Autonomic Intelligent Cyber Sensor to Support Industrial Control Network Awareness

Todd Vollmer, Milos Manic, *Senior Member, IEEE* and Ondrej Linda, *Member, IEEE*

***Abstract*—The proliferation of digital devices in a networked industrial ecosystem, along with an exponential growth in complexity and scope, has resulted in elevated security concerns and management complexity issues. This paper describes a novel architecture utilizing concepts of Autonomic computing and a SOAP based IF-MAP external communication layer to create a network security sensor. This approach simplifies integration of legacy software and supports a secure, scalable, self-managed framework. The contribution of this paper is two-fold: 1) A flexible two level communication layer based on Autonomic computing and Service Oriented Architecture is detailed and 2) Three complementary modules that dynamically reconfigure in response to a changing environment are presented. One module utilizes clustering and fuzzy logic to monitor traffic for abnormal behavior. Another module passively monitors network traffic and deploys deceptive virtual network hosts. These components of the sensor system were implemented in C++ and PERL and utilize a common internal D-Bus communication mechanism. A proof of concept prototype was deployed on a mixed-use test network showing the possible real world applicability. In testing, 45 of the 46 network attached devices were recognized and 10 of the 12 emulated devices were created with specific Operating System and port configurations. Additionally the anomaly detection algorithm achieved a 99.9% recognition rate. All output from the modules were correctly distributed using the common communication structure.**

***Index Terms*—autonomic computing, industrial ecosystems, service oriented architecture, network security, control systems**

I. Introduction

THE expanding reach and wide deployment of digital systems into our most fundamental physical infrastructures pose both a complexity and security concern [1],[2]. The complexity of modern industrial networks primarily stems from three areas: presence of heterogeneous hardware and software, dynamic network composition and usage patterns, and decentralization of control [3]. Dealing with these complexities requires a solution that is flexible and manageable.

An Autonomic Digital Ecosystem (DE) is a model for future production systems that builds on the notion of autonomic self-management by embedding exploitable control features within modules [4]. Cyber physical ecosystems (CPE) include interconnected subsystems that sense and act upon the physical world to form a complex system of systems. These CPE's are a foundational layer of a wider more encompassing digital ecosystem. CPEs typically bridge the cyber world of computing and communications with the physical world by embedding wireless sensor nodes into the physical world [5]. Therefore wireless sensor networks (WSN) are a fundamental building block of digital ecosystems.

DE's utilize large amounts of distributed data gathered from the physical devices found in WSN's. Despite the large actual and planned deployment of wireless devices a large core of wired connectivity is still prevalent. For reliability, security and financial reasons WSN's eventually attach to a physical wired connection. This connection boundary is a logical area of security concern that is addressed by the Autonomic Intelligent Cyber Sensor (AICS).

This paper describes a novel implementation of the AICS algorithm designed to provide cyber security and industrial network situational awareness for Ethernet based control network implementations. The focus of the work presented is on the concept of an integrated framework of communication fundamentals derived from Autonomic research and Service Oriented Architecture (SOA). The AICS utilizes collaborative intelligent mechanisms to: 1) identify anomalous network traffic; 2) provide network entity information; 3) deploy deceptive virtual hosts and 4) implement self-configuring modules. Additionally, AICS dynamically reacts to the industrial human-digital ecosystem in which it resides through examination of network traffic and communication of data from external entities. A proof of concept implementation was tested and results are presented.

The remainder of this paper is structured as follows. Section II discusses related work. Section III explains background concepts of autonomic computing and the communication structure. Section IV presents the design architecture of the AICS communication and functional modules. A proof of concept implementation and test scenario is described in Section V. Section VI provides a comparison to similar solutions and an evaluation of AICS. The last section concludes the paper and briefly discusses future work.

Todd Vollmer is with the Idaho National Laboratory, Idaho Falls, ID 83415 USA (e-mail: denis.vollmer@inl.gov).

Milos Manic is with the University of Idaho, Idaho Falls, ID 83402 USA (e-mail: misko@uidaho.edu).

Ondrej Linda is with Expedia Incorporated, Bellevue, WA 98004 USA (e-mail: olindaczech@gmail.com).

II. Related Work

The application of existing security tools to implement self-protection within an autonomic concept has not been widely explored [6]. However the following section describes three recent efforts closely related to the work presented in this paper.

The work presented in [7] pursued a common architectural solution to support phases of a device lifecycle. The authors utilized concepts of evolvable Production systems (EPS) and Service Oriented Architectures to implement a proof of concept in an industrial automation scenario. In contrast to our approach of using IF-MAP, a Devices Profile for Web Services (DPWS) standard was used to define devices. This work concentrated mainly on deployment of production devices with no mention made of security related equipment.

Kyusakov, Eliasson, Delsing, Deventer and Gustafsson discuss the integration of SOAP based web service implementations in resource constrained wireless sensor nodes [8]. They propose removing middleware and deploying the services directly on devices. This approach is shown to be feasible but limited by performance overhead of the communication scheme. A dedicated security device, as proposed in AICS, with possible multiple deployments may prove more feasible by providing dedicated relatively powerful hardware. This assumes that addition of a new device is preferable to adding additional functionality to hardware constrained devices.

A security related architecture implementing virus detection and removal services for digital ecosystems has been proposed. In [9] the authors presented a distributed approach with hosts working in tandem on a network. A service-oriented architecture was suggested as a communication framework. Any host entities finding virus information in the network would share this information with other hosts thus enabling further actions. This communication mechanism is similar to that proposed for use by AICS. However no empirical tests were executed to validate the approach.

III. Background

This section explains fundamental concepts for Autonomic research, Service Oriented Architecture, IF-MAP and D-Bus. In a large digital ecosystem, constituent members have to be able to continuously adapt to unforeseen situations and evolve in an autonomic way without human intervention. Devices are expected to cooperate to accomplish a mission. They become part of a larger, more complex infrastructure where complementary single elements are exploited to achieve an emergent complex behavior.

A. Autonomic Research

Autonomic computing research is a response to the realization that traditional software systems are facing a decreasing benefit from technological advances because the complexities of management and development are overwhelming. The concept of implementing technology designed specifically to continuously manage and optimize the functionality of other technologies is an extension of the notions present in control theory and managed elements. Control theory provides descriptions of closed systems whose components and desired properties are well known and described by linear or nonlinear models. However, when the system is constantly changing and exhibits varying or uncertain information performance may be poor [10]. Autonomic design attempts to enable adaptive and flexible functionality. This design should react to evolving situations and proactively look forward to future demands.

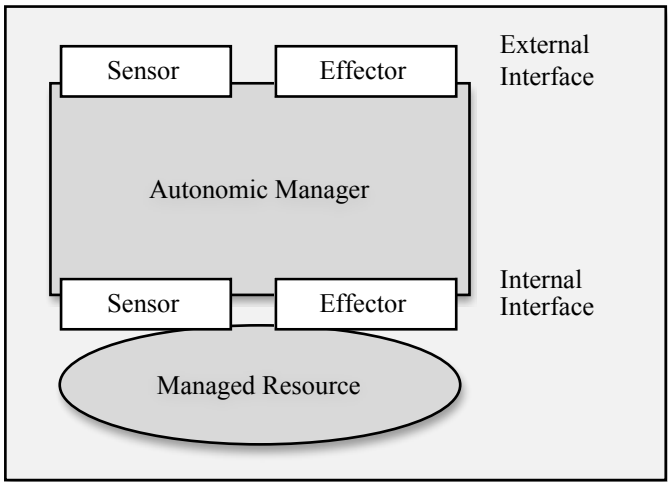

Fig. 1. Autonomic Design

It is the Autonomous Nervous System (ANS), present in human anatomy, which serves as inspiration for computer based autonomic system architectures. The ANS is that portion of the nervous system concerned with regulation of activity of visceral functions generally not controlled by conscious thought. However some actions, such as breathing, work in tandem with active control. ANS is typically divided into sympathetic and parasympathetic divisions. These divisions function in opposition to each other in a complementary manner. The sympathetic division typically functions with actions requiring quick responses. The parasympathetic division functions with actions that do not require immediate reaction. Together the systems can work in concert to achieve homeostasis of the relevant activity [11].

