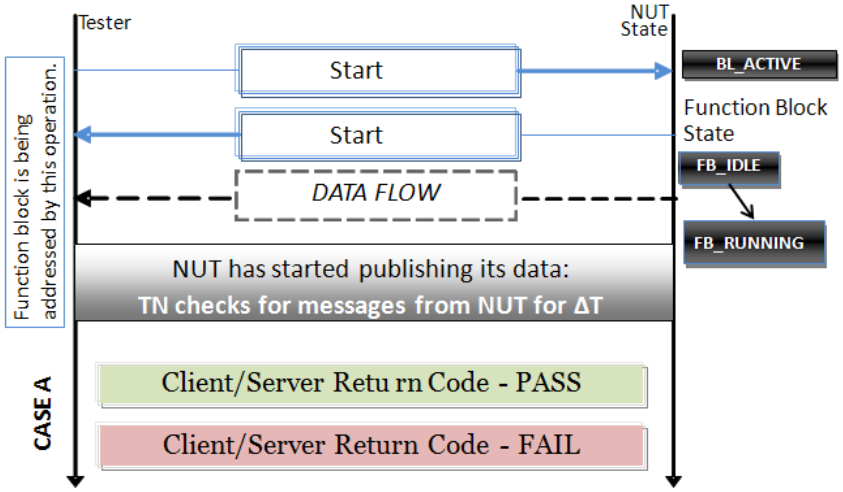
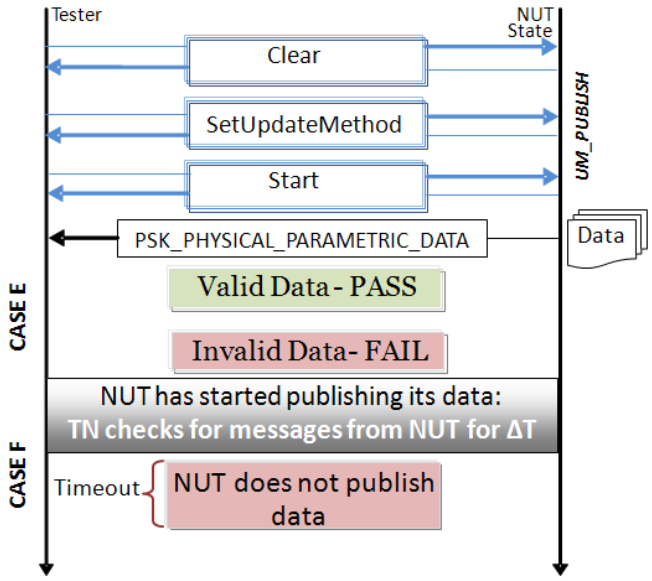


Figure 7. (a) Start operation will activate the publication of data in a multicast fashion when the update method is set to "Publish."



(b) When the update method is deliberately set to publish, the test is run again to verify that the data flow starts after the start operation.

D.2. Conformance Testing of Dot1

Conformance testing refers to the format of application packets and data encoding that an NCAP needs to follow to communicate from one function block to its corresponding function block in another NCAP. These packets are parsed by an NCAP to derive the information being conveyed from the other NCAP. In order to successfully communicate over the network, it is essential that they use the same packet format. However, Dot1 or any other part of the IEEE 1451 standards suite does not define physical layer data exchange format or the data packet fields in order to keep the standard flexible enough for interface design among networked devices of specific applications. Instead, it only specifies the minimal content of the fields of a packet. For example, an "operation id" specifies the arguments of a packet so that the application process can look for the data types and values. The operation id of "GetNetworkVisibleServerObject-Properties" is 4106. As long as the destination NCAP can parse and identify the field of the packet with this operation id, the network visible object properties will be fetched,

assembled into a packet, marshaled and then sent to the client. The test procedures implemented and presented in this paper uses the XDR (external data representation [19]) description and data encoding as an example. The data types in XDR are mapped onto the data types defined in Dot1 using [20].

