Macrothink Institute™

Network Protocols and Algorithms
ISSN 1943-3581
201X, Vol. X, No. X

Integrating Machine Learning Techniques to Adapt Protocols for QoS-enabled Distributed Real-time and Embedded Publish/Subscribe Middleware

Joe Hoffert

Dept. of Electrical Engineering and Computer Science (EECS), Vanderbilt University

400 24th Ave S. - Nashville, Tennessee (United States)

Tel: 1-615-343-8197 E-mail: jhoffert@dre.vanderbilt.edu

Daniel Mack

Dept. of Electrical Engineering and Computer Science (EECS), Vanderbilt University

400 24th Ave S. - Nashville, Tennessee (United States)

Tel: 1-615-343-8197 E-mail: dmack@isis.vanderbilt.edu

Douglas C. Schmidt

Dept. of Electrical Engineering and Computer Science (EECS), Vanderbilt University

400 24th Ave S. - Nashville, Tennessee (United States)

Tel: 1-615-343-8197 E-mail: schmidt@dre.vanderbilt.edu

Abstract

Quality-of-service (QoS)-enabled publish/subscribe (pub/sub) middleware provides the infrastructure needed to disseminate data predictably, reliably, and scalably in *distributed real-time and embedded* (DRE) systems. Maintaining QoS properties as the operating environment fluctuates is challenging, however, since the chosen mechanism (*e.g.*, transport protocol or caching algorithm for data persistence) may no longer provide the needed QoS. Moreover, some adaptation approaches are tailored for particular types of operating environments, such as environments whose configuration properties (*e.g.*, number of data receivers or data sending rate) are known prior to runtime versus unknown until runtime.

For DRE pub/sub systems operating in dynamic environments, adjustments to mechanisms must be timely, accurate for known environments, and resilient to environments unknown until runtime. Several adaptation approaches, such as policy-based [1] and reinforcement learning [2], have been developed to ensure end-to-end quality-of-service (QoS) for enterprise distributed systems in dynamic operating environments. Not all approaches are applicable for DRE pub/sub systems, however, due to their stringent accuracy, timeliness, and development complexity requirements.

Supervised machine learning techniques, such as *artificial neural networks* (ANNs) [3] and *support vector machines* (SVMs) [4], are promising approaches to address the accuracy, time complexity, and development complexity concerns of adaptive enterprise DRE systems. This article describes the results of research that (1) empirically evaluates supervised machine learning techniques used to adapt the transport protocols of QoS-enabled pub/sub middleware autonomically in a dynamic environment and (2) integrates multiple techniques to increase accuracy for environments known *a priori* and not known until runtime. Our results show that both ANNs and SVMs provide constant time complexity, low latency, and reduced development complexity. ANNs are generally more accurate in providing adaptation guidance for environments whose properties are known prior to runtime and provide sub-μsec response times, whereas SVMs provide higher accuracy with μsec latencies for environments whose properties are not known until runtime. Both approaches can be leveraged together with QoS-enabled pub/sub middleware to address the timeliness, accuracy, and development complexity needs of enterprise DRE systems executing in dynamic environments.

Keywords: Adaptation of Transport Protocols, Artificial Neural Networks, Autonomic Adaptation, Event-based Distributed Systems, Publish/Subscribe Middleware, Supervised Machine Learning, Support Vector Machines.

1 Introduction

1.1 Emerging trends and challenges.

Enterprise *distributed real-time and embedded* (DRE) publish/subscribe (pub/sub) systems provide the management of critical resources and data for the ongoing objectives of projects and organizations. Examples include shipboard computing environments, air traffic management systems, and recovery operations in the aftermath of regional or national disasters. These enterprise DRE pub/sub systems often modify their operational behavior based on their external environment. For example, search and rescue missions as part of disaster recovery operations can adjust the image resolution of captured video streams used to detect and track survivors depending on the resources provided by the environment (*e.g.*, computing power, network bandwidth) [5].

Many enterprise DRE pub/sub systems autonomically[1] (1) monitor their environment and (2) adjust their operational behavior as the environment changes since manual adjustment is tedious, slow, and error prone. For example, a shift in network reliability can prompt *quality-of-service* (QoS)-enabled middleware, such as the OMG *Data Distribution Service* (DDS) [6], to change mechanisms (such as the transport protocol used to deliver data) since some mechanisms provide better reliability than others in certain environments. Likewise, applications leveraging cloud computing environments where elastically allocated resources (*e.g.*, CPU speeds and memory) cannot be characterized accurately *a priori* may need to adjust to available resources (such as using compression algorithms optimized for the available CPU power and memory) at system startup. If adjustments take too long the mission(s) the system implements could be jeopardized.

One way to autonomically adapt enterprise DRE pub/sub systems involves *policy-based approaches* [7] [1] [8] that externalize and codify logic to manage the behavior of the systems. Policy-based approaches provide deterministic response times to guide appropriate adjustments given changes in the environment and can be optimized to ensure low-latency performance. The complexity of developing and maintaining policy-based approaches for enterprise DRE systems can be unacceptably high, however, since developers must determine and implement the policies which are applicable for certain environmental configurations. Moreover, developers must manage how the policies interact to provide needed adjustments.

Machine learning techniques support algorithms that allow systems to adjust behavior based on empirical data, *e.g.*, inputs from the environment. These techniques can be used to support autonomic adaptation by learning appropriate adjustments to various operating environments. Unlike policy-based approaches, however, machine learning techniques automatically recognize complex sets of environment properties, provide highly accurate support for environment properties not previously known or encountered, and make appropriate decisions accordingly.

Conventional machine learning techniques, such as decision trees [9] and reinforcement learning [2] , have been used to address autonomic adaptation for non-DRE systems [10].

[1] An autonomic system operates by managing itself without external intervention [36].

These techniques are not well-suited for enterprise DRE pub/sub systems, however, since they do not provide bounded times when determining adjustments [11]. Some techniques, such as reinforcement learning [12], explore the solution space until an appropriate solution is found, regardless of the elapsed time. Other techniques, such as decision trees, have time complexities that are dependent upon the specific data and cannot be determined *a priori*. Moreover, decision trees may contain decision branches that are much longer than others, thereby making the determination of appropriate adaptations unpredictable, which is undesirable for DRE pub/sub systems.

Supervised machine learning techniques with bounded times provide a promising way to addressing the accuracy, timeliness, and development complexity of DRE pub/sub systems. Some techniques, however, provide higher accuracy for environment configurations known *a priori*, whereas other techniques provide higher accuracy for environment configurations unknown until runtime. Since known and unknown environment configurations are relevant in dynamic operating environments, both types of techniques should be leveraged when they are most appropriate.

1.2 Solution approach → Integrate multiple machine learning techniques to guide QoS mechanism adaptation for QoS-enabled pub/sub middleware in dynamic environments.

In general, machine learning uses guidance from past known environments to handle new and unknown environments. This generality sacrifices some accuracy, however, that would otherwise be provided for known environments. Machine learning techniques that are specialized for the environments they have seen—and on which they have been trained—are said to be overfitted [13], which makes the accuracy comparable to policy-based approaches (*i.e.*, 100% accurate). Overfitted techniques are particularly suited for environments known *a priori* where the technique can be specialized for the known cases. Generalized machine learning techniques, however, are more appropriate for handling environments unknown until runtime.

This paper describes how we integrate pub/sub middleware with both overfitted machine learning and generalized machine learning to (1) reduce the complexity of autonomic adaptive enterprise DRE pub/sub systems, (2) provide the constant time complexity required of DRE pub/sub systems, and (3) increase the adaptation accuracy over any one single machine learning technique. In particular, our approach tunes an *artificial neural network* (ANN) [3] (which is a technique modeled on the interaction of neurons in the human brain) to retain as much information about environment configurations and adjustments known *a priori* as possible (*e.g.*, greatly increasing the number of connections between input environment characteristics and output adjustments typically used in an ANN). Our approach also tunes a *support vector machine* (SVM) to provide highly accurate adaptations for environments unknown until runtime.

This article expands on previous work [14] that outlined key benefits of overfitting neural networks by (1) including additional experimental data and analysis for ANNs, (2) empirically evaluating SVMs, and (3) integrating both ANNs and SVMs to increase overall adaptation accuracy. Our *ADAptive Middleware And Network Transports* (ADAMANT) technology

presented in this article utilizes the strengths of these two machine learning techniques and combines them with the DDS QoS-enabled pub/sub middleware to ensure accurate, timely, and predictable adaptation to both operating environment changes known *a priori* and those unknown until runtime, such as an increase in the data sending rate or number of data receivers.

The remainder of this paper is organized as follows: Section 2 describes a representative search and rescue application to motivate the challenges that ADAMANT addresses; Section 3 explains how ADAMANT addresses the challenges in Section 2; Section 4 presents empirical results of using ANNs and SVMs to determine appropriate adaptations and analysis of when to use each technique; Section 5 compares our work with related work; and Section 6 presents concluding remarks.

2 Motivating Example – Search and Rescue (SAR) Operations for Disaster Recovery

To motivate the need for integrating pub/sub middleware with overfitted and generalized machine learning techniques to provide high accuracy for both environments known *a priori* and those unknown until runtime, we motivate our work in the context of *search and rescue* (SAR) operations. SAR operations are part of disaster recovery enterprise DRE pub/sub systems that manage relief efforts in the aftermath of a disaster, such as a hurricane, earthquake, or tornado. SAR operations help locate and extract survivors in a large metropolitan area after a regional catastrophe. SAR operations increasingly use unmanned aerial vehicles (UAVs), existing operational monitoring infrastructure (*e.g.*, building or traffic light mounted cameras intended for security or traffic monitoring), and (temporary) datacenters to receive, process, and transmit event stream data from sensors and monitors to emergency vehicles that can be dispatched to areas where survivors are identified. In particular, our work focuses on configuring the QoS-enabled pub/sub middleware used by the temporary *ad hoc* datacenter for data dissemination.