Autonomic system architectures consist of a dynamic collection of autonomic elements each performing a constrained function [12]. This architecture is generally composed of an autonomic manager (AM) that controls one or more managed resource elements (RE). The resources themselves may be legacy components or exhibit autonomic features.

A manageability interface composed of sensor and effector facilitates the communication between AM and RE and is implemented with D-Bus for this project. Sensors obtain data from the resources and effectors are used to perform operations on the resource. Additionally this sensor/effector layer is conceptually replicated to provide access to outside entities [13]. Fig. 1 presents the described autonomic element architecture with the two interface layers labeled as inner interface and outer interface. These interface concepts are a critical concept expanded upon later in this document.

A prominent concept from ANS is a standard communication channel that can carry information to processing entities. This communication can occur over a distributed network of nodes similar to those found in modern electronic networks. It has been noted that a dynamic description language is required to fulfill this function. It has been shown that XML is an appropriate choice of implementation [14].

Finally, the description of an autonomic system is incomplete without mention of the self-* properties. A key feature of autonomic systems is the automated management of resources exhibiting self-configuration, self-optimization, self-healing and self-protection characteristics [12]. In order to harness these properties into a cohesive whole, a control loop monitors, analyzes, plans and executes as appropriate.

B. IF-MAP Introduction

As was mentioned in the previous section the communication interface is a critical component of AICS. A messaging interface based on the Interface to Metadata Access Points (IF-MAP) version 2.0 standard was implemented. IF-MAP is an open protocol standard published by the Trusted Computing Group [15]. Originally made available in April 2008, version 2.0 rev. 47 was published November 2011. It utilizes a publish, subscribe messaging paradigm implemented with the Simple Object Access Protocol (SOAP). The design goal of the IF-MAP working group was to enable the sharing of data between network devices and systems.

IF-MAP has multiple defined parts: a base protocol and metadata for a specific usage. Currently the only metadata defined is for network security, which is sufficient for use in AICS [16]. One of the benefits provided by this definition is a common reference standard of network security data. In addition to the currently defined metadata, it is possible to create a new metadata specification if the base protocol is followed.

1) Base Protocol and Metadata

The base protocol specifies possible actions for clients. IF-MAP clients have access to three actions for metadata: publish, search and subscribe. Clients store or publish metadata into a MAP (Metadata Access Point) for others to consume. A search allows clients to obtain published metadata from the MAP based on a flexible criterion. Subscribe requests asynchronous results from a predefined search whenever a client publishes new metadata. In this way clients can monitor for specific relevant events and be alerted when one occurs. It should be noted that all IF-MAP operations and data types are expressed utilizing XML. This helps ensure a consistent format of the data when it is exchanged between multiple diverse parties.

The protocol also defines two species of data: metadata and identifiers. Metadata in this context is any shared data about network devices, policies, status, behavior or observed relationships. The five defined identifiers are identity, IP Address (v4 and v6), MAC address, Access Request and Device. Identifiers are used to refer to specific metadata items. The metadata of interest to this project are as follows.

- **ip-mac**: A binding between an IP address and a mac address valid for a time frame.
- **device-characteristic**: An inherent characteristic of an entity such as its manufacturer or OS.
- **device-ip**: The IP address of an associated device.
- **discovered-by**: A link identifying which sensor device has discovered a new IP or MAC address.
- **event**: Activity of interest on a network such as a virus attack or abnormal behavior.

2) SOAP

All IF-MAP actions are transmitted using SOAP. The SOAP version 1.2 protocol is a specification utilizing XML for exchanging structured and typed information in a distributed environment. Typically Hypertext Transfer Protocol (HTTP) or SMTP is used for message handling. It provides an extensible framework for transferring application specific information without specifically detailing the semantics of the data carried. The framework provides required actions to perform on a SOAP message in order to send and receive it. SOAP is typically used as a messaging framework layer of a web based service-oriented architecture (SOA). SOA is an enabling technology for development of large systems in industry. It is a flexible solution that can decrease deployment and maintenance costs [17].

C. D-Bus

In this paper, the use of D-Bus is described for internal communication between sensor components and the external messaging service. D-Bus is a system created for inter-process communication (IPC) that has been used in reconfigurable mobile network devices [18]. It was designed to avoid round trips and allow for asynchronous operation. Conceptually it works in terms of messages, and handles many of the difficult IPC synchronization issues. Several high-level language bindings are available which provides a flexible implementation solution. The following describes the D-Bus system and the terminology commonly used with it.

A *bus* is a virtual path that messages are passed over. Multiple instances of a bus may exist and are managed by a software daemon. Several Linux based operating systems offer two busses, system and session, for use without requiring user configuration. Fine grained user access to buses and actions on the appropriate bus service is available via configuration files. Applications that utilize a bus take on the role of either a server or client. Server processes listen for incoming method requests from clients.

A *method* is made available by publishing it on a message bus. Methods are remotely invoked operations of an object. As is found in object-oriented programming languages, objects contain methods. The input and output parameters need to be defined in advance of publication. A method is registered with the D-Bus daemon under an *interface* and available through an *object path*. Interfaces are a collective group of methods that help define the type of an object.

A *service* is a named collection of methods with an associated .service file. The service file defines an executable program to handle receipt of a message related to a given

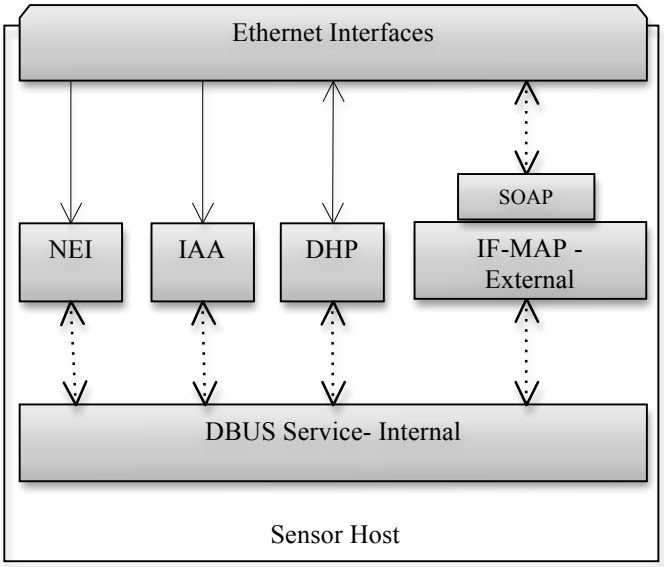

Fig. 2. Sensor Overview

method. If the configured executable is not currently running the message is queued and the process is started. When the process is ready the queued message is delivered. An example service file is found later in section IV A2.