There are primarily three main types of packet exchanges that have been defined in Dot1: publish or subscribe, client to server (c2s), and server to client (s2c). A c2s packet for an operation always has a return packet from the server in the form of a s2c packet. The test procedures are designed to identify (parse) and check the packet fields and formats. The packet formats have been defined with the following fields: **Byte** is the number of bytes for a particular field in the packet, e.g. 6-8: the field begins from 6th byte in the packet and ends at the 8th byte. Thus, it can be inferred that the length of the field is 3 bytes, which includes 6th, 7th and 8th bytes. **Interpretation** is the field name of a parameter in the packet, e.g. OperationID is the field name for the operation of the packet. **Details** section is on the structure and the way in which the bytes are designated to each part of the data string. Some of the absolute values will be specified and unknown or arbitrary ones will be denoted by N. Tables 2 and 3 display sample packets.

D.2.I. Publish/Subscribe Packet Format

Publish/subscribe operations are performed with multicast communication among the NCAPs and use UDP (User Datagram Protocol) as the transport protocol. The flow diagram of the process is shown in Fig. 8. As long as the listed attributes are included in the multicast message, Dot1 compliance will be achieved.

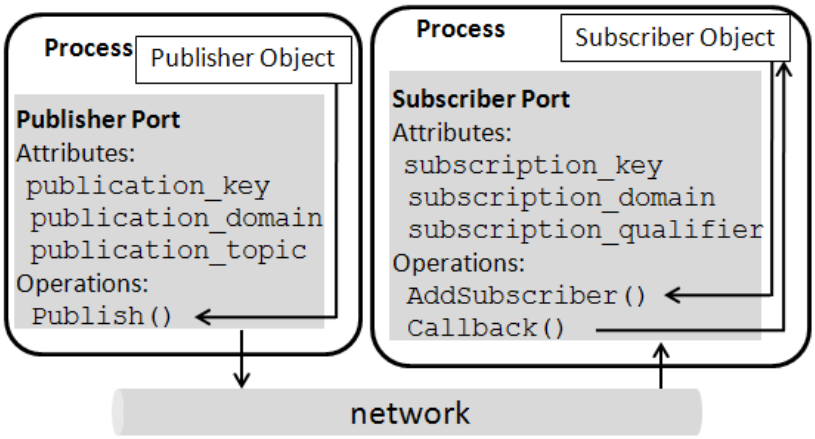


Figure 8. Publish/ Subscribe processes communicate using the attributes listed in the standard [Dot1 page 30 figure 7].

PSK_REQUEST_NCAPBLOCK_ANNOUNCEMENT: NCAP block announcement request is sent by the tester. The publication key is "04" [8: Table 135 page 242]. The response is an announcement message from the NCAP under test. Table 2 shows the fields of this packet. Magic number (implementation-dependent) identifies this packet as a publish/subscribe message. The version of data encoding style is the next byte. Total message and header lengths are variable and represented by 4 bytes. Publication keys are defined in Dot1 as unsigned integer type and 8 bits long [8: page 242 table 135].

PSK_NCAPBLOCK_ANNOUNCEMENT: NUT is expected to respond to a request for announcement with a PSK_NCAPBLOCK_ANNOUNCEMENT message. Publication key for this operation is "02". This publication delivers the network-specific information on the NUT such as its object dispatch address and object id that will include network address information depending on the particular networking protocol.

Table 2. Packet formats of PSK_REQUEST_NCAPBLOCK_ANNOUNCEMENT:

Byte	Interpretation	Details
0-1	Magic Number, Version	0xF7, 0x04
2-3	Total Message Length	0xNN NN

4-5	Header Length	0xNN NN
6	Publication Key	0x04 [PSK_REQUEST_ NCAPBLOCK_ANNOUNCEMENT]
7-14	Publication Domain	0xFF FF FF FF FF FF FF FF
15-16	Publication Topic	0x00 00 [Not applicable]
17-18	Publication Contents	0x00 00 [Not applicable]

and PSK_NCAPBLOCK_ANNOUNCEMENT:

Byte	Interpretation	Details
0-1	Magic Number, Version	0xF7, 0x04
2-3	Total Message Length	0xNN NN
4-5	Header Length	0xNN NN
6	Publication Key	0x02 [NCAP_BLOCK_ANNOUNCEMENT]
7-14	Publication Domain	0xFF FF FF FF FF FF FF FF
15-16	Publication Topic	0x00 00 [Not Applicable]
17-20 + N	Publication Contents	0x|00 02|: number of arguments – in this case, there are two arguments, object tag and object dispatch address. |*datatype*|: (1 byte) object tag |object tag|: (N bytes) 1st argument of this publication |*datatype*|: (1 byte) object dispatch address |object dispatch address |: (N bytes) 2nd