Fig. 1 shows an example SAR scenario where infrared scans along with GPS coordinates are provided by UAVs and video feeds are provided by existing infrastructure cameras. These infrared scans and video feeds are sent to and disseminated by the *ad hoc* datacenter, where they are processed by fusion applications to detect and locate survivors. Once a survivor is detected the application can use the video and infrared feeds to develop a three dimensional view and highly accurate position information so that rescue operations can commence.

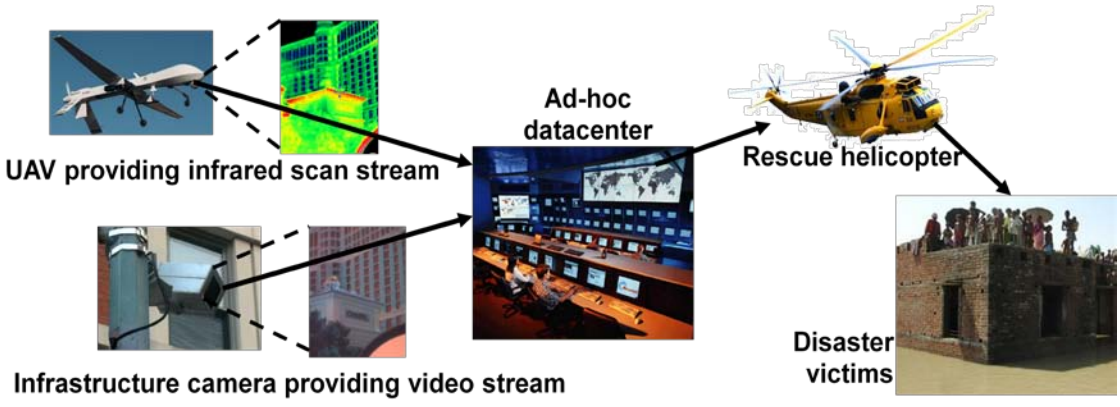

Figure 1. Motivating Search and Rescue (SAR) Example

A key requirement of data fusion applications within the datacenter is the tight timing bounds on correlated event streams such as the infrared scans coming from UAVs and video coming from cameras mounted atop traffic lights. The event streams need to match up closely so the survivor detection application can produce accurate results. If an infrared data stream is out of sync with a video data stream the survivor detection application can generate a false negative and fail to initiate needed rescue operations.

Likewise, without timely data coordination the survivor detection software can generate a false positive, thereby unnecessarily expending scarce resources, such as rescue workers, rescue vehicles, and data center coordinators. The timeliness and reliability properties of the data are affected by the transport protocols utilized by the pub/sub middleware to disseminate the data. For example, some protocols provide better reliability and lower latency for certain operating environments (*e.g.*, a low number of receivers and a low sending rate) whereas other protocols perform better for other operating environments (*e.g.*, high number of receivers and a high sending rate) [15].

2.1 Key Challenges of Enterprise DRE Systems in Dynamic Environments

Below we summarize key challenges that arise when developing autonomic enterprise DRE systems, such as the SAR motivating example described above.

2.1.1 Challenge 1: Reduction of Development Complexity

Developing autonomic behavior can incur high complexity due to the number and type of relevant environmental conditions. For example, the number of data receivers for SAR operations can affect the optimal transport protocols and parameter settings used since some protocols provide adequate QoS for a small number of receivers whereas other protocols provide adequate QoS for a larger number of receivers. Codifying this knowledge requires developers of SAR applications to manually (1) manage the appropriate protocols for given environments and (2) map these associations accurately into implementation artifacts (*e.g.*, source code). The manual management of mapping between environment and protocol is tedious and error-prone, which increases development complexity and reduces system trustworthiness. ADAMANT addresses this challenge with adaptation techniques that automatically manage the associations between the operating environments for SAR operations and the appropriate transport protocols, thus helping to manage development complexity, as described in Section 3.3.1.

2.1.2 Challenge 2: Timely Adaptation to Dynamic Environments

Due to the dynamic environment inherent in enterprise DRE systems, SAR operations (such as disseminating video and infrared data with both timeliness and reliability properties) must adjust in a bounded timely—ideally constant time—manner as the environment changes. SAR operations that cannot adjust quickly and in a bounded amount of time will fail to perform adequately when resources change, *e.g.*, if resources are lost or withdrawn—or demand for information increases—operations must be configured to accommodate these changes

with appropriate responsiveness to support the timeliness and reliability properties. If resources increase or demand decreases, SAR operations should adjust as quickly as possible to provide higher fidelity via enhanced timeliness and/or reliability of video or infrared data. Manual modification is often too slow and error-prone to maintain QoS. ADAMANT addresses this challenge via machine learning based on a fixed number of equations and that exhibit low latency and constant-time complexity, as described in Section 3.3.2.

2.1.3 Challenge 3: Accurate Adaptation to Environments Known *A Priori*

Application operations in enterprise DRE systems such as SAR operations must be able to adjust accurately to environment configurations known *a priori*. When changes in the environment of SAR operations occur, which have been previously encountered (*e.g.*, used to train the adaptation approach) the system should be able to select the transport protocol that can provide the ideal QoS for that environment with respect to timeliness and reliability. If SAR operations fail to provide the appropriate adjustment for an environment configuration then the SAR operations could be jeopardized. ADAMANT addresses this challenge by using machine learning techniques specifically fine-tuned for the data on which they are trained, thus providing 100% accuracy for environments known *a priori*, as described in Section 3.3.3.

2.1.4 Challenge 4: Accurate Adaptation to Previously Unknown Environments

SAR operations must be flexible enough to provide highly accurate adaptation guidance even for environment configurations that have not been previously encountered (*e.g.*, increase in network loss as wireless networks may need to be used due to damage of wired networks from the catastrophe, increase of receivers for video and infrared data as additional local, state, and federal agencies are wanting to monitor the situation). Due to the dynamic and turbulent nature of SAR operations, environment changes unknown until runtime can occur. As these unknown changes occur, the system must provide as much adaptation guidance as possible. Moreover, the guidance must provide greater than 50% accuracy. Otherwise, the guidance does more harm than good. ADAMANT addresses this challenge by using machine learning techniques designed to generalize from their training contexts, thus providing high accuracy for environments that are unknown until runtime, as described in Section 3.3.4.

3 Integrating Pub/Sub Middleware with Multiple Supervised Machine Learning Techniques to Guide Transport Protocol Adaptation

Our solution combines the benefits of (1) overfitted machine learning to increase accuracy in determining appropriate adjustments for environments known *a priori* and (2) generalized machine learning to provide guidance for environments unknown until runtime. These technologies along with DDS are detailed in Section 3.1. This approach enables enterprise DRE pub/sub systems to adjust to their environments autonomically, whether the properties of the environment are known *a priori* or not. Moreover, we leverage techniques that provide the time complexity assurance needed for enterprise DRE pub/sub systems. In particular, we are using this approach within the context of our *ADAptive Middleware And Network Transports* (ADAMANT) software platform to make adjustments to transport protocols to support

the QoS of pub/sub systems in dynamic environments.

3.1 Overview of Supporting Technologies

ADAMANT incorporates the use of the supervised machine learning techniques of ANNs and SVMs to provide robust guidance for transport protocol selection in dynamic environments. Supervised machine learning involves training a machine learning technique on data that is known to be correct. The supervised machine learning is then able to generalize this learning to provide guidance for data on which it has not been trained. Modeled on the interaction of neurons in the human brain [3], an ANN is a supervised machine learning technique that provides high accuracy to environments known *a priori* and robustness to environments unknown until runtime. Fig. 2 illustrates the architecture of an ANN with an input layer for relevant aspects of the operating environment, *e.g.*, percent network loss, sending rate and an output layer that represents the transport protocol that is generated based on the input of the operating environment.

Connecting the input and output layers is a hidden layer. As the ANN is trained on inputs and correspondingly correct outputs, it strengthens or weakens connections between the layers to generalize its learning based on the inputs and outputs. Nodes in the hidden layer (aka hidden nodes) are the computational components that provide connections between the relevant properties of the operating environment (*e.g.*, CPU speed, network reliability) with the adjustments needed for those environments. In general, an increase in the number of hidden nodes allows for an increase in learning that the ANN is able to perform. The stopping error is a specified value used during training the ANN that indicates that the ANN should keep iterating over the data until the error between the response that the ANN generates and the correct response is the specified value. In general, as the stopping error decreases the accuracy of the responses increases.