IV. The Architecture of the Autonomic Intelligent Cyber Sensor (AICS)

This section describes the architecture of AICS. Reflecting some of the core attributes of autonomic computing the software structure for this project is broken into functional areas as depicted in Fig. 2. The managed resources are Intelligent Anomaly Assessment (IAA), Network Entity Identification (NEI) and Dynamic Honey Pot (DHP). Because of the flexible internal communication implementation these resources modules can be enhanced or replaced with other potential solutions with little effect on the sensor operation.

The communications infrastructure provides a common interface for additional modules to be deployed on the sensor and is the autonomic manager. As can be seen in Fig. 2, the dotted lines represent messaging paths while the solid lines are direct network connections. Some modules are composed of legacy software that inherently did not communicate via D-Bus. For these components, software wrappers were required. Each individual component is explored in more detail in the following sections.

A. Communication Infrastructure

The communication component is responsible for the external and internal communication mechanism of the sensor.

1) External Communication

External communication is the transfer of information to network entities outside the confines of the sensor system. Examples of this communication include a list of IP addresses to monitor, information on network entities and alerts on abnormal network traffic. Internal communication includes the sharing of information between sensor components. An example includes the passing of passively discovered network entity information from NEI to DHP. All external communication is restricted to the IF-MAP based component with the exception of network packet monitoring and emulated honeypot hosts.

External communications conform to the IF-MAP specification. The published Web Services Description Language (WSDL) document for IF-MAP describes the interfaces available for ad hoc requests to the service. WSDL is an XML-based language for describing Web services and how to access them. The interface is exposed on the network over the SSL secured HTTP protocol. This interface is hosted on a server machine available to the sensor on the network. The Intelligent Reaction on Network Events project produced an open source IF-MAP server called Irond [19]. This Java based program was used as the MAP communication server.

Anomaly alerts and network entity metadata from the sensor system are pushed to the MAP server via an HTTPS based soap call. These asynchronous information pushes are executed upon the identification of behaviors in the network traffic or internal monitoring processes.

In the IF-MAP definition, the metadata type may be either singleValue or multiValue. This is explicitly communicated by setting the ifmap-cardinality attribute in a message. The D-Bus to IF-MAP service for host publication, described later, sets this to singleValue. When a publish request is processed this attribute determines how the MAP server updates the records for a given identifier. Single replaces the information while multi adds to the existing record even if it contains duplicate data. Because singleValue was chosen, a republish of the entire record is necessary even if only one sub-item changes. This simplifies publishing of records at the expense of added communication overhead.

All incoming messages are delivered from the MAP server. This includes configuration or capability querying. The sensor binds to an IF-MAP notify interface for updates and responds accordingly. Consequently asynchronous information is obtained through notify events. These events can be requested by interested outside parties as well. However they are ephemeral in that only subscribed clients will get the information if they attach prior to message publication.

The IAA IP address monitoring list and DHP IP address emulation list are sent via messaging from an external authority. Internally these lists are stored as a configuration file in each component. The stored IP's are then used to extract information from the internal messages created by NEI and used to configure the honeypot hosts. The IAA module tracks the monitor IP addresses for anomalies. The IAA, NEI and DHP modules are explained in more detail later in this section.

The presented two-layer path of communication isolates the sensor from direct network contact with other entities. The externally visible attack surface is limited to one component instead of multiple components. A primary benefit is that any change in external protocols affects only one component. Additionally, it provides the capability to wrap the legacy communication of internal tools that may not be IF-MAP compliant without having to modify the tools source. Finally, it provides a convenient debugging facility.

2) Internal Communication

There is no direct communication between the internal sensor components and the outside MAP server or between modules. All messages, regardless of internal or external destination, use the D-Bus IPC mechanism described in section III. A D-Bus service for centralized processing of external messaging was created. Any module on the sensor host that wishes to send messages externally need only call the sendMessage method of the service. This provides flexibility and security with the additional expense of complexity from adding the D-Bus service. Only the message service needs be concerned with the communication particulars providing for a simplified configuration.

A service file, called org.sa.MessageService.service, created in the /usr/share/dbus-1/services directory specifies what file to execute if the D-Bus service is not already running. This ensures that client messages will be delivered even if the receiving process is not running. The message is queued up in the D-Bus system until the process is started and then delivery ensues. The service is started with the same user id as that which belongs to the caller. This means it is possible to have multiple instances running, one for each user id that makes a call. The service file is shown next.

```
[D-BUS Service]
   Name=org.sa.MessageService
   exec /mnt/Projects/Perl/MessageService/msg_service.pl'
   User=root
```

By default *system* and *session* bus types are provided by the Linux D-Bus daemon. A new *session* bus instance is created for each user login session. This can result in multiple busses or none depending upon interactive user logins. In contrast to the *session* bus, a singleton *system* bus is instantiated upon host startup.

The communication service was attached to the *system* bus as it should not be dependent upon an interactive login. The *system* bus has restrictions where access is explicitly prohibited. Configuration changes were required to allow access for the specific user id associated with the communication service. Additionally, fine grain access control lists are maintained for access to the services provided. This includes who can call the methods and who can retrieve information from the bus.

The following parts of this section describe the managed resource components of the AICS. These components are all clients to the D-Bus internal message service just described.

B. Intelligent Anomaly Assessment

Fuzzy logic provides a framework for system modeling in linguistic form capable of coping with imprecise and vague meanings of words. The Type-2 Fuzzy Logic System (T2 FLS) is an extension of the Type-1 Fuzzy Logic System (T1 FLS) that has been successful in modeling and minimizing the effects of various kinds of dynamic uncertainties. Here the Interval Type-2 Fuzzy Logic System (IT2 FLS), as a specific case of T1 FLS, is used to encode the linguistic domain knowledge about the specific network system and dynamically adjust the sensitivity of the anomaly detection algorithms.

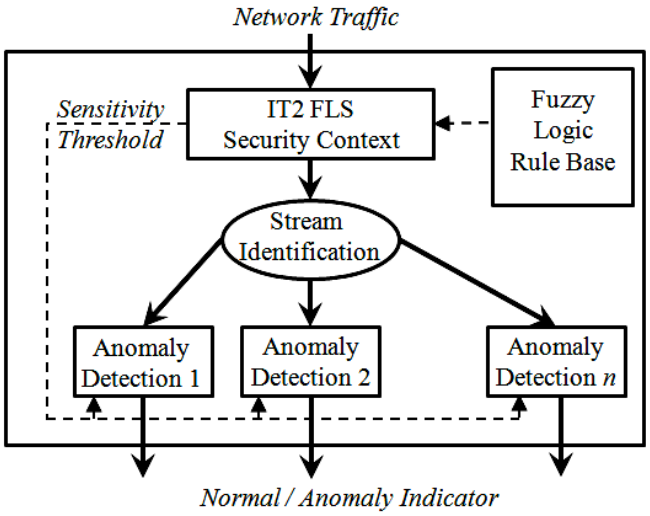

Fig. 3. Anomaly Detection Design

The Intelligent Anomaly Assessment (IAA) component reacts to information from external messaging and passive monitoring of network traffic. An internal bus delivers an externally originated message dictating, by IP address, which hosts are to be actively monitored. The IAA selectively monitors hosts for anomalous behavior while simultaneously utilizing global network trends to adjust its sensitivity. Any anomalous behavior triggers a message passed externally via the internal D-Bus service. Previous work by the authors provides further details on the IAA implementation not presented here [20].

The IAA component is based on an existing low-cost online rule extraction technique [21]. The model is composed of a set of fuzzy rules that are constructed based on a window-based feature vector using an online version of the adapted Nearest Neighbor Clustering (NNC) algorithm. The anomaly detection algorithm was specifically designed to allow for both fast learning and fast classification on the constrained computational resources of an embedded device.

