Monitoring Network Aware Sensors Using BACnet

Paul Vaillancourt and Julia Johnson

Dept. of Math & Computer Science, Laurentian University, Sudbury, Ontario, Canada, P3E 2C6

Summary
We propose a protocol and software to use within a slow control system where slow in this context means that values are read from a variety of sensors at most once every second. We demonstrate how to use the Building Automation and Control network protocol (BACnet) to monitor network aware end-devices and set alarms on their sensor values.

Key words:
Sensor network, Network protocol, Monitoring network sensors, BACnet

1. Introduction

Ours is the matter of standardizing the technologies for implementing a highly specialized computer network. The devices sitting at the ends of the network are computers and heterogeneous sensors that read continuous data from the environment at a relatively slow rate (about once a second).

1.1. Sudbury Neutrino Observatory

The (SNO) is a research laboratory that contains a unique neutrino telescope, the size of a ten story building, two kilometers underground in INCO's Creighton Mine in Lively Ontario. It is operated by a one hundred member team of scientists from Canada, the United States, and the United Kingdom [12]. Neutrinos make up one quarter of all known fundamental particles in the Universe. By contributing to our understanding of neutrinos, the researchers at SNO are learning about the core of the sun, hoping to recognize characteristics that would unveil the secrets of our origins.

Acquiring information on the properties of neutrinos requires a large flow of information to be handled on a continuous basis. SNO monitors approximately four hundred data points, from temperature and humidity probes to oxygen meters and pressure transducers. These data points are gathered using what is called a *Slow Controls System* (SCS) that monitors, controls and sets alarms on various end-devices sitting on the network. A system is considered to be *slow* if it reads cycles in the one to ten second range, not at the millisecond or multiple minute range. The components used in a SCS can be classified as being either *proprietary* or *generic*. By *proprietary*, we mean that the system can only be used with other systems from the same manufacturer. When dealing with many different end-devices from many different manufacturers, having proprietary hardware or software is not beneficial from an interoperability point of view.

The problem with the current proprietary hardware and software combination at the Neutrino Observatory (SNO) is that it lacks extensibility. SNO will be expanding its facilities to extend their research area. The number of data points being monitored will multiply by a factor of three. It is, therefore, essential to embrace the new generation of relatively inexpensive sensors that communicate via a local area network using a variety of protocols, some of which will be examined in this research.

1.2. Focus of Research

The scope of this project is the monitoring on slow control signals. Signals consist of 1.) Digital inputs (on/off, open/closed, alarm/no Alarm) 2.) Raw or calibrated analog signals (calibrated temperatures, uncalibrated voltages that need to have the calibrations applied by the monitoring programs) 3.) Packets of digital data from complex devices such as a computer that receives a request for data and returns a variable length packet.

In this research, we evaluated some of the existing standard protocols for network communication among devices transmitting and receiving sensor data. We examined the Simple Network Management Protocol (SNMP), Programmable Logic Controllers (PLCs) and the Building Automation and Control network protocol (BACnet). BACnet [1], [2], [3], [4], [15] was selected as a protocol that can be used as a common language for communicating among devices. We demonstrate how to use BACnet to set up a network architecture to monitor network aware end-devices on their sensor values.

The criteria that were used for choosing an appropriate protocol and software are discussed in the next section.

1.3. Methodology

We provide guidelines divided into two sections: Protocol Guidelines and Software Guidelines. The Protocol Guidelines will be used to evaluate the alternative protocols in order to choose the one that is the most

Manuscript received January 25, 2006.
Manuscript revised January 30 , 2006.

suitable. Once we have proposed a protocol, we will use the Software Guidelines to evaluate the software needed to handle the protocol within our slow control system. Following are the requirements for finding an appropriate network communication standard and its software:

1.3.1. Protocol Guidelines

Upon reviewing a protocol, some of the key aspects to examine are: What are its strengths and limitations? How widely used is this protocol? What vendors support this protocol?