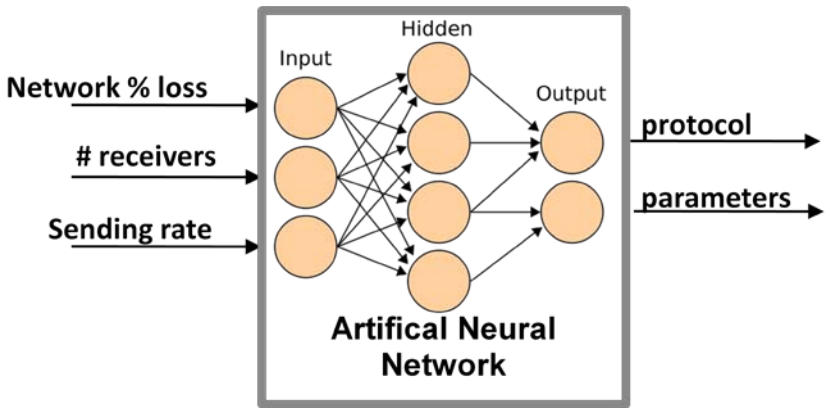

Figure 2. Artificial Neural Network to Determine Transport Protocol

Moreover, Fig. 2 outlines how an ANN is configured statically in the number of hidden layers and the number of nodes in each layer. This static configuration of an ANN directly affects the processing time complexity between the input of operating environment conditions and the output of an appropriate transport protocol and settings. Therefore, ANNs are able to support bounded response times.

Supervised machine learning uses guidance from past known environments to handle new and unknown environments. This generality sacrifices some accuracy, however, that would otherwise be provided for known environments. Overfitting the machine learning techniques specifically focuses learning on the environments that are known *a priori*. Parameters for the machine learning can be adjusted to provide high accuracy only for these particular environments (*e.g.*, increasing the number of nodes in the hidden layer or decreasing the stopping error in an ANN). Over-fitting reduces development complexity and makes the accuracy comparable to policy-based approaches.

SVMs are supervised learning methods used for classification and prediction. Given a set of training examples where each example is denoted as belonging to a particular class or grouping, an SVM builds a model that predicts into which grouping a new example should be categorized. The SVM generates the boundaries between the different groupings to maximize the differences between the groupings. This maximization helps to correctly classify new examples that have not been used in training the SVM model using the heuristic of locality, *i.e.*, examples that belong in the same group should be fairly close to each other in the classification space.

Fig. 3 illustrates conceptually how an SVM makes its determination for classification boundaries. The examples in grouping A are represented by solid circles while the examples in grouping B are represented by hollow circles. For simplicity and clarity, the examples are classified using two attributes. The dashed line C1 and the solid line C2 represent two different classifiers. An SVM produces a classifier similar to C2 since its margin between the two classification groupings is larger than with C1. A new example (*i.e.*, a solid grey circle) that needs to be classified using the SVM belongs in classification grouping A. The example is close to the line C1 but on the opposite side of the rest of the examples for that grouping and is therefore incorrectly classified whereas with C2 the new example would be correctly classified. An SVM maximizes the margin of differences between classification groupings.

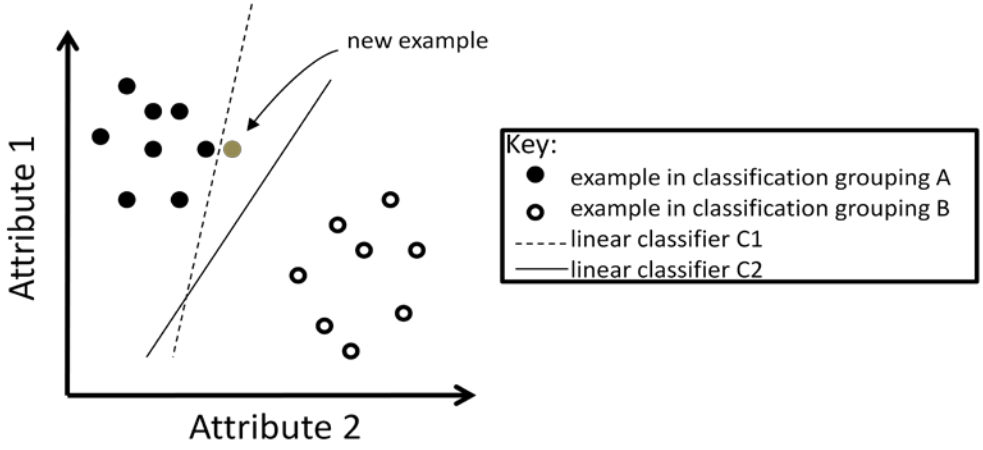

Figure 3. Support Vector Machine Classification Example

DDS specifies standards-based anonymous QoS-enabled pub/sub middleware for exchanging data in event-based distributed systems. DDS provides a global data store in which publishers and subscribers write and read data, respectively. DDS provides flexibility and

modular structure by decoupling: (1) redundancy, by allowing any numbers of readers and writers, (2) location, via anonymous publish/subscribe, (3) platform, by supporting a platform-independent model that can be mapped to different platform-specific models, such as C++ running on VxWorks or Java running on Real-time Linux, and (4) time, by providing asynchronous, time-independent data distribution.

The DDS architecture consists of two layers: (1) the data-centric pub/sub (DCPS) layer that provides APIs to exchange topic data based on specified QoS policies and (2) the data local reconstruction layer (DLRL) that makes topic data appear local. This paper focuses on DCPS since it is more broadly supported than the DLRL. The DCPS entities in DDS include *Topics*, which describe the type of data to be written or read; *Data Readers*, which subscribe to the values or instances of particular topics; and *Data Writers*, which publish values or instances for particular topics. Various properties of these entities can be configured using combinations of the 22 QoS policies specified by DDS. Moreover, *Publishers* manage groups of data writers and *Subscribers* manage groups of data readers.

3.2 Overview of ADAMANT

The ADAMANT platform integrates and enhances the DDS QoS-enabled pub/sub middleware, autonomic adaptation of the transport protocols used by DDS, the *Adaptive Network Transports* (ANT) framework (which supports multiple transport protocols), and supervised machine learning techniques to select appropriate transport protocols in a timely and reliable manner whether for environments known or unknown before runtime. ADAMANT focuses on adapting the middleware and protocols in dynamic environments, thereby enhancing prior work [16] that focused on accurate start-time configuration of QoS-enabled middleware for elastically provisioned resources in cloud computing environments. Fig. 4 shows the architecture and control flow for ADAMANT, which provides feedback on multiple aspects of the operating environment (*e.g.*, the number of receivers, data sending rate, percent network loss) through the use of a monitoring topic in DDS. The controller monitors the feedback and sends it on to the transport protocol optimizer. The optimizer determines if the environment configuration has been encountered previously and uses the appropriate machine learning technique that will provide the highest accuracy.

The optimizer utilizes a combination of an ANN and an SVM to determine the transport protocol that ADAMANT should use to disseminate data for the given operating environment. As detailed in Section 4, ANNs provide perfect accuracy for environment configurations whose properties are known prior to runtime while providing bounded, sub-μsec response times. SVMs provide higher accuracy for environments whose properties are not known until runtime with bounded μsec latencies. The optimizer determines whether to use an ANN or SVM for protocol determination.

The selected transport protocol is then sent to the adapter controller. The controller checks to see if the protocol recommendation differs from the current protocol used and, if so, passes the new protocol information to ANT, which updates ADAMANT to use the protocol for data dissemination. Fig. 4 illustrates a logically centralized architecture for the autonomic adaptation controller and the protocol optimizer. However, these components are physically

distributed across the computing platforms in the system, *i.e.*, each computing platform has its own local autonomic adaptation controller and protocol optimizer. Since the environment configuration changes are published to all the subscribers all the local controllers and optimizers receive the same updates. Since the controllers and optimizers are deterministic, all the controllers and optimizers will generate the same transport protocol to use. This physically distributed architecture allows for scalability in the number of publishers, subscribers, and computing platforms.

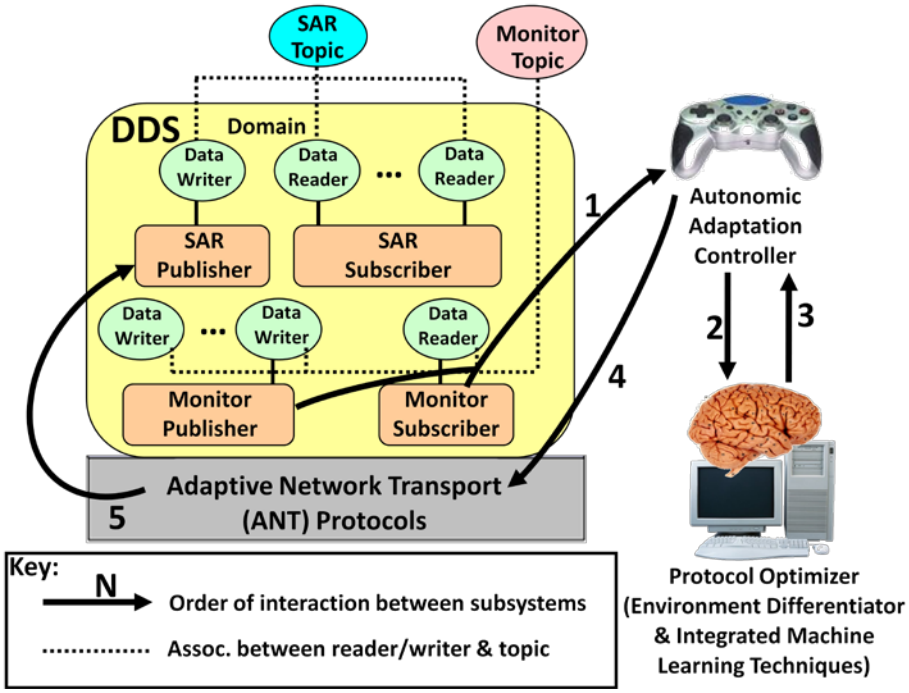

Figure 4. ADAMANT Architecture and Control Flow

The ANT framework supports various transport protocol properties, including multicast, packet tracking, NAK-based reliability, ACK-based reliability, flow control, group membership, and membership fault detection. These properties can be configured dynamically to achieve greater flexibility and support configuration adaptation. ANT also supports a negative acknowledgment (NAK) based multicast protocol (NAKcast) that supports a timeout parameter to determine when a data receiver should send NAKs to the data sender. The timeout parameter affects the timeliness of the transmitted data.