The overall architecture of the proposed anomaly detection system is depicted in Fig. 3. The network traffic is processed by an IT2 FLS that uses a fuzzy logic rule base with encoded linguistic domain knowledge to calculate the cyber-security context. This cyber-security context expresses the belief that an intruder is currently present in the system.

In the next stage, the network traffic is separated into individual communication streams. Each stream is delineated by an IP address retrieved from the internal message bus. Other features, such as port numbers or protocol types, can also be used. Packets assigned to individual communication streams are then passed into dedicated anomaly detection algorithms. Each anomaly detection algorithm maintains its own buffer of incoming packets, which is used to extract a set of window-based features.

The windowing technique extracts statistical features from a limited set of consecutive packets and maintains them in a vector. The window is shifted over the stream of incoming

network packets. As the next arriving packet is inserted into the window, the last packet is removed from the end. The new feature vector is then computed from all of the packets currently present in the window.

As described in [20],[21] the fuzzy logic based anomaly detection algorithm is used to assign a real value to each input vector. This value expresses the belief that the current packet window contains intrusive packets. The closer a value is to 1 the more confident the algorithm is that there is an intrusion present.

The final classification is performed by comparing the real-valued output to a sensitivity threshold. When the real-valued output is above the sensitivity threshold, a network anomaly is reported for the specific communication stream. When the output value is below the sensitivity threshold the network traffic is marked as normal. The actual value of the sensitivity threshold is dynamically computed based on the cyber-security context computed by the Interval Type-2 (IT2) FLS.

C. Network Entity Identification

The Network Entity Identification (NEI) process passively monitors network traffic from which it extracts the IP and MAC address, ports and possible OS identification of hosts. This network entity information is updated on a continuous basis and made available, as messages, to both internal sensor components and external communications.

The software tool used for extraction of the network host information is Ettercap [22]. This tool was identified for use in prior work because of its accuracy, capabilities and XML output [23]. Ettercap is an extensible network manipulation and reconnaissance tool. Its capabilities extend well beyond fingerprinting which is both a risk and advantage. The risk lies in readily providing functionality to a user with nefarious intentions if the sensor system itself is compromised. However the advantage is easily extending future functionality with abilities already present on the system.

Ettercap is run as a daemon with unified sniffing. This mode produces a binary log file. Etterlog is then run on the log file with a -x option to produce an XML file. This file is the information source for communication of the discovered host entities to external interfaces and is placed on the internal message bus for delivery.

Two implementations of an IF-MAP metadata host entity message structure were pursued. The first followed the predefined IF-MAP network security schema and the second utilized a custom addition to the schema. This modification is allowed in the specification. A third possibility, that was not pursed, is creating an entirely new predefined metadata with any necessary custom additions.

The second implementation was necessary because two fields are not available in the predefined schema. These are port protocol i.e. UDP, TCP, etc. and the service name. The port protocol is important for the DHP component. As DHP is the primary internal consumer for this information it was considered important to retain the second option. A sample message is found in Section V Step 1.

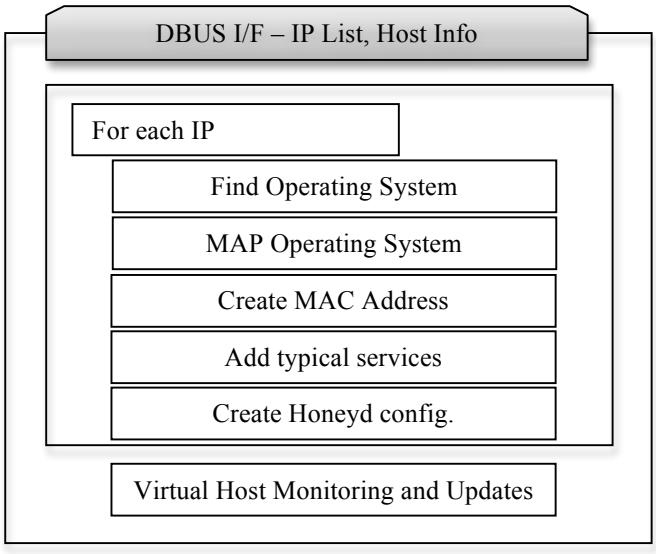

Fig. 4. Dynamic Virtual Honeypot Pseudo Code

D. Dynamic Honeypot

In previous work of the authors an algorithm was proposed and demonstrated to automatically deploy deceptive virtual network entities, or Honeypots, in a control system network [23]. The algorithm creates unique network stack signatures with information provided by the NEI component. Continuous monitoring by NEI is then utilized to reconfigure the virtual hosts in response to changes. This work was leveraged to implement the DHP component of the AICS implementation. Specifically it was chosen as hybrid anomaly detection honeypot systems have been shown to improve results [24].

The DHP algorithm focuses on managing the complexity of self-configuration by adapting to a deployed network environment. A self-updating model of the network devices is created and maintained by passively monitoring traffic. This model is used to automatically configure deceptive virtual network entities, called honeypots, designed to draw the focus of malicious intent [25]. Given that the NEI component is deployed on the sensor, it is a natural source for the host information required to instantiate a dynamic virtual honeypot. This information is available internally on the message bus and a XML file.

The conceptual architecture for the DHP module is shown in Fig. 4. The two primary activities consist of monitoring for host changes and dynamic updating of the emulated hosts. Passively gathered host information is retrieved from the internal message bus. For each IP address found, a comparison is made to any existing host information. First an operating system is identified. This may include creating an OS signature based on statistics of existing systems on the network or random generation. Next a MAC address for the virtual host is created with a correct vendor identification portion. Care is taken not to create duplicate MAC addresses. Device specific information files are then consulted for inclusion of typical services. Finally, the existing virtual host configurations are reviewed and updated as needed.

As part of the monitor and update process messages are delivered internally via D-Bus providing information on the simulated hosts. This information may then be leveraged by

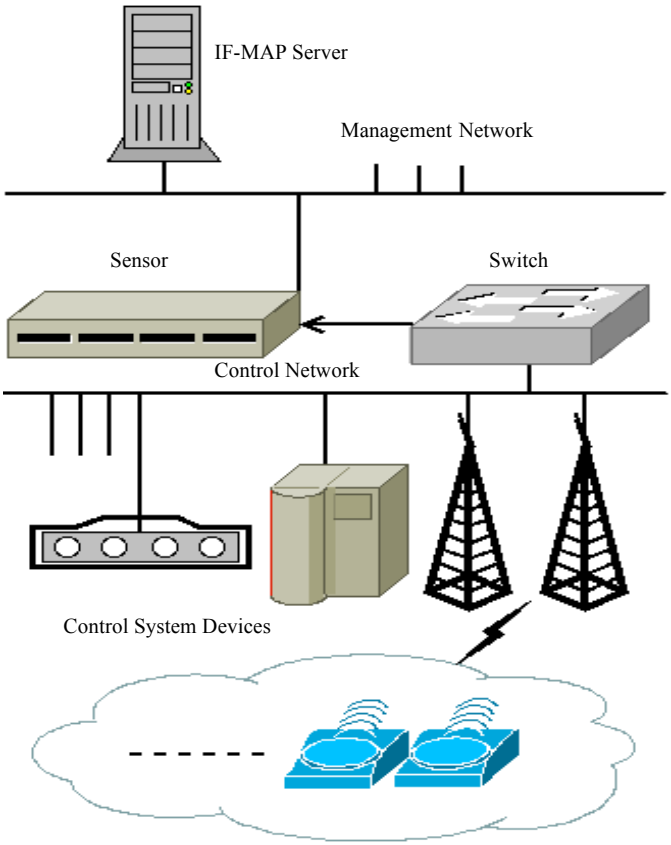

Fig. 5. Network Layout

external components to differentiate virtual hosts from real systems on the network.