1.3.2. Software Guidelines

The functionality required is as follows:

- The software can read data from arbitrary network devices or, if only from proprietary devices, interfacing software can be written to connect an arbitrary device to this system.
- The system should read analog and digital signals as well as arbitrary data packets.
- Software analysis can be done on the signals once the data is read (e.g. calibration constants may be applied to the data). Mathematical operations can be performed on the data such as calculating the square or the log of a value).
- The system should log data and make it accessible by client programs.
- Alarms generated by the system should be appropriate for the device being monitored. For example, if we are monitoring the temperature in a computer room, it would be crucial to have a loud audio alarm go off when the temperature rises to a certain threshold. Simply sending out an email would not suffice since we need to deal with the problem immediately (increasing the air conditioning so the hardware does not catch on fire).
- The software/hardware combination should scale up with respect to how many signals it can handle. That is, it should not get unduly slower as the number of sensors increase.
- It should be possible to set up several standalone systems and have them interact with each other. For example, a subsystem that monitors air quality should be able to pass on alarms and selected information to a supervisory system.
- The status of the sensors can be viewed from an arbitrary number of locations. There are two ways this could be done: Each viewing location connects to all the desired sensors. Each viewing location is a client that connects to a server which in turn connects to the sensors. The latter is more desirable since it ensures that there is no extra burden on the sensors.
- The system should be platform independent.

This list has served as a guideline for proposing an appropriate solution. We concentrate on data acquisition rather than data presentation. High level alarming and fancy graphical user interfaces are not of interest to SNO.

1.4. Evaluating the Alternatives

The first protocol we evaluated was the Simple Network Management Protocol (SNMP) [5]. The functionality of SNMP is also explained in [14] where we have written an online application demonstrating its simplicity. We saw a problem of lack of compatibility among SNMP's various versions and we observed that the intended market was for an industry other than our own. Although the general idea behind SNMP was for it to be as simple as possible, the enhancements that were added have eventually made it complicated. For these reasons, we have dismissed SNMP as a possible network monitoring standard for SNO.

In order to branch out into a more relevant field, we have evaluated Programmable Logic Controllers (PLCs) [8] as our second protocol, since they are considered the industry standard for industrial instrumentation. We have discussed many of their advantages, such as reliability, versatility, and modularity [14]. However, despite these strengths, PLCs are very proprietary and expensive. Since cost is an important aspect when implementing a new standard within a network, we have disregarded PLCs as an economical option for network monitoring within the facilities at SNO.

The third and final protocol that we have evaluated was the Building Automation and Control Network protocol (BACnet). A detailed explanation is given in [14] explaining the protocol's specification and many of its advantages. We have seen that BACnet is interoperable, cost-effective, and flexible. We have shown that it has real-world application knowledge as its foundation, and contains many rich services not found in many of its competitors. We have also illustrated its scalability by giving a few real-world examples of large scale BACnet implementations using thousands of devices. In addition, we have explored two possible limitations of BACnet, but have concluded that they were not enough of a concern for us to disregard it as an option. Because of the high number of strengths found within this protocol, BACnet was chosen as our network monitoring standard of choice.

We wanted to illustrate how to use the chosen protocol in a practical environment, so we have explored a few possibilities for setting up a BACnet network architecture.

We have demonstrated [14] these various hardware and software combinations which we have simplified into an affordable schema feasible for SNO's purposes.

The above summary will be expanded in the next section by describing the features of BACnet in more detail and by evaluating this protocol on the guidelines given in Section 1.3.

2. BACnet

2.1. Introduction

In the mid 1980s, there was a high demand for a cost-effective system that could centralize the monitoring, operation and control of various devices in buildings. Rigorous effort by AHSRAE, the American Society of Heating, Refrigerating and Air-Conditioning Engineers, resulted in the Building Automation and Control networks protocol, also known as BACnet. The protocol became part of the American National Standards Institute (ANSI) in December of 1995 and part of the International Organization for Standards (ISO) in January 2003. In the remainder of this paper, we will show that the BACnet protocol is an appropriate choice for sensor networks.

2.2. BACnet Specification as Defined by ASHRAE

The BACnet standard is divided into three major parts. The first part describes a method for representing any kind of building automations equipment in a standard way. The second part defines messages that can be sent across a computer network to monitor and control such equipment. A set of acceptable Local Area Networks that can be used to convey BACnet communications is described in part three.

2.2.1. Representing Devices Using Objects and Properties

Representing the functions of any device in a standard way is done by assigning a series of predefined objects to the device. These objects define such things as analog and binary inputs and outputs, schedules, control loops, and alarms. Each object has a set of *properties* that further characterizes the object. The object is a collection of related information accessible via different properties. For example, an analog input could be represented by a BACnet object called "Analog Input Object" which has a set of standard properties such as *Present_Value*, *Description* and *Device_Type.*

The objects that are present in a given BACnet device depend on its function and capabilities [13]. For example, a device that controls a VAV box (i.e. Variable Air Volume – a device that provides constant or variable air depending on the temperature demands of the space) will probably have several Analog Input and Analog Output objects, but a workstation that does not have sensor inputs or control outputs will not.