D.2.II. Client/Server Packet Format

Once the address of NUT is obtained using announcement messages, point-to-point connections are possible. For example, in the case of a TCP/IP network with Ethernet as the data link layer, the publication contents in the PSK_NCAPBLOCK_ANNOUNCEMENT can reveal the object dispatch address in the form of: **(i)** OctetArray[N] as ipAddress; **(ii)** UInteger16 as portNumber; **(iii)**

OctetArray[N] as the objectID, where the IP Address may be encoded as a four element OctetArray with each byte containing one octet. IP Address may also be encoded as a human-readable ASCII string without a NULL terminator. An example case is with IP Address of 192.168.1.100, the corresponding four element OctetArray on-the-wire data is 00 04 C0 A8 01 64, and the human readable on-the-wire-data becomes 00 0D 31 39 32 2e 31 36 38 2e 31 2e 31 30 30. Test procedures are implemented for a selected set of mandatory operations listed in Table 1. An example c2s/s2c message is presented below:

GetNetworkVisibleServerObjectProperties: Network visible server object properties are retrieved using the operation id of 4106. Each block object properties will be listed with identification. Format for the elements in the array of the server object properties will follow the sequence of object tag, owning block object tag, object dispatch address, object name, and block cookie. The number of blocks on the NUT and their object properties are retrieved using this operation. The packet format is included in Table 3 for c2s request and the s2c response packets.

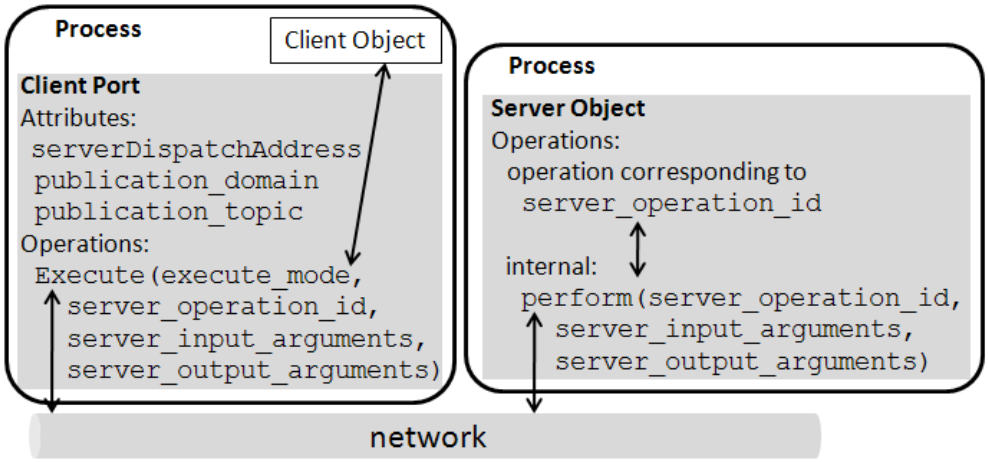


Figure 9. Client/Server communications using the attributes from Dot1 [8: p. 29 fig. 6].

Table 3. The c2s GetNetworkVisibleServerObjectProperties message:

Byte	Interpretation	Details
0-1	Magic Number, Version	0xED, 0x04
2-3	Total Message Length	0xNN NN
4-5	Header Length	0xNN NN
6-16	Server Object's ObjectID	0x|NN NN|: length |03|: UUID Algorithm ID |NN NN NN NN NN NN|: MAC address (6 bytes) |NN NN|: miUUID (minor Universal Unique Identifier)
17-18	OperationID	0x10 0A [4106: GetNetworkVisibleServer...]
19	Execute Mode	0x00
20-21	ClientPort Block Cookie	0x00 00 [NOT_SET]
22-23	Input Arguments	0x00 00
24-25	Expected Number of Output Arguments	0x00 02

and, the s2c GetNetworkVisibleServerObjectProperties message:

Byte	Interpretation	Detail
0-1	Magic Number, Version	0xD5, 0x04
2-3	Total Message Length	0xNN NN
4-5	Header Length	0xNN NN
6-16	Server Object's ObjectID	0x|NN NN|: length |03|: UUID Algorithm ID |NN NN NN NN NN NN|: MAC address (6 bytes) |NN NN|: miUUID (minor Universal Unique Identifier)
17-18	OperationID	0x10 0A [4106: GetNetworkVisibleServer...]
19	Execute Mode	0x00
20-21	ServerPort Block Cookie	0x00 00 [Not Set]
22-25	C/S Return Code	0xNN NN NN NN

26-29+N	Return Arguments	0x|00 02|: number of arguments – in this case this_block_object_tag and server_object_properties |*datatype*|: (1 byte) object tag (0x1A) | *this_block_object_tag* |: (N bytes) 1st arg. of operation |*datatype*|: (1byte) object properties array (0x19). |*server_object_properties*|: (N bytes) 2nd arg. of oper.

E. Implementation

The NCAP software has been implemented in java to have a reference version of an NCAP using the open source implementation as a reference [21]. The software implementation has a user interface to display which tests are being run. It initializes networking functions, follows the sequence of events in the tests to put together messages, marshals the messages, and then sends them to the network as outlined in Fig. 10. Listening thread demarshals and parses the messages for the tests.

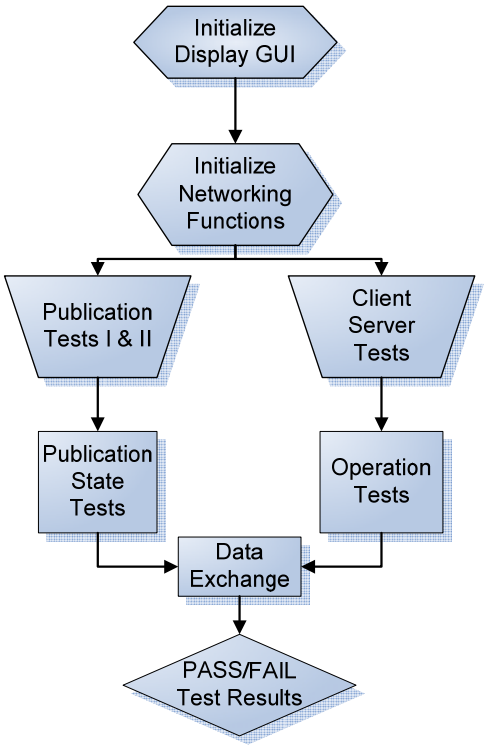
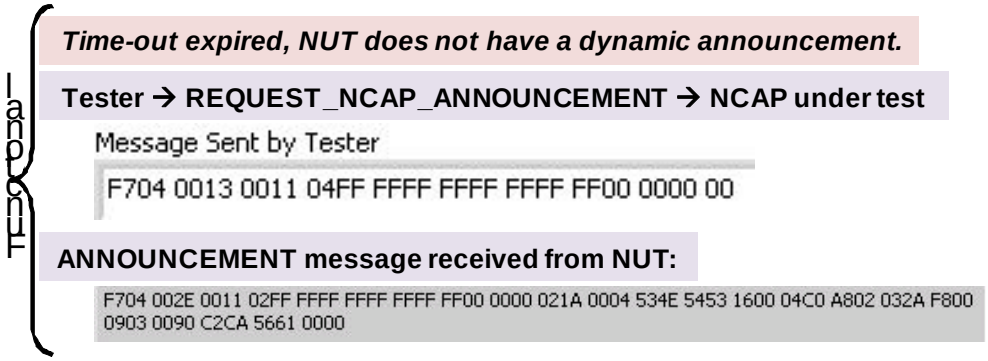