V. System Setup and Test Scenario

This section describes the test system, a test scenario and empirical results. First, the network layout and sensor hardware platform are shown. Second, a five-step testing scenario is summarized. Third, each testing step and its execution is described in more detail. The description includes a performance evaluation of the relevant components and sampled IF-MAP messages.

A. Test Network and Hardware Setup

A small campus grid (SCG) and sensor network that physically resides at the Center for Advance Energy Studies in Idaho Falls, Idaho was used as a test platform. The network consists of a heterogeneous mixture of 46 directly connected devices, including wireless sensors monitoring environmental conditions in the building. The SCG is connected to a small wind turbine, a solar power station, a wireless Advanced Metering Infrastructure (AMI) with two wireless access points (WAP) and a variety of control system devices. Twelve smart meters, which wirelessly connect to the WAP's, are physically attached to test harnesses to generate a signal. The network has several Windows based computers, web camera's, a Rockwell Automation Micrologix 1100 PLC and a National Instruments PLC. The Micrologix controls or monitors 6 lighted push buttons, 7 lights, 2 potentio-meters, 2 temperature sensors and a small electric fan which constitutes both digital and analog input/output points. All of the network hosts are part of a single network segment directly attached to a Cisco network switch with a SPAN port. The SPAN port receives a copy of all network traffic and is used for monitoring by AICS.

The SCG includes access to three wireless sensor networks from the following commercial vendors: Emerson, Honeywell and Arch Rock. Each one of these three systems connects to wireless environmental sensors via one or more wireless access points. As with the previously mentioned AMI deployment, the WAP gateways have both a wired network connection and a wireless interface to the remote sensors. The multitude of environmental sensors is not directly visible to the wired test network. However the wired side of the WAP communication is visible to the AICS device. Several data historian hosts attach to the WAP's on the wired side and retrieve the environmental data using the appropriate protocol. Each vendor has a unique implementation of a wired network protocol built on top of Ethernet. A representation of the test network and placement of the sensor is shown in Fig. 5.

The implemented AICS software was deployed on a test hardware platform. This platform runs a 32 bit Ubuntu 12.04 LTS OS on a recently released Via AMOS-5002 fanless compact embedded system. It consists of a VIA dual core EdenX2 (U4200) CPU, 4 GB's of DDR3 1333MHz RAM, a 320GB SATA II hard drive and 3 Gb Ethernet ports. One of the Ethernet ports (not shown in Fig. 5.) was assigned to the dynamic virtual honeypot for emulation of network hosts on the control network. A second interface was dedicated to passive monitoring on the switch SPAN port by the anomaly detection and entity identification modules. The third interface was utilized to provide external IF-MAP communications on the management network for the sensor. This hardware was chosen for its flexibility of deployment in industrial/commercial environments and a relatively strong computational capability.

B. Test Scenario

The steps in Fig. 6 provide a scenario devised to test the communication and managed resource functionality of AICS when deployed on the SCG network. Each step provides a

Step 1: Attach sensor to SCG network and initiate system.
- Evaluate NEI host identification model dynamic updates and related IF-MAP messages.

Step 2: Send IP lists to DHP Component
- Test dynamic virtual host creation based on IP list and utilization of NEI host information from step 1.

Step 3: Target Network probes at DHP hosts.
- Verify emulated network presence of devices.

Step 4: Send IP list to IAA anomaly behavior monitoring.
- IAA initiates learning mode, creates normalization and clustering values for fuzzy rule creation.

Step 5: Send monitor message to IAA and initiate network attacks.
- IAA recognizes attacks and sends IF-MAP alerts.

Fig. 6. Test Scenario Steps

high level action followed by the primary functionality to be evaluated. In general, verification is accomplished by inspection of IF-MAP messages or local component data stores. The results of the verification and other related information is provided in section C.

Two test tools were created to facilitate execution of the scenario. Tool one provides the input IP addresses to the MAP server for subsequent delivery to both the DHP and IAA. Tool two monitores the MAP server for published messages from the sensor.

During execution of the scenario, traffic from the control network was captured on the switch's SPAN port and preserved in a pcap formatted file. The captured data provides a consistent baseline for repeat system evaluation. In the presented scenario and results, the sensor was attached to the live system unless otherwise indicated. The preserved traffic was subsequently used to perform evaluations on the AICS sensor. The test results from the system, using this stored data, produced no discernable differences with those found in the live scenario. In other words, the results are repeatable and deterministic.

C. Scenario Execution and Results

Step 1- Attach sensor to network and power on.

As the sensor started to recognize hosts on the network the NEI component created its internal model and produced host messages. The internal host model is stored as an XML file and made available to the DHP component through messaging. Updated host messages are published to the MAP service every 60 minutes if a change occurs. During the complete test cycle, 45 of the 46 devices were identified by NEI. The single unidentified host used a custom protocol that does not utilize IP addresses and consequently was not recognized by NEI. The host communicates using raw Ethernet frames and is characteristic of a Honeywell proprietary protocol. A sample captured IF-MAP NEI host message, slightly modified for space, follows:

```
<metadata>
  <meta:host ifmap-cardinality="multiValue" ifmap-
  publisher-id="test--137625522-1" ifmap-timestamp="2012-
  03-26T14:40:33-06:00">
    <start-time>2012-03-26T14:40:00</start-time>
    <mac>00:12:3F:61:4A:69</mac>
    <ip>192.168.99.220</ip>
    <os>D-Link DWL-900AP</os>
    <port_list>
      <port proto="udp">137</port>
      <port proto="udp">138</port>
    </port_list>
  </meta:host>
</metadata>
```

Step 2 – Send IP lists to DHP component.

A PERL implementation of the DHP algorithm was run on the test sensor platform. Twelve heterogeneous systems, shown in the second column of Table I, were evaluated. The ID column is used as a reference identifier and corresponds to the last octet used in the emulated IP address. These hosts represent a variety of equipment found on the test network including network switches, PLC's and WAP's. Using tool one, an IF-MAP message containing the twelve host IP addresses was published to the IF-MAP server. DHP retrieved the list and used the IP's as a key to lookup host information in the NEI internal model file.

TABLE I
HOST IDENTIFICATION RESULTS

ID	Device	Mapped OS
1	Rockwell HMI	MS Windows ME, 2000 Pro or Advanced Server or Windows XP
2	Micrologix 1100 PLC	Novell NetWare 3.12 - 5.00
3	Arch Rock Server	Random
5	Honeywell HMI	MS Windows ME, 2000 Pro or Advanced Server or Windows XP
10	Arch Rock WAP	Random
25	D-Link WAP	Apple Airport Extreme Base Station (WAP)
99	D-Link Wireless camera	Apple Airport Extreme Base Station (WAP)
130	Arch Rock HMI	MS Windows ME, 2000 Pro or Advanced Server or Windows XP
150	Nat. Inst. PLC	MS Windows ME, 2000 Pro or Advanced Server or Windows XP
200	Emerson WiHart AP	Linux 2.4.16 - 2.4.18
215	HMI(Windows PC)	Windows for Workgroups 3.11 / TCP/IP-32 3.11b stack or Windows 98
253	Moxa 505A Switch	FreeBSD 4.4 for i386 (IA-32)

The DHP component, using the NEI information, was able to automatically create virtual hosts for all twelve of the network-attached devices. Each emulated host was assigned its own unique IP and MAC address and was instantiated on the test sensor hardware. Furthermore, as the NEI component became cognizant of more host details, each related virtual device managed by DHP was updated to reflect the new information. For example, when a host started communicating on a new port.