The ANT framework was derived from the Ricochet [17] transport protocol, which uses a bi-modal multicast protocol and a novel type of forward error correction (FEC) called lateral error correction (LEC) to provide QoS and scalability properties. Ricochet supports (1) time-critical multicast for high data rates with strong probabilistic delivery guarantees and (2) low-latency error detection along with low-latency error recovery. The Ricochet protocol has two tunable parameters. The *R* parameter determines the number of packets a receiver should receive before it sends out a repair packet to other receivers. The *C* parameter determines the number of receivers that will be sent a repair packet from any single receiver. Both parameters together affect the timeliness and reliability of the transmitted data.

With ANT's flexible support of transport protocols, the protocols that ADAMANT uses can be varied depending on the QoS properties that are needed. The Ricochet and NAKcast transport protocols have been the focus of the protocols in ANT due to their support for balancing the important QoS data dissemination properties of reliability and low latency [15]. Both of these properties are important for enterprise DRE pub/sub systems, such as SAR operations, which publish and subscribe to data with timeliness concerns so that the data is not stale as well as reliably delivering enough data to be useful.

3.3 Addressing the Key Challenges of Enterprise DRE Systems in Dynamic Environments

The ADAMANT approach to integrating multiple machine learning techniques combines (1) over-fitting an ANN to retain a high degree of information about environment configurations known *a priori*, (2) training an SVM to provide generalized adaptation guidance for environment configurations unknown until runtime, and (3) constant time complexity determination of whether an environment configuration is known or unknown at runtime so that the most accurate machine learning technique can be used. We over-fit the ANN by increasing the number of hidden nodes and decreasing the stopping error used for training the ANN.

As the ANN learns, it strengthens or weakens the connections between inputs, hidden nodes, and outputs to provide appropriate adjustments. The stopping error is an indication to the ANN that it should keep iterating over the data until the error between what the ANN generates and the correct response is sufficiently low. Increasing the number of hidden nodes and decreasing the stopping error increases the level of specificity that the ANN maintains. ADAMANT resolves the challenges presented in Section 2.1 as follows described below.

3.3.1 Solution 1: Machine Learning to Manage Development Complexity

The supervised machine learning techniques of ANNs and SVMs address Challenge 1 from Section 2.1.1 by decreasing the development complexity involved with codifying adjustments for multiple configurations of operating environments. Policy-based approaches for autonomic adaptation place the complexity burden on application developers, who must manually maintain operating environment configurations, appropriate adjustments needed, and the mapping between the configurations and the adjustments. Moreover, developers must accurately codify this mapping in their implementations. ANNs and SVMs relieve developers of this burden since they manage the complexity via training to react appropriately, *i.e.*, determining the appropriate transport protocol and parameter settings as an operating environment changes.

3.3.2 Solution 2: Constant-time Complexity Machine Learning

Machine learning techniques that utilize a static number of equations for learning address Challenge 2 from Section 2.1.2 by providing predictable time complexities for determining appropriate adjustments. In particular, ADAMANT applies SVMs and over-fitted ANNs to QoS-enabled pub/sub middleware to support enterprise DRE systems by incorporating the appropriate transport protocol adjustments according to feedback provided while the technique is trained. When an ANN or SVM is used to determine the appropriate transport protocol in an enterprise DRE system, the time to determine an appropriate adjustment is bounded

by the constant number of equations involved.

3.3.3 Solution 3: Over-fitted Machine Learning for Perfect Accuracy in Environments Known *A Priori*

Over-fitting a machine learning technique addresses Challenge 3 from Section 2.1.3 by increasing the adaptation accuracy for environments known *a priori*. ADAMANT increases the accuracy of ANNs for determining appropriate adjustments for known operating environments by (1) decreasing the error value by which the training for the ANN stops and (2) increasing the number of hidden nodes that connect the operating environment properties, such as CPU speed and network bandwidth, with the appropriate adjustments, such as transport protocols to support QoS. Specifically, over-fitting an ANN provides accuracy equal to policy-based approaches.

3.3.4 Solution 4: Generalized Machine Learning for High Accuracy in Environments Known at Runtime

ADAMANT's use of a generalized machine learning technique addresses Challenge 4 from Section 2.1.4 by increasing the adaptation accuracy for environments unknown until runtime. While over-fitted ANNs provide high accuracy for environments known *a priori*, the over-fitting reduces the accuracy for environments unknown until runtime. ADAMANT uses SVMs for determining appropriate adjustments for operating environments unknown until runtime. SVMs are designed to provide classification that maximizes the differences between data that are in different categories, *e.g.*, an environment where the Ricochet transport protocol provides the best QoS vs. an environment where NAKcast provides the best QoS. This maximization of differences increases the probability that an unknown environment will be classified correctly if it is similar to other environments that have been classified correctly during the training of the SVM.

The differentiation between known and unknown environments also helps to address Challenge 3 in Section 2.1.3 and Challenge 4 in Section 2.1.4 by determining which machine learning technique should be used for a given operating environment. This determination increases the overall adaptation accuracy since over-fitted ANNs provide better accuracy for environments known *a priori* than SVMs while SVMs provide better accuracy than over-fitted ANNs for environments unknown until runtime. The distinction between environments known before and after runtime allows us to leverage the most accurate machine learning technique.

ADAMANT leverages perfect hashing [18] for the environment configurations to determine in constant time whether or not an environment configuration is known or unknown. Perfect hashing utilizes the environment configurations, on which the ANN has been trained, as keys to map to the corresponding scaled environment configuration data, which is used by the ANN. If the key is mapped in the perfect hash then ADAMANT knows to use the ANN since it will provide perfect accuracy. If the key is not mapped then ADAMANT knows to use the SVM since it provides the highest accuracy for environment configurations, which have not been used for training. Perfect hashing provides the constant time complexity

needed for DRE systems.

4 Experimental Results and Analysis

This section presents the results of experiments that use ANNs and SVMs to determine development complexity, timeliness, and accuracy in selecting an appropriate transport protocol for ADAMANT given a particular operating environment. We conducted experiments for both environments known *a priori* (*i.e.*, using the same data for training and testing of the machine learning technique) and environments unknown until runtime (*i.e.*, using mutually exclusive data for training and testing). The experimental input data used to train the ANNs and SVMs include ADAMANT with multiple properties of the operating environment varied (*e.g.*, CPU speed, network bandwidth, DDS implementation, percent data loss in the network), along with multiple properties of the application being varied (*e.g.*, number of receivers, sending rate of the data) as would be expected for an *ad hoc* datacenter used for SAR operations.

From previous experiments that empirically evaluate how transport protocols perform in different operating environments [15] we gathered 394 inputs where an input consists of data values for features that determine a particular operating environment (*e.g.*, CPU speed, network bandwidth, number of data receivers, sending rate). Table 1 delineates the inputs for both of the machine learning techniques that ADAMANT utilizes (*i.e.*, ANNs and SVMs). The composite QoS metrics combine multiple QoS concerns into a single value. ReLate2 combines reliability and average network packet latency while ReLate2Jit extends the ReLate2 metric to also include standard deviation of packet arrival times (*i.e.*, packet jitter).

We also provided the expected output to the machine learning techniques, *i.e.*, the transport protocol that provided the best QoS with respect to data reliability, average latency, and jitter (*i.e.*, standard deviation of the latency of network packets) depending on which combination of these QoS properties were required. Both machine learning techniques used output a transport protocol and protocol settings given the operating environment inputs. An example of one of the 394 inputs is the following: 15 data receivers, 5% network loss, 100Hz data sending rate, 3GHz CPU, 1Gb network, using the OpenSplice DDS implementation, and specifying the combination of reliability and average latency as the QoS properties of interest. Based on our experiments, the corresponding output would be the Ricochet transport protocol with the *R* and *C* parameters set to 4 and 6, respectively.