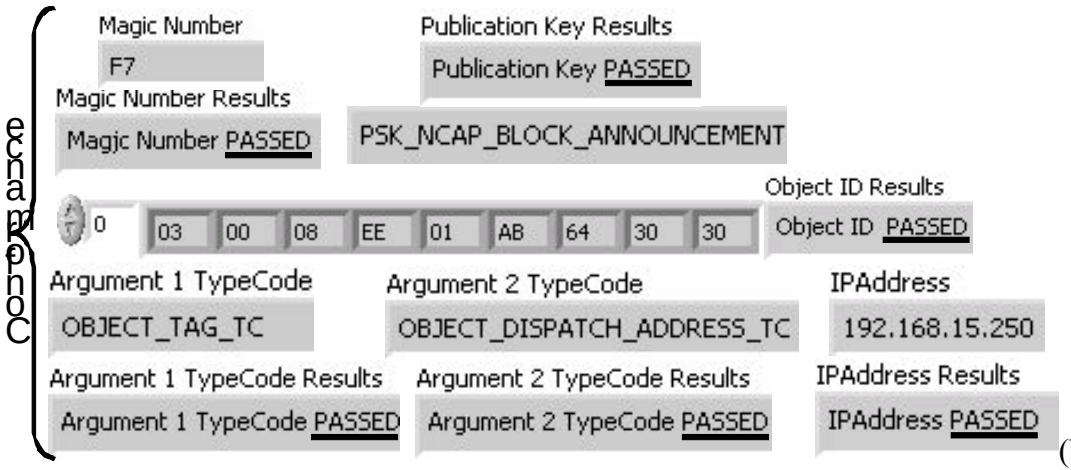
Figure 10. Flow of testing software program.

In addition, LabView implementation has been developed based on data types defined in Dot1. Typical test results from the LabView reports include conformance and functional compliance results. Tests have been conducted on NCAP units manufactured by Mobitrum Inc. [22]. These NCAP units also follow the XDR format and the same packet field definitions with the tester. Two test results are presented in this paper, one with multicast messages and another with client-server messages.

E.1. Publication Test Results: The tester waits for ΔT for an announcement from the NCAP that has just been connected to the network. After the timeout, NCAP under test (NUT) will be reported on whether it initiated its dynamic announcements. Then the tester sends a request for an announcement. This message is a multicast that addresses all NCAPs subscribed to the specific publication key of "PSK_REQUEST_ NCAPBLOCK_ANNOUNCEMENT." The testing setup has only one NCAP under test.



(a)



(b)

Figure 11. (a) Functional testing of configuration of the NCAP for dynamic vs. static announcements. (b) Conformance verification of fields in return packet.

The received message, PSK_ NCAPBLOCK_ANNOUNCEMENT, is parsed for conformance verification. The message will be parsed to verify all packet fields. Figure 11 shows some of the fields in the test results of a successful test that indicates the NUT does not have a dynamic announcement and its return packets for the announcement request passed the conformance test. The announcement message has two arguments: object tag and object dispatch address. In this implementation, object tag and object identification refer to one composite entity: a universally unique identifier assigned to each NCAP unit that is composed of IP address, port number, and the object id of the NCAP.

E.2. Data Exchange Test Results: The test verifies that the NUT has an active (FB_RUNNING) function block to report readings from a sensor. Figure 12 (a) shows the server to client message received from the NUT with the correct operation id, major return code, and the function block state. A GetChannelData message triggers the NUT to send the readings at the update rate. In this implementation, temperature sensor readings

have been reported. A simple conversion reports the average temperature to be 70 F based on the sensor readings.