123 different properties of objects are defined in the BACnet standard each of which must contain at least the three properties Object_Identifier, Object_Name and Object_Type. Each property has a specific behavior defined by the BACnet specification. Once devices have a common appearance on the network, we can define messages for communicating among them.

2.2.2. BACnet Services – Monitoring and Controlling Equipment

A client is any device that requests a service (e.g. a piece of equipment or a computer) while the server is any device that performs a service. When an operator workstation is set up, the software can display a list of sensor inputs. The operator can then issue service requests to the objects of those devices and get all of the sensor's current values. The device's application program responds to the request and sends the data that have been requested.

Currently, the BACnet specification defines forty two services divided into five categories called *classes*. For example, one class contains messages for accessing and manipulating the properties of the BACnet objects. A common service in this class is "ReadProperty" which makes a request to the device's application program to return the value of a particular property in a particular object. Other classes of services deal with alarms and events, file uploading and downloading, managing the operation of remote devices, and virtual terminal functions.

2.2.3. BACnet Network Technologies

The BACnet architecture is made up of several layers. They consist of an application layer, a network layer, a data link layer, and a physical layer. BACnet controllers from different vendors can share a common LAN that was pre-selected by a system designer. The sensors or actuators sitting at the ends of the network can have the BACnet logic build into their hardware, in which case they are called *native speaking BACnet devices*, or there may be an intermediate BACnet controller that carries the logic. Either way, the BACnet standard is implemented before the devices reach the network so that they may communicate with other devices.

2.3. Evaluation According to the Protocol Guidelines

The BACnet protocol was built specifically to read values at a low resolution (such as once every second or slower) making it capable of reading in slow control signals. For example, to monitor a fire alarm system which outputs a

binary signal indicating whether or not the alarm went off (e.g. alarm/no_alarm), the BACnet client would read its Present_Value property in the Binary Input Object using the ReadProperty service request. To read in a temperature sensor's current analog value, we would use the Present_Value property in the Analog Input Object using the ReadProperty service request.

BACnet was designed to be interoperable. Being able to mix and match products from a variety of vendors optimizes the cost of our network solutions.

The BACnet object and service model were designed to be easily extended. The model was created in such a way that if a vendor has an idea for new functionality, new properties can be added to existing objects or new objects can be created entirely.

3. SCADA Engine Client Development

The *BACnet Rapid Development Kit* is software developed by *SCADA Engine* (SCADA stands for Supervisory Control and Data Acquisition) which allows developers to create their own BACnet applications.

3.1. Device Simulator Component

This component allows us to simulate a BACnet device without actually having one physically on the network. The RDK Device Simulator is very simple. As shown in Fig. 1, it contains four analog and four digital values that are read by the BACnet server as if it were a real device. These values can be read from or written to by a client application connecting to the server (since the server already knows the current values, it is the server who returns the information to the client and not the device simulator itself).

The Device Simulator also allows one to change its analog or binary values manually so that the functionality of a device's value varying according to the current environment may be simulated. For example, changing this value manually would be like a temperature sensor reading a temperature change in the current room. We can change one of the device's values by clicking on the ".." button next to the internal point one wishes to manually change (see Fig. 1). The popup that allows the setting of this value is shown in Fig. 2. The Trace Messages window within the Device Simulator GUI of Fig. 1 displays all of the messages that are sent or received to or from the server. This can be quite useful when trying to debug a client application using the Device Simulator because all of the communication to the server is outputted to that window. Once the Device Simulator values are stored in the BACnet server, they can be read by any device sitting on the network, real or simulated.

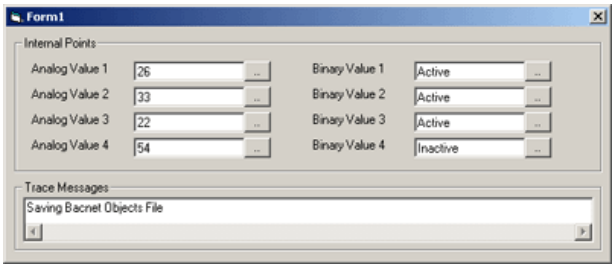


Fig. 1 The BACnet Device Simulator component

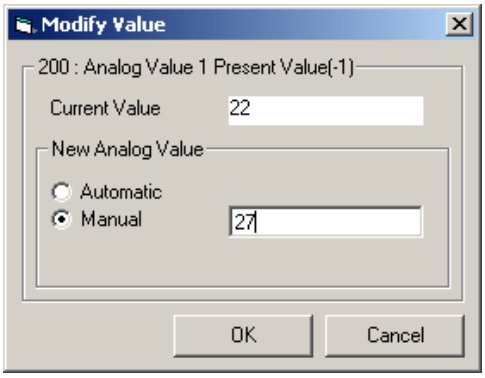


Fig. 2 Window within the Device Simulator GUI which allows manual modification of a value