Machine Learning Inputs	**Values**
Number of data receivers	3 - 25
Network bandwidth	1 Gb/s, 100 Mb/s, 10 Mb/s
DDS implementation	OpenDDS, OpenSplice
Percent end-host network loss	3 – 10 %

CPU type	850 MHz, 3 GHz
Data sending rate	10 Hz, 25 Hz, 50 Hz, 100 Hz
Available RAM	500 MB, 2 GB
Composite QoS metric	ReLate2, ReLate2Jit

Table 1. Machine Learning Inputs

4.1 Evaluating the Development Complexity of Policy-based Approaches

Policy-based approaches support a fairly straightforward way to determine optimal transport protocols for a given operating environment. After certain operating conditions are checked and met, these approaches guide the system to adapt its behavior. Fig. 5 shows an example where the application checks for the following environment properties applicable to the experimental data we collected relevant to SAR operations:

1. percentage loss in the network (*i.e.*, network_loss_percent),
2. number of data receivers (*i.e.*, num_receivers),
3. the rate of publishing data (*i.e.*, sending_rate),
4. the CPU processing speed (*i.e.*, CPU_speed),
5. the random-access memory available (*i.e.*, RAM),
6. the network bandwidth provided (*i.e.*, net_bw),
7. the DDS implementation used (*i.e.*, DDS_impl), and
8. the QoS properties of interest (*i.e.*, metric).

```
if (network_loss_percent == 1 && num_receivers < 5
    && sending_rate <= 25Hz && CPU_speed == 3GHz
    && RAM == 2GB && net_bw == 1Gb
    && DDS_impl == OpenSplice
    && metric == reliability_and_avg_latency) {
  transport_framework->use (transport1);
} else if (network_loss_percent == 3
           && num_receivers >= 5 && num_receivers < 10
           && sending_rate > 50Hz && CPU_speed == 850MHz
           && RAM == 500MB && net_bw == 100Mb
           && DDS_impl == OpenDDS
           && metric == reliability_and_jitter) {
  transport_framework->use (transport2);
} else if ...
```

Figure 5. Policy-based Adaptation Example

The time complexity of some policy-based approaches (*e.g.*, the manual implementation outlined in Fig. 5) can be optimized since (1) the bounded number of conditions that are checked and (2) the bounded number of the behaviors used to adapt the system is explicitly identified. For example, Fig. 5 shows that nested if statements or a switch statement in a pro-

gramming language can be used to implement policy-based approaches. In general, policy-based approaches can provide bounded times in searching for an adaptation solution and therefore address the boundedness evaluation criterion of Challenge 2 in Section 2.1.2 for adaptation approaches (*e.g.*, switch statements can be optimized for constant-time performance). Policy-based approaches also are highly accurate for environments known *a priori* since developers can codify the exact behavior needed for a known environment, thereby addressing Challenge 3 in Section 2.1.3.

Accidental complexity is exacerbated, however, when the conditions and responses for policy-based approaches are managed manually. Of the 8 environment properties shown in Fig. 5, 6 properties can take an infinite range of potential values, which cause an infinite number of combinations to be checked. Manually managing 8 properties for a configuration increases the complexity since humans are typically only able to efficiently process 7 pieces of information at a time [19]. Moreover, if the policies need to be modified the chance of introducing an error increases with the number of properties considered along with the number of ranges of values for each property.

Even policy-based approaches that alleviate manual management of conditions and responses by helping to automate these interactions, such as the Policy-Based Approach to Architectural Adaptation Management (PBAAM) [20] and the Spontaneous Object Framework (SOF) [21], still do not address the inherent complexity of determining appropriate policies and the accidental complexity of transforming these policies into implementation artifacts. Policy-based approaches therefore do not address the accidental development complexity criterion for adaptation approaches, whereas machine learning techniques manage this complexity automatically as part of their learning thus decreasing development complexity.

4.2 Evaluating the Accuracy of ANNs and SVMs

We next empirically evaluated the accuracy of ANNs and SVMs for both environments known *a priori* and unknown until runtime. We ran experiments with various numbers of hidden nodes and stopping errors. We used the *Fast Artificial Neural Network* (FANN) library (leenissen.dk/fann) as our ANN implementation due to its configurability, documentation, ease of use, and open-source availability. FANN offers extensive configurability for the neural network including the number of hidden nodes that connect the inputs with the output. We used the libSVM library (www.csie.ntu.edu.tw/~cjlin/libsvm) for our SVM implementation due to its configurability, documentation, tutorials, and open-source availability.

4.2.1 Evaluating the Accuracy of ANNs and SVMs for Environments Known *A Priori*

Below we evaluate the accuracy of ANNs and SVMs for environments known *a priori*. We first focus on ANNs and then SVMs. Our first step to measuring the accuracy of ANNs for environments known *a priori* was to train ANNs on the 394 inputs described in Section 4. We ran training experiments with the ANNs using different numbers of hidden nodes to determine the most accurate ANN. For a given number of hidden nodes we trained the ANN 10 different times. The weights of the ANNs determine how strong connections are between nodes. The weights are randomly initialized and these initial values have an effect on how

well and how quickly the ANN learns.

Fig. 6 shows the accuracies for the ANNs configured with 6, 12, 24, and 36 hidden nodes and a 0.0001 stopping error over 10 training runs. Additional experiments were conducted with higher stopping errors (*e.g.*, 0.01) but lower stopping errors consistently produced more accurate classifications as expected. Fig. 6 also shows the effect of random initial weights on the accuracy of the ANN since the accuracy can vary across training runs. Accuracy was determined by querying the ANN with the 394 inputs on which it was trained.

A 100% accurate classification was generated at least once with all ANN configurations. The ANN with 24 hidden nodes provided the best accuracy (as measured by the number of 100% accurate classifications) across all the training runs even compared to using 36 hidden nodes—100% accuracy all but 2 times out of 10. The ANNs with 36 and 12 hidden nodes both provided 100% accuracy 7 out of 10 times.

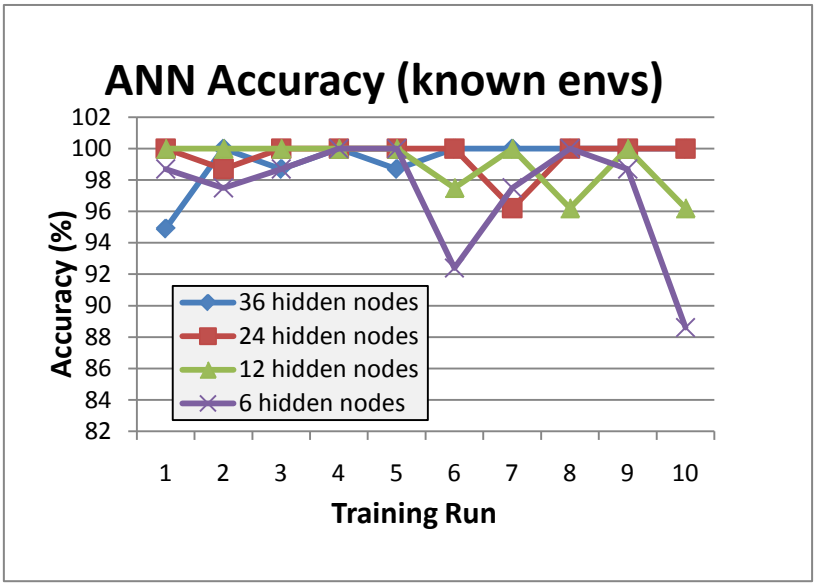

Figure 6. Accuracy of ANN with 6, 12, 24, & 36 Hidden Nodes & 0.0001 Stopping Error

The amount of error between values generated by the ANN and the known correct values is another measure of accuracy. The transport protocol output produced by the ANN is considered accurate if it is closer numerically to the correct protocol than to any other protocol. The ANN produces numerical values that are compared to the numerical representation of the correct protocol. The differences between these values can be accumulated across the 394 inputs and compared between different ANN configurations (*e.g.*, 36 hidden nodes, 24 hidden nodes).

Fig. 7 shows the errors accumulated for all the data of a single training & test run (*i.e.*, using the 394 inputs). The training runs here are only included if the ANN had 100% accuracy (as shown by Fig. 6) in that run since we want 100% accuracy for environments known *a priori*. For example, the ANN configured with 6 hidden nodes had only 3 training runs where the ANN was accurate for 100% of the 394 inputs and thus only 3 data points exist for that ANN configuration.

Across all the runs where the ANNs produced 100% accuracy, the ANN with 36 hidden nodes produced the lowest average error (*i.e.*, 8.99). The ANN with 24 hidden nodes, however, produced the second lowest average error (*i.e.*, 9.18) with only a 2.1% increase over the lowest error value. In this regard there is not a significant difference in the cumulative errors between the ANNs configured with 36 and 24 hidden nodes. By contrast, the 3rd lowest average error (*i.e.*, 10.65 using 12 hidden nodes) represents a 19% increase.

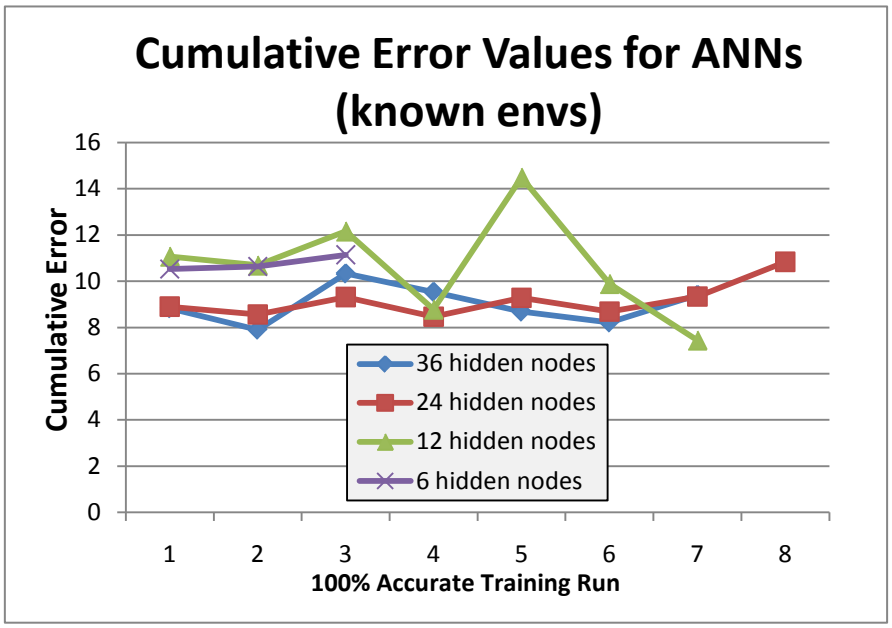

Figure 7. Cumulative Errors for ANNs with 100% Accuracy

As with the ANNs, we trained the SVMs on the 394 inputs to compare accuracy for environments known *a priori*. SVMs are a specialized kind of linear classifier that utilize kernels, which create additional features from the inputs that can be used for training and classification [22]. A variety of kernels can be used, such as radial basis function (RBF) and polynomial kernels, which produce different levels of accuracy.