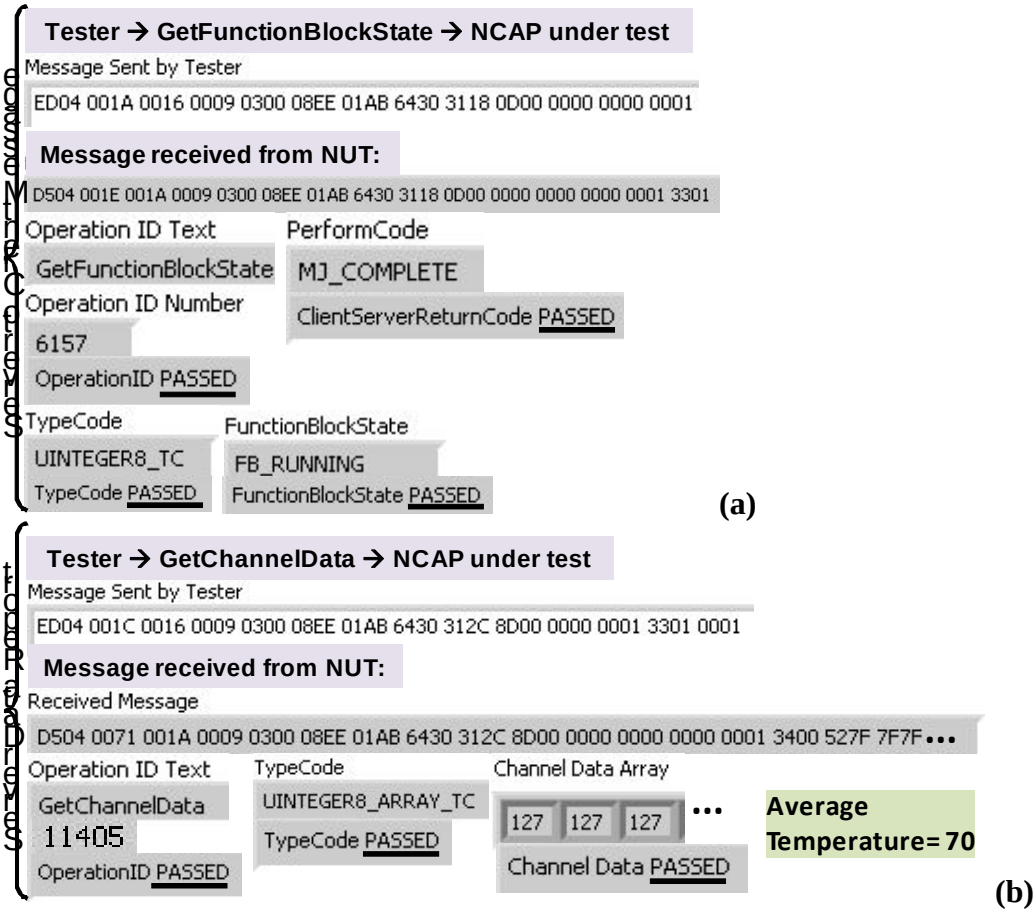


Figure 12. (a) Function block state is returned as FB_RUNNING which is suitable for sensor readings and reporting. **(b)** Channel data has been posted for temperature sensor.

F. Conclusions

IEEE 1451-based smart sensor technology is expected to be the key enabling technology to implement distributed control systems. In particular, IEEE 1451.1 will provide network access and communications among different smart sensors through a network environment and function as a gateway to sensor information. As many smart sensors technologies are being implemented by different research and industry groups, testing the interoperability of these implementations is necessary to ensure proper system

integration and interaction without regard to any specific technology and implementation. The advantage of the Dot1 standard is its clear definition of the architecture of a smart sensor, along with the software and network communications functions to keep the sensor in a known state.

To ensure that an IEEE 1451 implementation will interoperate within heterogeneous network environment, testing strategies are developed and adopted in this paper. The paper describes the Dot1 test cases and an integrated software tool at the application layer that executes these testing procedures when connected to smart sensors. The testing tool focuses on system perspective and exercises all the functionalities among NCAPs. These test cases are developed for Conformance testing and Functionality testing. The Conformance testing captures the general packet format dissection and standard compliance; while the Functionality testing tests the behavior of a smart sensor as it responses to different events. All these testing will help manufacturers to develop interoperable units and consequently enable better control and fault management with easier access to monitoring of physical phenomenon.

G. Acknowledgments

This project has been funded by the NASA - Stennis Space Center, CAN No. NNA06AB90A, under the Integrated System Health Management (ISHM) topic.

H. References

[1] E. Song and K. Lee, “An Implementation of the Proposed IEEE 1451.0 and 1451.5 Standards,” IEEE Sensors and Applications Symposium, 2006, Houston, TX.

[2] D. Gurkan, X. Yuan, D. Benhaddou, F. Figueroa, and J. Morris, “Sensor Networking Testbed with IEEE 1451 Compatibility and Network Performance Monitoring,” IEEE Sensor Applications Symposium 2007.

[3] L. H. Eccles, “The need for smart transducers: an aerospace test and evaluation perspective,” *IEEE Instrumentation & Measurement Magazine,* vol.11, no.2, pp.23-28, April 2008.

[4] E. Y. Song, Kang Lee, "Understanding IEEE 1451-Networked smart transducer interface standard - What is a smart transducer?," *Instrumentation & Measurement Magazine, IEEE* , vol.11, no.2, pp.11-17, April 2008.

[5] D. Wobschall, “Networked sensor monitoring using the universal IEEE 1451 Standard,” *Instrumentation & Measurement Magazine, IEEE* , vol.11, no.2, pp.18-22, April 2008.