3.2. Client Applications

In this section we will show how to read analog and binary values to a device. This application will be written in Visual Basic using the SCADA Engine BACnet Rapid Development Kit software. Since we do not have actual physical devices to read or write to, the Device Simulator component exemplifies an architecture with an actual BACnet device sitting on the network.

3.2.1. Reading Analog and Binary Values

In order to read an analog or binary value from a device, we need to use the ReadProperty service request as defined in the BACnet specification. The SCADA Engine software allows us to use all of BACnet's services or data types using the BACnetX DLL component within a Visual Basic application. The first thing that we need to do is include this component as a reference within our Visual Basic application. This can be done within Microsoft Visual Basic 6 by selecting 'Project' < 'References' and ticking both the 'SCADA Engine BACnetX components' as well as the 'BACnetxml 1.0 Type Library' option (both of these will appear in the list when the SCADA Engine RDK is installed). Once these are checked, the BACnetX component will be referenced within our application and will take care of our client connecting to the server. Everything is done automatically behind the scenes; all we

have to do is write a few 'magic' lines that tell our application to use the BACnetX component. We illustrate this in Fig. 3.

```
1  Option Explicit
2  Private WithEvents objApplicationLayer As BACNETXLib.ApplicationLayer
3
4  Private Sub Form_Load()
5      Set objApplicationLayer = New BACNETXLib.ApplicationLayer
6      objApplicationLayer.Initialize ""
7
8      'Functions are called here
9  End Sub
```

Fig. 3 Code which allows us to use the BACnetX component in our Visual Basic program

Note that Form_Load() in a Visual Basic application is comparable to the main method in Java or C++. The sequence of instructions and function calls that we place within this function are what get executed when we run the program.

Let us now illustrate how to use the ReadProperty service request to retrieve a device's analog value. The function which does this is depicted in Fig. 4.

```
1  Private Sub ReadAnalogValue()
2      'This example demonstrates the use of the readProperty service by reading
3      'the present value property from Analog Value number 1 inside device 200.
4      'If the service is successful, then the value of the property will be
5      'stored in the ack object.
6
7      Dim Service As New ReadProperty
8
9      Service.async = False
10     Service.DeviceID = 200
11     Service.Request.objectIdentifier.Instance = 1
12     Service.Request.objectIdentifier.ObjectType = BACnetObjectTypeAnalogValue
13     Service.Request.propertyIdentifier = Property_presentValue
14     Service.Execute
15
16     If Service.Error.choice = BACnetErrorChoiceEmpty Then
17         'No Error - Display the results
18         MyDebug "Value = " & Service.Ack.propertyValue.Real
19     Else
20         'An Error has occured
21         MyDebug "Error Class = " & Service.Error.Error.Class
                   & " Error Class = " & Service.Error.Error.Code
22     End If
23 End Sub
```

Fig. 4 Function that creates a ReadProperty service request and reads the device's analog value

The first thing that we do is create a ReadProperty object of type Service (these objects are available within the BACnetX component which we have now referenced within MS Visual Basic 6). On line 9, we set the asynchronization variable to *False* which means that the execute command will not return until a response, or timeout (set to 18 seconds by default) occurs. If we would set async to *True*, the execute command would return immediately whereas we could make a call to the ReadyState Property to find out when the service has completed. Now we need to tell the service which device we wish to poll, and we do this by assigning that particular device's ID to DeviceID on line 10. Every device on the BACnet network has its own ID, including the BACnet server component (it is treated as a special kind of device). The Device Simulator component has an ID value of 200. In order to read a value from any another device on the network, we would have to assign its ID to the DeviceID variable on line 10.