Of the twelve devices chosen for emulation, ten operating systems were configured autonomously and two were assigned as 'random'. If NEI is not able to identify an OS, then DHP randomly choses a value from the list of identified OS's in the host model file. The assumption is that devices on a network are likely to have OS's similar to each other. Another option is to randomly pick one of the thousand OS's defined in Honeyd. The third column in Table I shows the mapped OS names selected by the algorithm. Host IF-MAP messages similar to those in Step 1, only with a different identifier to differentiate real from emulated hosts, were created and published to the MAP server for the twelve emulated hosts.

Step 3 – Target network probes at 12 DHP emulated hosts.

This step verified, from a network perspective, that the emulated hosts exist and at a superficial level appear as separate, diverse devices. In addition to the OS emulation performance exercised in Step 2, four network test probes with Nmap were completed. Nmap is a network scanning tool that is commonly used to find attached hosts, scan ports and

TABLE II
IAA CONFIGURATION VALUES

Configuration Item	Value
Data Window Length	20
Data Dimension Input	18
Data Dimension Output	17
Data Normalization Input Length	1000
Normalize Data	Yes
Cluster Spread	2.0
Cluster Blur	0.2
Maximum Cluster Radius	0.25

identify operating systems [26]. The four probes included OS detection, IP protocol evaluation, ICMP echo 'pings' and UDP/TCP port scanning.

All twelve of the emulated test devices were found with ICMP and responded within 2.2 seconds. Nmap OS detection produces several possibilities for each target. For the 12 emulated devices, a total of 223 possible device types and 40 OS matches were returned. The reported accuracy of OS and device identification ranged from 85% to 97%. As there were multiple results for most of the emulated devices, any of the entries that matched either the original device or its mapped OS were considered a success. Of the ten non-random devices, Nmap correctly identified seven for a 70% success rate. Of the three that failed, no information was produced for device 2 in Table 1. Twenty-one incorrect entries were created for device 215. Finally, Device 5 was identified by one incorrect entry.

Step 4 – Send IP List and monitoring request to IAA.

Three IP addresses were selected for monitoring by a C++ implementation of the IAA component. A message containing the IP's was sent to the MAP server with tool one. IAA retrieved this message and started passively monitoring for the appropriate network traffic. For evaluation purposes, the training mode was constrained to contain 100,000 packets recorded during normal network usage. An isolated network was maintained to ensure the normality of the recorded data.

The configuration of the IAA fuzzy logic based anomaly detection routine is maintained in an XML file. This allows for exploring the effect of changing key parameters for clustering, fuzzy rules, network packet windows and more. Prior work exploring these variables can be found in [20],[21]. The values used for the test scenario are provided in Table II. 17 features were derived from a 20 packet network buffer. These features were normalized and clustered using an online nearest neighbor algorithm with a radius of 0.2. The resulting clusters, for the three individual IP communication streams, were composed of 19, 57 and 2 cluster members. These cluster sizes reflect the heterogeneous network behaviors of the different devices.

Step 5 – Send monitor message and initiate network attacks.

Following the collection of training information in the previous step, 200,000 packets with abnormal behavior comingled with normal behavior were evaluated. This set of data included the Nmap tests that were run during the DHP testing in Step 3.

TABLE III
ANOMALY PERFORMANCE

Threshold	Correct Rate	False Pos.	False Neg.
STREAM 1			
0.3	99.8539%	0.1461%	0.0000%
0.6	99.8705%	0.1295%	0.0000%
0.9	99.8788%	0.1212%	0.0000%
IT2 FLS	99.8722%	0.1278%	0.0000%
STREAM 2			
0.3	99.9037%	0.1217%	0.0275%
0.6	99.5504%	0.1082%	1.3753%
0.9	99.3799%	0.1082%	2.0079%
IT2 FLS	99.9111%	0.1116%	0.0275%
STREAM 3			
0.3	99.8643%	0.2953%	0.0000%
0.6	99.8960%	0.2265%	0.0000%
0.9	99.8960%	0.2265%	0.0000%
IT2 FLS	99.8960%	0.2265%	0.0000%

The Nmap and Nessus software applications were used to generate anomalous network traffic behavior. Nessus provides auditing capabilities, system vulnerability assessments, and profiling information [27]. The simulated intrusion attempts included but were not limited to: ARP pings, SYN stealth scans, port scanning and control system device probes. Cyber attacks ranged from long attacks composed of many packets to short attempts of single packets.

Table III depicts the results of the anomaly detection for the streams. The three rows labeled 0.3, 0.6 and 0.9 show the performance with a fixed sensitivity threshold value. The IT2 row provides the performance with domain knowledge encoded in form of fuzzy rules performing dynamic adjustment of the sensitivity threshold. The correct classification, the false negative and the false positive rates were used as performance measures. The correct classification rate is the percentage of correctly categorized data packet instances. The false negative rate is the ratio of incorrectly labeled normal behavior inputs and the overall number of normal behavior instances. The false positive rate is the ratio of incorrectly labeled anomalous inputs and the overall number of anomalies.

An example asynchronous IF-MAP message describing abnormal behavior is shown next. This message provides the device name of the sensor and describes an anomaly found between two IP addresses. The .220 address is one of the three being monitored and the .10 was the origination of the attacker. The proto_list shows that the 20 packet window contained UDP and ICMP protocol packets.

```
<notify>
  <device><name>CS:ID1:002</name></device>
  <metadata>
    <event ifmap-cardinality="multiValue">
      <disc-time>2012-03-26T14:50:00</disc-time>
      <confidence>1.0</confidence>
      <type>behavior</type>
      <window-size>20</window-size>
      <ip_list>
```

```
            <ip>192.168.99.220</ip>
            <ip>192.168.99.10</ip>
          </ip_list>
          <proto_list>
            <proto>1</proto>
            <proto>3</proto>
          </proto_list>
        </event>
    </metadata>
</notify>
```

VI. Comparison to Existing Work and Evaluation

A related anomaly detection project by P. Dussel et al. relies on the computation of similarity between transport-layer packet payloads embedded in a geometric space [28]. They performed experiments on traffic from a SCADA test bed containing various application-layer protocols. A primary difference from AICS includes the use of packet data, as opposed to header derived information, to produce *n*-gram populated feature vectors. A simple distance measure is then compared with stored 'normal' vectors to determine if an anomaly exists in the traffic. The AICS IAA algorithm maintains two models of system behavior: a global threat model that considers all traffic and a set of individual models for a given stream. Their solution appears to broadly apply itself to all traffic. It is difficult to compare results with different test data sets but their average detection rate of 88%-92% at a false positive level of 0.2% may be due to the differences just mentioned.

Another approach taken in [29] is similar to the AICS architecture. The design incorporates a self-organizing map (SOM) anomaly detection routine and SOAP communication. The routine is embedded on a small sized low power MOXA device with two wired Ethernet ports. This implementation is comparable to work done at the genesis of the AICS solution [21]. A binary XML (MTOM) version of SOAP is used for eternal messaging. The detection rate is apparently affected by the limited hardware. As network traffic increases the anomaly detection rate decreases. For example, the detection rate is 80% with a 6 Mb/s network load. It was not clear if this was a result of the feature extraction algorithm or a possible loss of packets captured.