[6] Kang Lee and E. Song, “A Wireless Environmental Monitoring System Based on the IEEE 1451.1 Standards,” Proceedings of the IEEE Instrumentation and Measurement Technology Conference, 2006. IMTC 2006. April 2006 Page(s):1931 – 1936

[7] Object-oriented application framework for IEEE 1451.1 standard [smart sensors] Kang Lee; Song, E.; Instrumentation and Measurement Technology Conference, 2004. IMTC 04. Proceedings of the 21st IEEE, Volume 2, 18-20 May 2004 Page(s):1182 - 1187 Vol.2

[8] "IEEE Standard for a Smart Transducer Interface for Sensors and Actuators-Network Capable Application Processor (NCAP) Information Model ," *IEEE Std 1451.1-1999* , vol., no., pp.-, 2000

[9] Anshul Singla, H. Ma, R. Franzl, H. Liu, D. Gurkan, D. Benhaddou, X. Yuan, J. Morris, M. Turowski, and F. Figueroa, “Design of a Test Suite for NCAP-to-NCAP Communication based on IEEE 1451,” IEEE Sensor Applications Symposium 2008.

[10] D. Gurkan, X. Yuan, D. Benhaddou, A. Singla, R. Franzl, H. Ma, H. Liu, F. Figueroa, and J. Morris, “Sensor Networking with IEEE 1451 Compatibility Testing,” Earth and Space Conference 2008, March 2-4, Long Beach, CA.

[11] Richard Franzl, Jonathan A. Morris, and D. Gurkan, “Implementation of IEEE 1451.1 Conformance/Functionality Testing using LabView,” IEEE Sensor Applications Symposium 2008.

[12] Helena Maria G. Ramos, J. M. Dias Pereira, Vítor Viegas, Octavian Postolache, and P. M. B. Silva Girão, “A Virtual Instrument to Test Smart Transducer Interface Modules (STIMs),” IEEE Transactions on Instrumentation and Measurement, vol. 53, no. 4, August 2004.

[13] L. Chara, O. Ruiz, and J. Samitier, “Complete IEEE-1451 Node, STIM and NCAP, Implemented for a CAN Network,”

[14] Vítor Viegas, J.M. Dias Pereira, and P. Silva Girão, "Using a Commercial Framework to Implement and Enhance the IEEE 1451.1 Standard," IMTC 2005 – Instrumentation and Measurement Technology Conference, Ottawa, Ontario, CANADA, 17-19 May 2005.

[15] Vítor Viegas, J. M. Dias Pereira, Senior Member, IEEE, and P. M. B. Silva Girão, Senior Member, IEEE, ".NET Framework and Web Services: A Profit Combination to Implement and Enhance the IEEE 1451.1 Standard," IEEE TRANSACTIONS ON INSTRUMENTATION AND MEASUREMENT, VOL. 56, NO. 6, DECEMBER 2007.

[16] R. L. Oostdyk, C. T. Mata, and J. M. Perotti, "A Kennedy Space Center implementation of IEEE 1451 networked smart sensors and lessons learned," *Aerospace Conference, 2006 IEEE* , vol., no., pp. 20 pp.-, 4-11 March 2006.

[17] A. Stepanenko, K. Lee, R. Kochan, V. Kochan, and A. Sachenko, "Development of a minimal IEEE 1451.1 model for microcontroller implementation," Proceedings of the 2006 IEEE Sensors Applications Symposium, pp. 88- 93, Houston, TX.

[18] George Percivall, OGC, "Sensor Access and Integration via Sensor Web Enablement and Demonstration of the OWS-3 Project," Sensors Expo 2006, Sensor Standards Harmonization Special Session.

[19] Sun Microsystems Inc., RFC1014, "XDR: External Data Representation standard" Network Working Group, June 1987.

[20] Online Source: "1451.1 On-the-wire format for IP," Agilent Technologies, Inc., Boeing Company, April 11, 2002, Rev. 1. last accessed July 14, 2008 on: http://sourceforge.net/project/showfiles.php?group_id=73157&package_id=109031.

[21] Online Source: "1451.1 Network Neutral Java Implementation," last accessed July 14, 2008 on: http://sourceforge.net/project/showfiles.php?group_id=73157.

[22] Ray Wang, "Wireless Intelligent Sensors Supporting Integrated Systems Health Management (ISHM) Architecture," IEEE Sensor Applications Symposium, 2008, Atlanta, GA.