Next, we need to give the service some information about the request. On line 11, we initialize the instance of objectIdentifier to 1, meaning that we want to poll the first analog value of the device. The Device Simulator has four analog values, so we could have set this variable to a number between 1 and 4, depending on which instance of the analog value we wish to poll. Each BACnet device on the network can have either number of analog or binary values, depending on what its behavior is. For example, we can have a device which would have two analog values, one to measure the oxygen, and the other to measure the pressure, as well as one binary value to turn the device ON or OFF. Next within the objectIdentifier object we specify the type. On line 12, we set ObjectType to BACnetObjectTypeAnalogValue indicating that we are interested in polling the first instance of the analog value within the device. We then set the propertyIdentifier variable to Property_presentValue since in order to get the analog value, we need to request it from the Present_Value property within this object. Finally, on line 14 we execute the service by issuing Service.Execute. When this method is executed, BACnetX will first try and locate the Device by sending a whoIs request (whoIs is a standard BACnet service used to synchronize the Time Clocks across the BACnet network). If an Iam response is returned by the required BACnet device (Iam is another standard BACnet service used to respond to the whoIs service), then BACnetX will issue a ReadProperty service to the required device. If the service successfully returns, then the Ack object will contain the response from the device. If an error or timeout occurs, then the Error object will contain the error codes associated with the service. The next If Else statement makes sure that the device has not returned any BACnet errors before displaying the value sitting in Service.Ack.propertyValue.Real, which we output in the debug window of MS Visual 6. In the case where an error has been returned, the Else block displays the class and code of the error in the debug window.

Let us now illustrate how to read a binary value from the device. The code that does this, shown in Fig. 5, is very similar to the code of Fig. 4. There are only a few differences. The first obvious difference is on line 12, when we assign the ObjectType variable within objectIdentifier to BACnetObjectTypeBinaryValue. After executing the service, the server will return a either a 1 or a 0 indicating whether or not that instance (in our case, the first instance) of the binary object is ON or OFF. Line 18 checks if Service.Ack.propertyValue.Enumerated is equal to BACnetBinaryPVActive (representing 1) indicating that our requested binary value is ON. In this case, we simply output "Value = On" in the debug window of MS Visual Basic 6, otherwise the server has returned 0 and stored it in

the Enumerated variable within propertyValue of the ack object, and so we display “Value = Off” in the debug window.

```
Private Sub ReadBinaryValue()
    'This example demonstrates the use of the readProperty service
    'by reading the present value property from Binary Value number 1
    'inside device 200. If the service is successful, then the value
    'of the property will be stored in the ack object.

    Dim Service As New ReadProperty

    Service.async = False
    Service.DeviceID = 200
    Service.Request.objectIdentifier.Instance = 1
    Service.Request.objectIdentifier.ObjectType = BACnetObjectTypeBinaryValue
    Service.Request.propertyIdentifier = Property_presentValue
    Service.Execute

    If Service.Error.choice = BACnetErrorChoiceEmpty Then
        'No Error - Display the results
        If Service.Ack.propertyValue.Enumerated = BACnetBinaryPVActive Then
            MyDebug "Value = On"
        Else
            MyDebug "Value = Off"
        End If
    Else
        'An Error has occured
        MyDebug "Error Class = " & Service.Error.Error.Class
                & " Error Class = " & Service.Error.Error.Code
    End If
End Sub
```

Fig. 5 Function that creates a ReadProperty service request and reads the device’s binary value

In this section we have shown the basic necessities for polling a device’s analog or binary value using Visual Basic. Once we have obtained our value, we have simply displayed it in a debug window but we could have easily, for example, displayed it in a webpage or applied some mathematical function to it. This idea is very powerful because it allows a client application to carry out any task that is needed for monitoring various end devices. The SCADA Engine RDK software gives us this flexibility.

3.3. Evaluation according to the Software Guidelines

In this section, we will evaluate the SCADA Engine Rapid Development Kit software according to the software guidelines presented in Section 1.3. The software guidelines include ten aspects which we will evaluate individually.

3.3.1. Reading from Arbitrary Network Devices

Can the software read data from arbitrary network devices or can it only read from proprietary devices? If it is the latter, what interfacing software must be written to connect an arbitrary device to this system?

In order for any device to be understood by the SCADA Engine software, there has to exist some form of BACnet logic between the software level, and the device level. Since most monitoring devices that SNO will be dealing with will not be native BACnet devices (i.e. have the BACnet logic built into its hardware), there must be an intermediate BACnet controller. This will allow the SCADA Engine software to understand any device, either arbitrary or proprietary since the BACnetX ActiveX component can read from any Object on any device on the network.