In addition to the different computational intelligence routines utilized, the AICS algorithm is effectively an evolution of this design that takes advantage of improved hardware. Furthermore, AICS is able to provide complementary services such as virtual honeypots (DHP) and host identification (NEI).

The sensor host device setup requires physically connecting the device to the networks. There are three physical copper network ports on the device. Eth0 was connected to the network switch SPAN port in read only mode. The NEI and IAA components shared this monitoring port. Eth1 was connected to the management network for read and write. This port was utilized for IF-MAP communications. Eth2 was connected to the operation network and assigned for use by the virtual honeypots. The average poll cycle of the wireless monitoring systems occurred once every minute. Playback of captured test data at rates from 0.6 Mb/s to 20 Mb/s showed no discernable difference in the AICS average detection rate. The first observable accuracy issue occurs when the physical network port capacity is overrun.

The use of IF-MAP by AICS is an improvement over the previously mentioned designs. IF-MAP provides a predefined data dictionary for network security information that eases integration efforts with other components. The modular nature and common communications infrastructure provides a flexible platform for changing functionality. The AICS solution delivers observed information via a published interface for integration with a system wide solution. While the components are self-configuring, the data used to perform that configuration comes from passive observations. Because of the IF-Map communication scheme, host information could be provided by external entities. For instance, information about host entities could be published on the IF-MAP SOAP interface. This broadens the potential use of AICS in a more sophisticated hierarchical security system.

AICS was developed as a deployable modular framework on a commonly available hardware platform with components that provide network host information, identify anomalous network traffic behaviors and deploy dynamic virtual honeypots. In the test scenario, the sensor was deployed on a self-contained hardware platform. This effectively hides the communication between components from other network entities, thereby adding another layer of defense. When considering the possible addition of information from external hosts on the IF-MAP interface, a balance between security and information access would need to be evaluated.

An additional beneficial outcome of the presented solution is removal of the configuration burden from the human operator. This capability stems from the self-configuring aspects of an Autonomic design. For example, the IAA anomaly detection routine doesn't require human created rules. It learns normal behavior from observations of the system. Additionally the dynamic honeypots are configured from system observations. This functionality reduces dependence on network expertise and level of human configuration effort.

A. Hardware and Software Performance

The CPU and RAM utilization was monitored during the test scenario. As the hardware platform is a fanless system, temperature data was recorded as well. The 200,000 anomaly evaluation packets contained 19,627,063 bytes of payload information. Approximately 97 packets per second were processed at normal network transmission speed. It was observed that each anomaly alert sent from IAA on the IF-MAP mechanism incurred an overhead of .7ms of communication processing time.

The CPU thermal sensors were polled every 5 seconds. Room temperature at the start of the test was 23C. Core 0 temperature after a 5-minute warm-up period was 44C. Values during the test ranged from 42C to 47C. Core 1 temperature during the warm-up period was 45C. The test values ranged

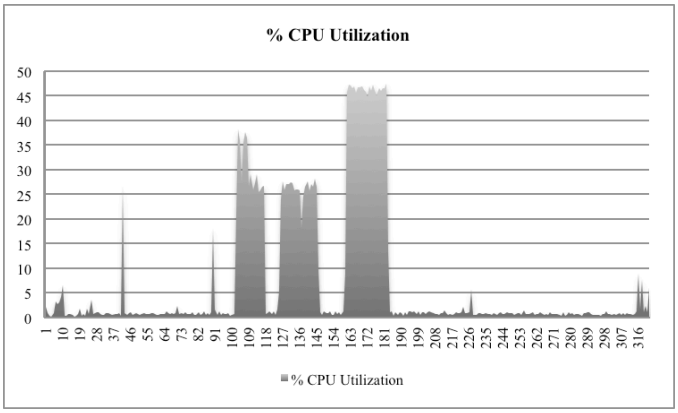

Fig. 7. Sensor CPU Utilization Chart

from 43C to 48C. It is speculated that the difference in the range low value and the initial start might be due to variations in room temperature during the test.

The at-rest CPU utilization, as measured by the Linux top command on one-second intervals, was 1.1%. This was due mainly to the Xorg process providing HMI services. The utilization maximum during the test was 47.4% with an average of .65% and standard deviation of 2.29%. The maximum occurred when the virtual honeypots were probed with Nmap. The graph in Fig. 7 shows the sampled CPU utilization before and after the maximum that occurred between time index 100 and 181. Another key performance indicator for Linux machines is the amount of time the CPU has been waiting for I/O to complete. The maximum value was 10% with an average of 0.59% and standard deviation of 0.62%. In other words, the machine was not overwhelmed with the test data when running in real time.

System memory on the sensor platform at the beginning of the test, prior to initiating network activity, was 801,516 kilobytes. Peak overall usage was 881,308 kilobytes. DHP related processes consumed a steady 20 megabytes of information. Internal messaging averaged 4 megabytes. IAA related processes used 13 megabytes. Finally, the NEI components utilized a consistent 36 megabytes of memory. Overall continued growth of sensor memory usage was attributed to non-sensor related processes such as Xorg. Turning off unneeded services, such as graphic management, would reduce the memory footprint.

VII. Conclusion

Inspired by self-configuring aspects of Autonomic computing, the work presented here supports the important goal of securing industrial ecosystems by providing network security awareness in a heterogeneous control system network. Contributions of this paper include: 1) A flexible two level communication layer based on Autonomic computing and Service Oriented Architecture and 2) Descriptions of three dynamic modules that respond to a changing environment.

A novel working sensor prototype was developed based on a unique implementation of the Trusted Network Group's IF-MAP web services based communication protocol. Sensor communication between self-contained modules is accomplished with D-Bus. At multiple steps of a test scenario, the communication layer was utilized to provide information in a well-defined format between components and to external entities.

As can be seen in the architecture and scenario sections, the self-configuration capability of Autonomic systems is exemplified by the IAA and DHP components. The DHP host entities are created automatically based upon passive monitoring of network activity. Anomaly behavior in IAA is detected by clustering of information retrieved from a moving window of traffic. The only source of outside information explicitly provided is the IP addresses of devices deemed to be critical in the functioning system. This information is asynchronously delivered by a decoupled two-part communication system.

Multiple intelligent modules were deployed on a test system to monitor for anomalous behavior and create deceptive emulated hosts. The modules are exemplary implementations of self-learning components. In a test scenario, 45 of the 46 network attached devices were recognized and 10 of the 12 emulated devices were created with representative characteristics. Additionally, 99.9% of anomalous packets were recognized. The modules utilized the communication layer to provide notifications and retrieve information.

Future work includes integration of AICS with external components utilizing the IF-MAP standard. These components include functionality to correlate information from the sensor with production data. Multiple sensors will be deployed necessitating further use web standards such as WS-Management.

References

[1] V. C. Gungor and G. P. Hancke, "Industrial wireless sensor networks: Challenges, design principles, and technical approaches," *IEEE Trans. Ind. Electron.*, vol. 56, no. 10, pp. 4258-4265, Oct. 2009.

[2] M. Cheminod, L. Durante, and A. Valenzano, "Review of security issues in industrial networks," *IEEE Trans. Ind. Inform*., to be published.

[3] V.C. Gungor, B. Lu and G.P. Hancke, "Opportunities and challenges of wireless sensor networks in smart grid," *IEEE Trans. Ind. Electron*., vol. 57, no. 10, pp. 3557-3564, Oct. 2010.