We ran training experiments with the SVMs using different kernels and scaling the data in different ways. In addition to RBF and polynomial kernels, we included a simple linear classifier as a baseline to compare to the SVMs. The libSVM library provides the flexibility to utilize the SVMs with RBF and polynomial kernels as well as a simple linear classifier.

In addition, we scaled the data in 4 different ways: (1) no scaling (*i.e.*, using the original environment configuration values), (2) scaling the input values (*i.e.*, environment configurations) to be between -1 and 1, (3) scaling the input to be between -1 and 1 and scaling the output (*i.e.*, the transport protocol specified) to be between 0 and 1, and (4) scaling both input and output to be between -1 and 1. Since there is no non-determinism with training SVMs there is no need to have multiple training runs for a single SVM configuration (*e.g.*, SVM with polynomial kernels and no scaling of data).

Fig. 8 shows the accuracies for the linear classifier and the SVMs using the RBF and polynomial kernels. The SVM with the RBF kernel (SVM-RBF) was able to correctly predict

the appropriate transport protocol for 100% of the environment configurations. This result is somewhat surprising since SVMs are designed to generalize their learning and to handle environment configurations unknown until runtime. The SVM using the polynomial kernel (SVM-Polynomial) and the linear classifier produced their highest accuracy when the input data was scaled between -1 and 1 (92.39% and 91.17% accuracy respectively). Fig. 8 also shows the affect that data scaling has on accuracy. SVM-RBF had its highest accuracy (100%) when the data was not scaled while the linear classifier and SVM-Polynomial had their highest accuracy when only the input data was scaled. SVM-Polynomial was not able to complete training when the data was not scaled and therefore has no accuracy for that case. SVM-RBF, SVM-Polynomial, and the linear classifier all had their lowest accuracies when the output data (*i.e.*, the selected transport protocol) was scaled.

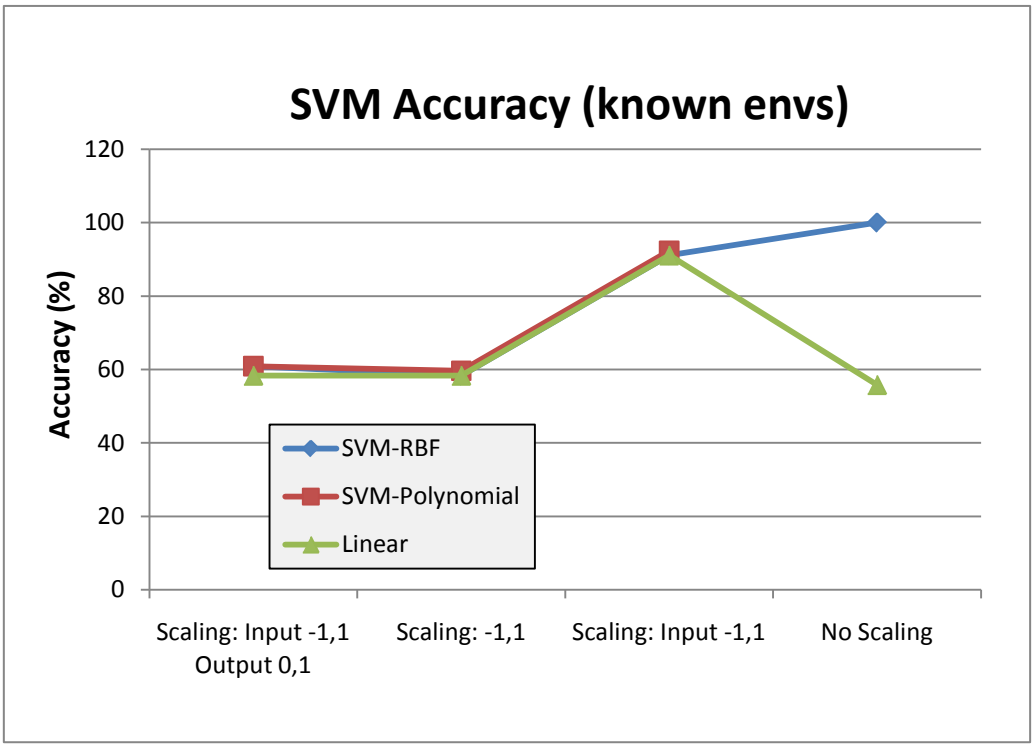

Figure 8. Accuracy of Linear Classifier and SVMs with RBF and Polynomial Kernels

Figs. 6, 7, and 8 show that ANNs with 24 and 36 hidden nodes and an SVM with RBF kernels and unscaled data produce the highest accuracies when the environment configurations are known *a priori*.

4.2.2 Evaluating the Accuracy of ANNs and SVMs for Environments Unknown until Runtime

Below we measure the accuracy of ANNs and SVMs for environment configurations that are unknown until runtime. This scenario splits out the 394 environment configurations into mutually exclusive training and testing data sets, which is referred to as *n*-fold cross-validation where *n* is the number of mutually exclusive training and testing data sets [23]. The value of *n* also determines the amount of data excluded from training and used only for testing. A 10-fold cross-validation indicates 10 sets of training and testing data where for

each fold the training and testing data are mutually exclusive and the training data excludes 1/10 of the total data, which is used only for testing. *N*-fold cross-validation provides insight into how well a machine learning technique generalizes for data on which it has not been trained.

We started by examining the accuracy of ANNs for environments unknown until runtime using 10-fold cross-validation. Since we are focusing on generalizing the machine learning for environment configurations not previously encountered, we added ANN configurations to those listed in Section 4.2.1 to include a larger stopping error to see the effect on accuracy. Fig. 9 shows the accuracy for the excluded data across the 10-folds for 10 different training runs. The ANNs in this figure all use the stopping error of 0.0001. Fig. 10 shows the accuracy for the excluded data across the 10-folds using a stopping error of 0.01. We split out these data into 2 separate figures for clarity.

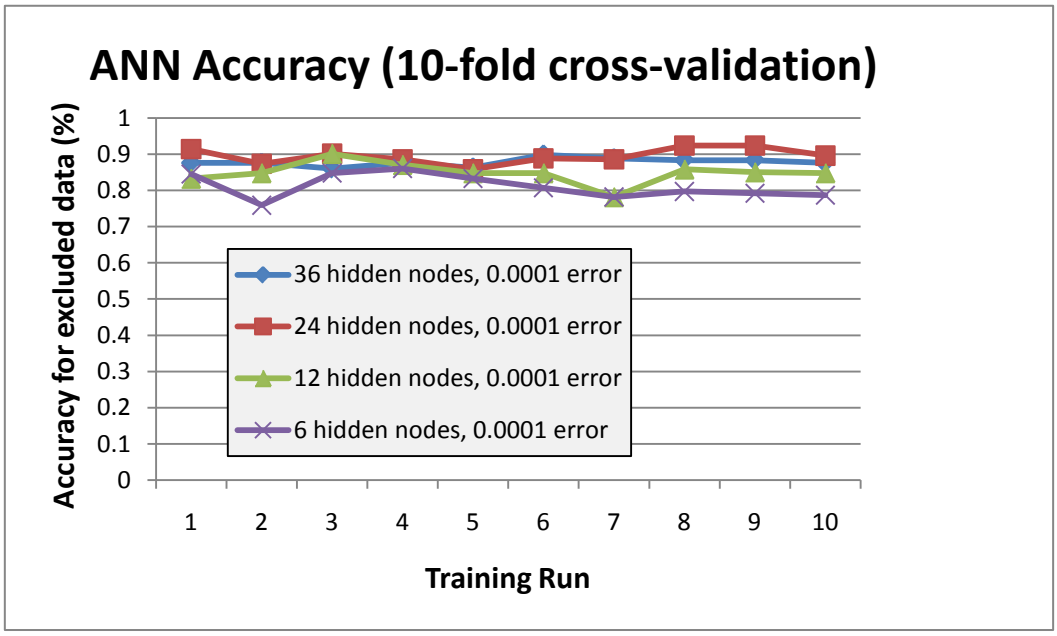

Figure 9. ANN Accuracies for 10-fold Cross-validation (0.0001 Stopping Error)