3.3.2. Reading Arbitrary Data Packets

Does the system read only analog/digital signals or can it read arbitrary data packets?

The BACnet specification allows manufacturers to implement their own proprietary properties, objects, or services within BACnet if they wish to include functionality that is not part of the standard specification. Because of this, a more complex device has the ability to return an arbitrary length string containing virtually any kind of information. Since the BACnetX component within the SCADA Engine RDK software can read from any object on any device on the network, it can also read proprietary objects. All objects and properties within BACnet are identified by a number, where proprietary objects will have a number above 500. In order to read these proprietary objects within a Visual Basic client application, one needs to know the number of the object and the property that they wish to reference. Once a read property service request has been sent, the object may return an arbitrary length packet, depending on how the manufacturer decided to implement the object. Once the data is returned, it is up to the client application to handle that data properly.

3.3.3. Software Analysis on the Signal’s Data

Can software analysis be done on the signals once the data is read into the system? i.e. can calibration constants be applied to the data? Can mathematical operations be performed on the data (such as calculating the square or the log of a number)?

Since we have the flexibility to accomplish whatever is required within our own client applications, it is certainly possible to apply any kind of software analysis on the signals we are monitoring, such as applying calibration constants or mathematical functions. For example, once we have polled a particular device’s analog value, we can easily apply any Visual Basic math function on the returned value before we display it for the user. The idea behind the SCADA Engine RDK software in allowing us to create our own client applications was to give us the utmost flexibility to manipulate our data how we see fit.

3.3.4. Logging Data

How does the system log data? How is the logged data accessible by client programs? i.e. if the data is logged to a database, how does one access the database?

Once again, since we have the freedom to do as we wish within our client applications, we can easily output our

data to a file or a database. When settings alarms on the values, the BACnet Server component stores its alarm notifications into an MS access database, so we do not have to create one ourselves. There are a variety of ways that one could store this information within Visual Basic. For example, we can just as easily send an alarm notification to any number of email accounts as a way of keeping a record of which alarms have been issued. Our data can be logged an endless number of different ways as long as it is feasible by writing code.

3.3.5. Generating Alarms

How does the system generate alarms? Are there audio alarms (special hardware devices or does it rely on a computer's audio system)? How are the alarm values set?

The SCADA Engine RDK software has a built-in alarm mechanism called BACnAla.exe. Unfortunately, the most this program can do to notify someone that an alarm has been issued is to pop up on the screen of the machine running the client application. Because this might not be enough of a notification, we can also trigger a sound to go off on the client machine using DirectX or winsock control, since we have this flexibility within our Visual Basic applications. If these methods are not enough to alarm someone in the area, one other possibility might be to purchase a device similar to a fire alarm system, attach it to our BACnet network, and invoke a WriteBinaryValue() function to turn on the audio alarm when a particular device's values are out of range. This example demonstrates how powerful using the SCADA Engine software can be. Concerning how the alarm values are set, these can be hardcoded within our application, or can even be dynamically set. For example, we could code our client application to read its alarm values from a file sitting on a webserver. This way, one could modify the alarm values within this file from anywhere in the world at any time and each copy of the client application would have the same updated alarm values.

3.3.6. Scalability

What is the scalability of the software/hardware combination? i.e. how many signals can it handle? Does it get slower as the number of sensors increase?

Scalability is an aspect that is difficult to measure because there are many factors which can influence it within a network, such as the combination of hardware and software. The SCADA Engine BACnetX component can be used in a couple of different modes of operation. In one of the modes, the BACnetX component is self contained in a client application. This allows only one client application per machine to run at a time, but as an advantage, the client application does not need its own copy of the server component. Each client machine then connects to a server machine that processes each request. In this mode, there may be more of a network lag with a large number of clients because each request must be handled by the same server component. Another mode of operation consists of BACnetX connecting to the server via DCOM. This mode of operation allows multiple client applications to run on the same computer, but the only drawback is that a copy of the server must reside on the client machine. A plus side is that since every client machine connects to its own copy of the server, the work load is now distributed among multiple servers. Because of this, running a large number of clients in this mode of operation would probably cause less network lag as in the previous mode. These various modes of operation might have been implemented by SCADA Engine to accommodate small vs larger BACnet networks.

3.3.7. Modularity

What is the modularity of the software? i.e. can several standalone systems be set up? Can standalone systems interact with each other? For example, could a subsystem that monitors air quality pass on alarms and selected information to a "supervisory system"?