[4] M. Ulieru and S. Grobbelaar, "Engineering Industrial Ecosystems in a Networked World," *5th IEEE Int. Conf. on Ind. Inform.*, vol.1, no., pp.1-7, 23-27 June 2007

[5] Y. Yang, Y. Xu, X. Li and C. Chen, "A Loss Inference Algorithm for wirelss sensor networks to improve data reliability of digital ecosystems," *IEEE Trans. Ind. Electron*., vol. 58, no. 6, Jun. 2011

[6] H. Ruan, M. Lacoste and J. Leneutre, "A Policy Management Framework for Self-Protection of Pervasive Systems," *2010 6th Int. Conf. on Autonomic and Autonomous Systems*, pp.104-109, Mar. 2010.

[7] G. Candido, A. W. Colombo, J. Barata and F. Jammes, "Service-Oriented infrastructure to support the deployment of evolvable production systems," *IEEE Trans. Ind. Inform.,* vol. 7, no. 4, pp.759-767, Nov. 2011.

[8] R. Kyusakov, J. Eliasson, J. Delsing, J. Deventer and J. Gustafsson, "Integration of Wireless Sensor and actuator nodes with IT infrastructure using service-oriented architecture," *IEEE Trans. Ind. Inform*., accepted for publication March 6, 2012.

[9] R. Agrawal, W. Grosky and F. Fotouhi, "Virus detection and removal service architecture in digital ecosystems," in *Proc. IEEE Conf. on Digital Ecosystems and Tech*., pp. 301-305, June 2009.

[10] S. Dobson, et al., "A survey of autonomic communications," *ACM Trans. On Autonomous and Adaptive Systems*, vol. 1, no. 2, pp. 223-259, Dec. 2006.

[11] P. Brodal, *The Central Nervous System*, Oxford University Press, Jan 15, 1998
[12] IBM, "An architectural blueprint for autonomic computing (4th ed.)," white paper, June 2006; http: //www-03.ibm.com/autonomic/pdfs/AC Blueprint White Paper 4th.pdf
[13] Y. Cheng, A. Leon-Garcia and I. Foster, "Toward an autonomic service management framework: A holistic vision of soa, aon, and autonomic computing," *IEEE Comm. Magazine*, vol. 46, no. 5, pp. 138– 146, 2008.
[14] H. Shuaib, R. J. Anthony, and M. Pelc, "Towards certifiable autonomic computing systems," TR 1, Autonomic Research Group, CMS, University of Greenwich, 2010
[15] *TNC IF-MAP Binding for SOAP*, Trusted Computing Group, ver. 2.0, Nov 2011, available: http://www.trustedcomputinggroup.org/resources/ tnc_ifmap_binding_for_soap_specification.
[16] *TNC IF-MAP Metadata for Network Security,* Trusted Computing Group, ver. 1.0, Sept. 2010, available: http://www.trustedcomputinggroup.org/resources/tnc_ifmap_metadata_f or_network_security.
[17] S. de Deugd, R. Carroll, K.E. Keyy, B. Millett and J. Ricker, "SODA" SERVICE Oriented Device Architecture," *IEEE Pervasive Computing*, vol. 5, no. 3, pp. 94-96, July-Sept. 2006.
[18] A. Merentitis, E. Patouni, N. Alonistioti, and M. Doubrava, "To reconfigure or not to reconfigure: Cognitive mechanisms for mobile devices decision Making," in *proc. IEEE VTC 2008*, pp. 1-5, Sept. 2008.
[19] Irond, URL from April 2012: http://trust.inform.fh-hannover.de/joomla/index.php/ downloads.
[20] O. Linda, T. Vollmer and M. Manic, "Improving Cyber-Security of Smart Grid Systems via Anomaly Detection and Linguistic Domain Knowledge," *5th IEEE Symposium on Resilience Control Systems,* ISRCS 2012, August 2012.
[21] O. Linda, T. Vollmer, J. Wright and M. Manic, "Fuzzy Logic Based Anomaly Detection for Embedded Network Security Cyber Sensor," in *Proc. IEEE Symposium Series on Computational Intelligence*, pp. 202-209, April 2011.
[22] Ettercap network sniffer, URL from April 2012, available:http://ettercap.sourforge.net/.
[23] T. Vollmer and M. Manic, "Adaptable autonomous virtual hosts in an industrial control network," *IEEE Trans. Ind. Inform*., submitted for publication.
[24] P. Garcia-Teodoro, J. E. Diaz-Verdejo, G. Macia-Fernandez and L. Sanchez-Casad, "Network-based Hybrid Intrusion Detection and Honeysystems as Active Reaction Schemes," *Inter. J. of Comp. Sci. and Network Security*, vol.7, no.10, pp 62-70, October 2007.
[25] N. Provos and T. Holz, *Virtual Honeypots*, Boston, MA: Addison-Wesley, 2007.
[26] Nmap webpage [URL], Available: http://nmap.org, from April 2012.
[27] Nessus webpage [URL], Available: http://tenable.com/products/nessus, from April 2012.
[28] P. Dussel, C. Gehl, P. Laskov, J. Buber, C Stormann and J. Kastner, "Cyber-critical infrastructure protection using real-time payload-based anomaly detection," in *Proc. of the 4th Workshop on Critical Information Infrastructures Security,* pp. 43–54, 2009
[29] F. Maciá-Pérez, F. Mora-Gimeno, D. Marcos-Jorquera, J.A. Gil-Martínez-Abarca, H. Ramos-Morillo and I. Lorenzo-Fonseca, "Network Intrusion Detection System Embedded on a Smart Sensor," *IEEE Trans. Ind. Electron.*, vol. 58, no. 3, pp.722-732, March 2011

Todd Vollmer (M'06–SM'13) Todd Vollmer, IEEE Senior Member, received his B.S. and M.S. degrees in computer science from South Dakota School of Mines and Technology in 1996 and 1998 respectively. He is currently pursuing a Ph.D. in computer science from the University of Idaho. Todd has worked as a Research Scientist at the Idaho National Laboratory since 2006. Prior work involved satellite system software development for Lockheed-Martin and DigitalGlobe, Inc. He has several recent published papers at IEEE conferences on computational intelligence and computer security.

Milos Manic (S'95-M'05-SM'06) received the Dipl.Ing. and M.S. degrees in electrical engineering and computer science from the University of Nis, Serbia in 1991 and 1997 respectively, and a Ph.D. degree in computer science from the University of Idaho in 2003. Dr. Manic is an Associate Professor at the University of Idaho. He has over 20 years of academic and industrial experience, including an appointment at the ECE Dept. and Neuroscience program at University of Idaho. As university collaborator or principal investigator he lead number of research grants with the National Science Foundation, Idaho National Laboratory, EPSCoR, Dept. of Air Force, and Hewlett-Packard. Dr. Manic is a Secretary of IEEE Industrial Electronics Society and is a member of several technical committees and boards of this Society such as IES Committees (previously the Boards) for Conferences and for Publications. Dr. Manic has published over hundred refereed articles in international journals, books, and conferences.

Ondrej Linda, (S'09) received his Ph.D. in Computer Science from the University of Idaho in 2012 and his M.Sc. in Computer Graphics and M.Sc. in Computer Science from the Czech Technical University in Prague and University of Idaho in 2010 and 2009, respectively. He is currently working as a Machine Learning Scientist with Expedia Inc. focusing on applications of machine learning in the area of natural language processing and information retrieval. His previous research experience includes research assistant positions at the University of Idaho and Kansas State University and an internship with the Robotics Group at the Idaho National Laboratory. His fields of interest include machine learning, pattern recognition, intelligent control systems, data mining and computer graphics.