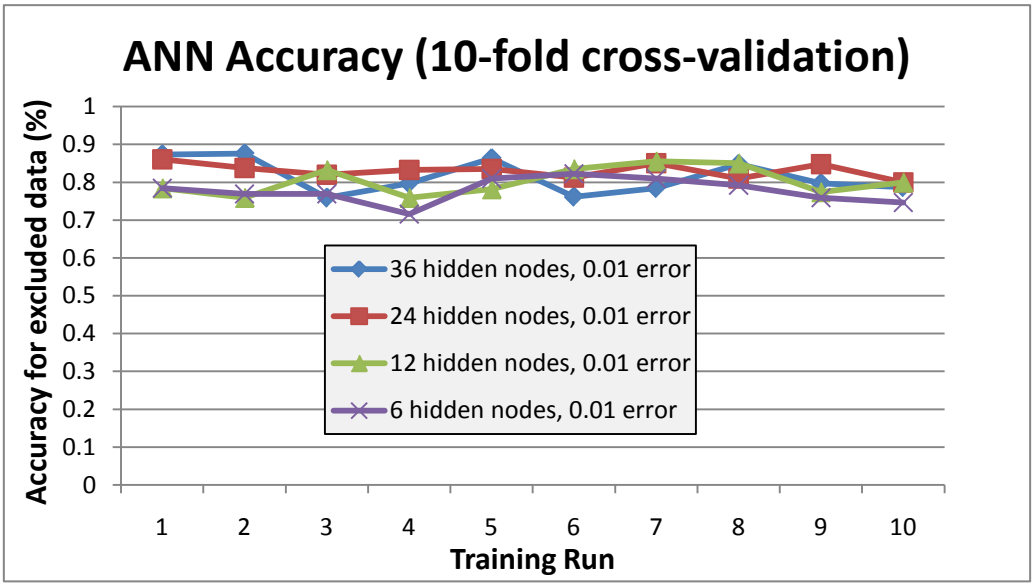

Figure 10. ANN Accuracies for 10-fold Cross-validation (0.01 Stopping Error)

As Figs. 9 and 10 show, the accuracy for determining the correct transport protocol given the unknown environment configurations was highest with the lower stopping error of 0.0001. The lowest accuracy for any run with a stopping error of 0.0001 was 75.89% (for 6 hidden nodes) while the lowest accuracy for any run with a stopping error of 0.01 was 71.57% (again for 6 hidden nodes). The highest accuracy was obtained by 24 hidden nodes and stopping error of 0.0001 for training run 9 (*i.e.*, 92.39%). This configuration also produced the highest average accuracy across all the training runs (*i.e.*, 89.49%) with the ANN configuration of 36 hidden nodes and a stopping error of 0.0001 producing the second highest average accuracy across all the training runs (*i.e.*, 87.79%).

To evaluate the linear classifier and SVMs, we used the same 10-fold cross-validation data described earlier in this section. Fig. 11 shows accuracies for the excluded data for the same linear classifier and the SVMs described in Section 4.2.1. As in Fig. 8, SVM-RBF again produces the highest accuracy for all the SVMs and the linear classifier (*i.e.*, 87.56%). This accuracy was produced, however, when the input data was scaled to values between -1 and 1. When no scaling of the data was performed, the accuracy of SVM-RBF decreased by 14.49% (*i.e.*, 74.87% accuracy).

SVM-RBF produced the highest accuracies across all data scalings compared to SVM-Polynomial and the linear classifier. As was the case with known environments, SVM-Polynomial was unable to complete training when the data was unscaled. When the input data was scaled between -1 and 1, SVM-RBF, SVM-Polynomial, and the linear classifier all produced their highest accuracies.

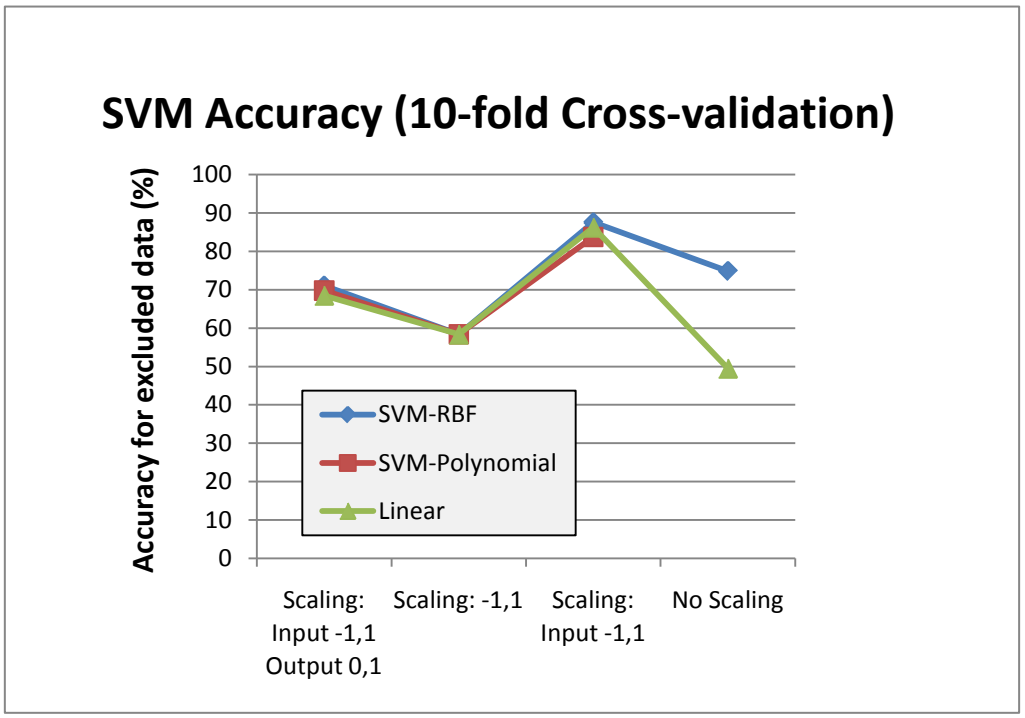

Figure 11. Linear Classifier and SVM Accuracies for 10-fold Cross-validation

With the current experiments involving environments unknown until runtime, the ANN with 24 hidden nodes and a stopping error of 0.0001 produced the highest accuracy. To see if these results would hold with an increased percentage of unknown data, we created 2-fold cross-validation data. For this set of data, half of the environment configurations would be used to train the ANNs and SVMs and the other half would be used to test the ANNs and SVMs for accuracy.

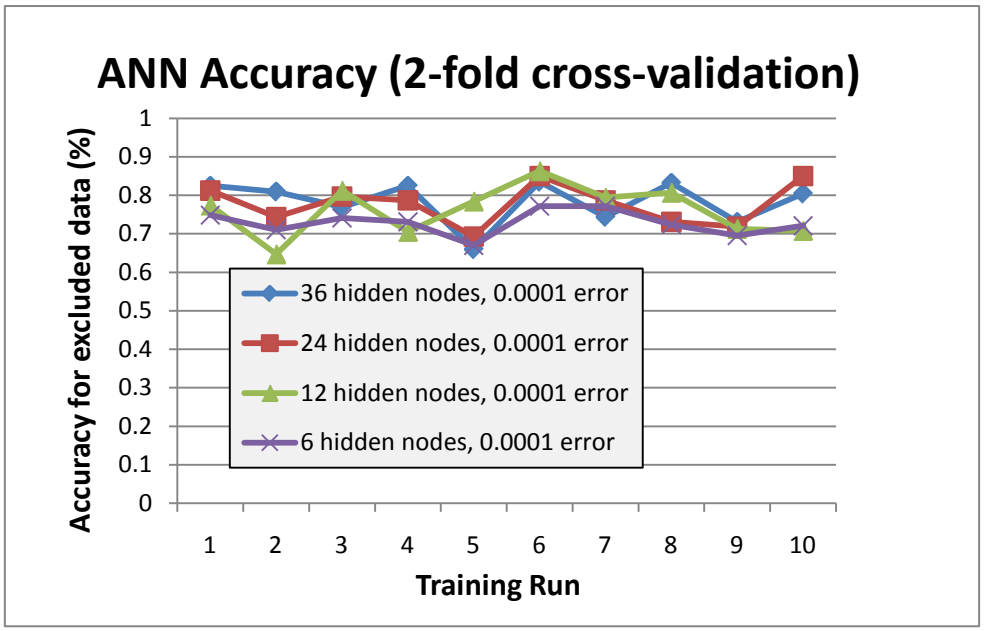

Figure 12. ANN Accuracies for 2-fold Cross-validation (0.0001 Stopping Error)

Figs. 12 and 13 show the accuracy for 2-fold cross-validation of ANNs configured with 6,

12, 24, and 36 nodes and 0.0001 and 0.01 stopping errors respectively. The highest average accuracy across the 10 trainings run is produced by the ANN with 36 hidden nodes and a stopping error of 0.0001 (*i.e.*, 78.35% average accuracy). The second highest average accuracy is now produced by the ANN with 24 hidden nodes and a stopping error of 0.0001 (*i.e.*, 77.69% average accuracy). It is also interesting to note that the ANN with 6 hidden nodes and a stopping error of 0.01 (*i.e.*, 74.19% average accuracy) produced higher average accuracies than with a stopping error of 0.0001. This result indicates that the ANN with 6 nodes generalizes its learning better with a higher stopping error.

Fig. 14 shows the accuracy of SVM-RBF, SVM-polynomial, and the linear classifier for 2-fold cross-validation. In this case, both SVM-RBF and SVM-Polynomial produce the highest accuracy (*i.e.*, 86.29%) when the input data is scaled from -1 to 1. Scaling the input data this way again produces the highest accuracies for SVM-RBF, SVM-Polynomial and the linear classifier as it did for 10-fold cross-validation. SVM-RBF also produced the worst 2-fold cross-validation accuracy when the input data was scaled from -1 to 1 and the output data was scaled from 0 to 1 (*i.e.*, 49.49% accuracy). Overall, SVM-RBF and SVM-Polynomial produced the most accurate results for 2-fold cross-validation even when including the accuracy results for the ANNs.