Because BACnet was designed to work well with interoperability, various client applications can easily interact with any device on the BACnet network. The degree of modularity within our network is not dependent on the SCADA Engine RDK software but rather on the client applications that we write. For example, we have the flexibility to write a client program that monitors the air quality in a laboratory, and send an alarm notification to another "supervisory system" application running on another computer. When setting up alarms, we can define a list of recipients to receive notifications of alarms. Therefore, the degree of modularity is dependent on the functionality of our client applications.

3.3.8. Viewing Location

Can the status of the sensors be viewed from an arbitrary number of locations? There are two ways this could be done: (i) Each viewing location connects to all the desired sensors; or (ii) Each viewing location is a client that connects to a server which in turn connects to the sensors. The latter is more desirable since it ensures that there is no extra burden on the sensors.

In a BACnet architecture using the SCADA Engine software, each client, or "viewing location" does not connect to all of the desired sensors (or devices) directly, but rather connects to the server component which connects to the desired sensors. The client applications can run on any computer with access to the BACnet network,

and therefore can be viewed from any of these locations.

3.3.9. Supported Operating Systems

Is the system platform independent? i.e. what operating systems does the software run under? SNO uses Unix/Linux, Windows, Mac OS9, Mac OSX and even in one case, DOS.

The SCADA Engine RDK software was initially created for a Windows environment, but has recently released a Linux version of its server. This means that any Linux box can be set up to be the BACnet server. Client applications can also be created under a Linux environment with the help of Visual Basic for Linux tools. Unfortunately, as of this time, there are no Macintosh supported components, or even a DOS based version of the RDK. However, the majority of the operating systems used at SNO are under the Windows or Linux environment, so the lack of Macintosh or DOS based versions of the software should not be too much of a concern. Since most other BACnet operator workstation software will only run under Windows, having a Linux based version of a server is definitely favorable. Additionally, the BACnet Server contains a built-in XML gateway which allows an application to communicate to other BACnet devices on the network by issuing simple text messages. This communication uses TCP/IP and the body of the text messages is formatted in XML, making the gateway platform independent.
Based on this analysis, it is obvious that SCADA Engine's Rapid Development Kit package meets all of SNO's needs, as defined by their guidelines. We recommend this toolkit as an effective and flexible solution for implementing BACnet.

4. Related Work

Commonly used protocols of Fieldbus Technology (FT) [7] are being advocated as a standard for real-time distributed control in automation networks. A device description language is provided by a subsystem of FT called HART. Likewise, for distributed control of slow processors the devices must be described using a common language in order to communicate. The objects and properties of BACnet are being proposed here as the basis for such a language.

Foundation Fieldbus (FF) is being proposed as a standard for implementing the real-time aspect of automation problems. We could do the same for BACnet.
To advocate your technology as a standard for a specialized class of problems, the concern with proprietary protocols which lack interoperability must be resounded as it is here and also in [7]. The relationship between the proposed standard and the OSI reference model must be outlined as it is here and also in [7].

Let us assume that to achieve compatibility, interoperability, and interchangeability, it is sufficient for peer processes to use the same technology as long as they interface properly with the n-1 and n+1levels. The scenario then becomes one in which technologies from different vendors can be used as long as they are used at different levels in the reference model. Therefore, a network technology can become the standard for a specialized class of applications for some but not all of the levels in its architectural model. The proponents of Fieldbus concede that European machine control suppliers lead the way for web-based control in automation applications.

To become the standard it is important to show upward compatibility. Fieldbus technology supports single, open and interoperable networks where single is a special case of open and open is a special case of interoperable. Other contemporary protocols such as PROFInet [11] vying to become the standard must be seen as special cases.

For distributed control in real-time network applications and complex applications in general, the problem is best conceptualized in terms of components. Component based distributed control [6] abides by the principle of assigning one control algorithm to one component. The function of a component is local computation.

Component based distributed control also abides by the principle of component reuse. This is similar to the use of objects and properties supplied by BACnet for representing devices.

In this paper, we have demonstrated that BACnet is a promising technology for slow control sensor networks. It remains to be seen whether PROFInet and Fieldbus which like BACnet are intended for automation, show similar promise.

5. Conclusion

The objective of this research was to investigate computer network protocols in order to propose the one that is the most suitable for the Sudbury Neutrino Observatory.
The Building Automation and Control Network protocol (BACnet) was found to be most suitable for SNO. We have given a brief explanation on the protocol's specification and shown many of its advantages. BACnet is a protocol that is interoperable, cost-effective, and flexible. It has real-world application knowledge as its foundation, and contains many rich services not found in many of its competitors. In addition, BACnet's scalability is shown throughout the many real-world examples of large scale BACnet implementations using thousands of devices.
We have used the SCADA Engine Rapid Development Kit

software package to create a client application using a device simulator. Our practical applications have shown how one could use BACnet to monitor various devices on the network. This is a contribution because there is little documentation in the literature showing how to get started using BACnet.

This dynamic area of research is one that is continuously changing due to the current exponential technology improvements. Indeed, the realm of network communication protocols available is broad enough to expand this research to include other communication standards such as Modbus, N2, LonTalk, DeviceNet, CANopen, or CAB. Our methodology for the evaluation of protocols provides insight for future research into sensor networks.

Acknowledgments

We would like to acknowledge the contribution of Clarence Virtue (SNO, Laurentian University) and Fraser Duncan (SNO, Queen's University) in preparing the guidelines for selecting a protocol. This work was supported in part by the Natural Sciences and Engineering Research Council of Canada (NSERCC).

References

[1] BACnet Europe Journal Vol. 2, 2/05. Successful protocol analysis in BACnet/IP networks.

[2] BACnet STANDARD: ANSI/ASHRAE Standard 135-2001, ASHRAE, 2001. ISSN 1041-2336

[3] Butler, J., Cimetrics, "BACnet: An Object-Oriented Network Protocol for Distributed Control and Monitoring" (2002), http://cimetrics.com/main/About/articles.php?id=3

[4] Butler, J., "BACnet and Ethernet: Partners for Building Automation" <http://ethernet.industrial-networking.com/building/i12bacnet.asp >

[5] Cisco Systems, "Configuring Simple Network Management Protocol (SNMP)" (2004), URL: <www.cisco.com/en/US/products/hw/contnetw/ps792/products_administration_guide_chapter09186a00801eea34.html >

[6] Kopetz, H. The temporal specification of interfaces in distributed real time systems. In: T.A. Henzinger, C.M. Kirsch (Eds) 2001. Embedded Software (First International Workshop MSOFT 2001 Proceedings), 237-253.

[7] Mahalik, N.P. Fieldbus Technology – Industrial Network Standards for Real-Time Distributed Control. Springer, 2003, 595 pages.

[8] Melore, P., 2004. What is a PLC? URL: http://www.plcs.net/chapters/whatis1.htm

[9] Neilson, C., Delta Controls, Private Communication, September 1st 2004

[10] Newman, H.M., September 1997. "BACnet - The New Standard Protocol" Electrical Contractor. pp. 119-122.

[11] Pigan, R. 2006 Automating With Profinet: Industrial Communication Based on Industrial Ethernet, John Wiley.

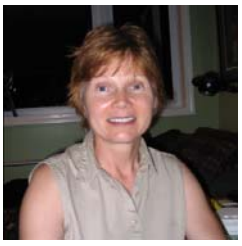
[12] Sudbury Neutrino Observatory, "First Results from SNO - Explain the Missing Solar Neutrinos and Reveal New Neutrino Properties" (June 2001), URL: <http://www.sno.phy.queensu.ca/sno/first_results>

[13] Swan, W., July 1996. "The Language of BACnet – Objects, Properties, and Services" Engineered Systems. Vol. 13, No. 7, pp. 24-32.

[14] Vaillancourt, P. 2005 Network Based Alarms and Monitoring for the Sudbury Neutrino Observatory (SNO), Honours Bachelor of Computer Science Thesis, Laurentian University

[15] Wong, S.S., Hong, S.H., Bushby, S.T., 2003. NISTIR 7038, A Simulation Analysis of BACnet Local Area Networks, *National Institute of Standards and Technology*.

Paul Vaillancourt received his Honours Bachelor of Computer Science degree at Laurentian University in 2005. He has collaborated with the Sudbury Neutrino Observatory throughout his education as well as while working on his thesis entitled "Network Based Alarms and Monitoring for SNO". Paul is currently residing in Canada's national capital Ottawa working as a software developer.



Julia Johnson received her M.Sc. and Ph.D degrees in Computer Science from the University of British Columbia. She is an Associate Professor in the Department of Math and Computer Science at Laurentian University. Her interests include uncertain reasoning based on rough sets for network applications.