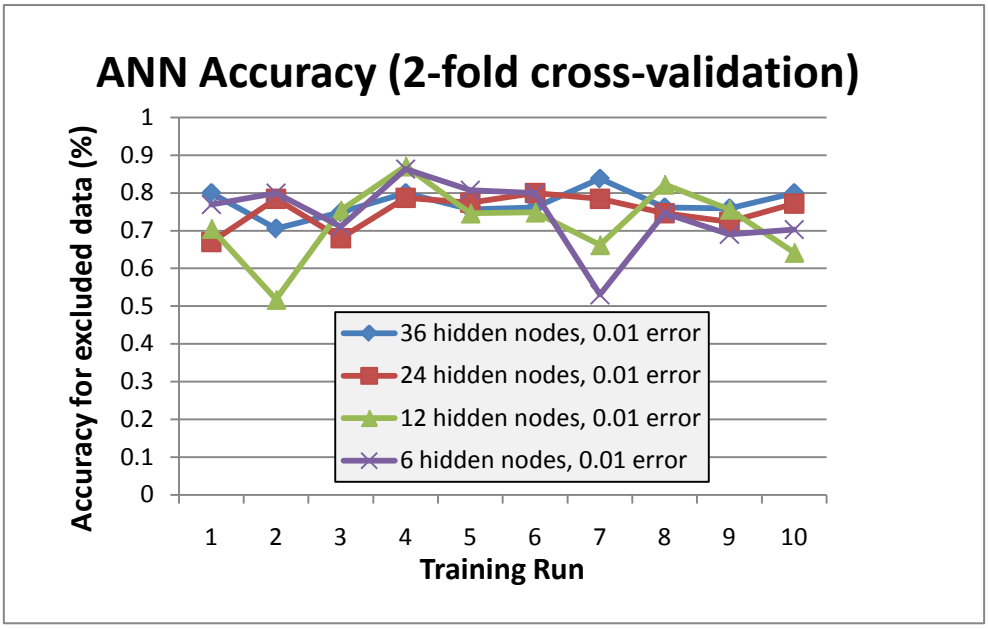

Figure 13. ANN Accuracies for 2-fold Cross-validation (0.01 Stopping Error)

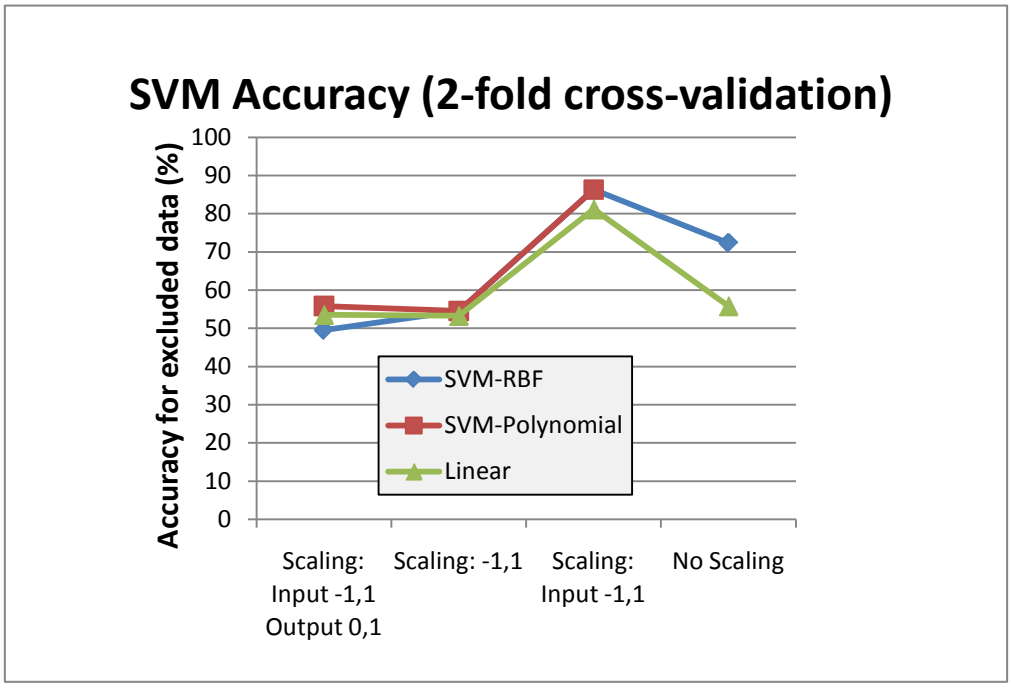

Figure 14. Linear Classifier and SVM Accuracies for 2-fold Cross-validation

4.3 Evaluating the Timeliness of ANNs and SVMs

For the SAR motivating example, we are not only interested in accurate transport protocol guidance. We also need low latency, constant time-complexity to meet the needs of DRE systems as outlined in Section 2.1.2. We now empirically evaluate the runtime timeliness of ANNs and SVMs.

To gather timing data, we used a 3 GHz CPU with 2GB of RAM running the Fedora Core 6 operating system with real-time extensions. Timeliness was determined by querying the ANNs and SVMs with all 394 inputs on which it was trained (*i.e.*, timing was done for the case of environments known *a priori*). A high resolution timestamp was taken before and after each call made to the ANN, the SVM, or the linear classifier and the times were calculated by subtracting the timestamp taken before the call from the timestamp taken after the call.

Figs. 15 and 16 show the average response times and standard deviation of the response times for ANNs, respectively, for 10 separate experiments where for each experiment we query the ANN for each of the 394 inputs. The figures show that the ANN provides timely and consistent responses. The standard deviations for all the ANNs and all the classification runs are below 1 μs. As expected, the response times using more hidden nodes are slower than response times with fewer hidden nodes. The increase in latency is less than linear, however, *e.g.*, response times using 12 hidden nodes are less than twice that using 6 hidden nodes.

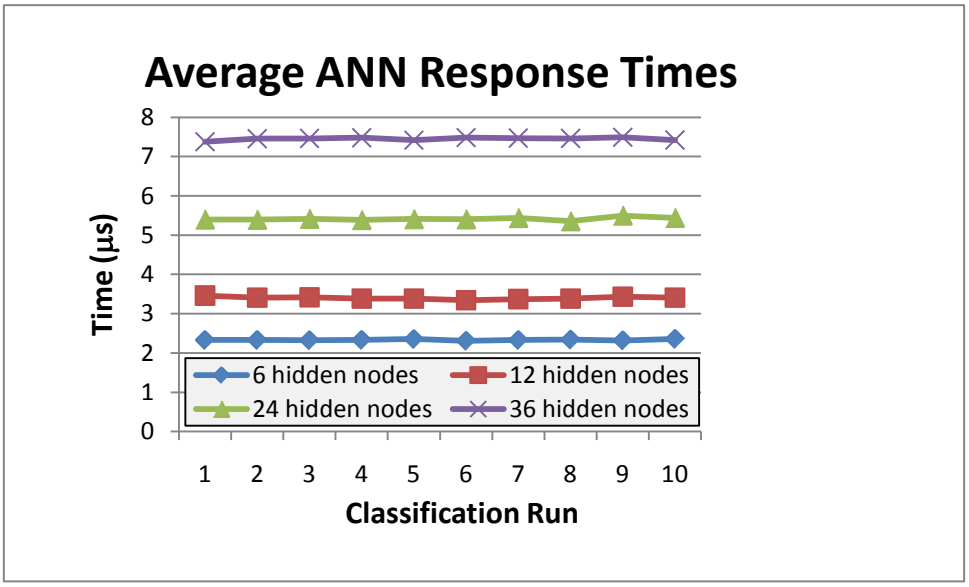

Figure 15. ANN Average Response Times (µsecs)

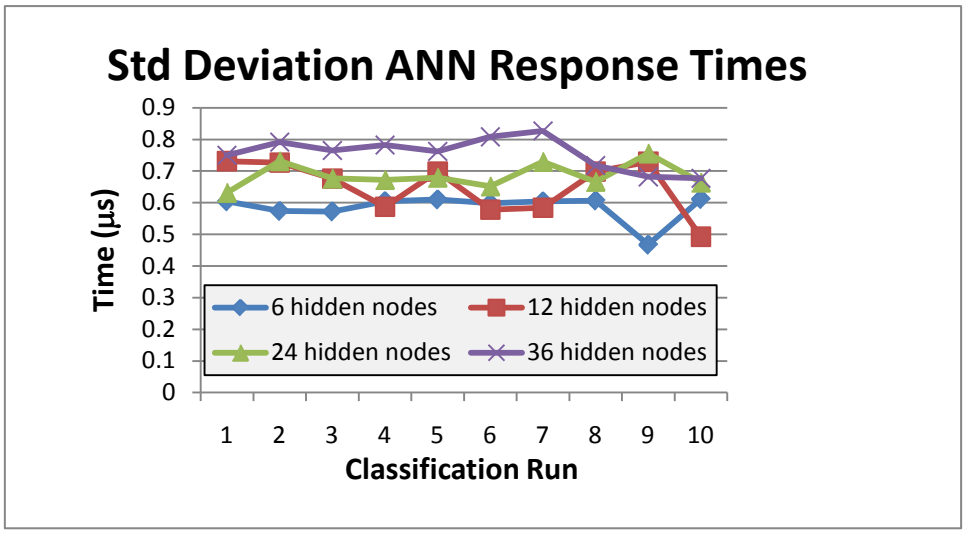

Figure 16. Standard Deviation for ANN Response Times (µsecs)

Figs. 17 and 18 show the average response times and standard deviations of the response times, respectively, for SVM-RBF, SVM-Polynomial, and the linear classifier where the data is scaled as described in Section 4.2.1. When trained on data that has been scaled differently, there is a difference in the average response time. The more data is scaled, the lower the average response time is (*e.g.*, scaling all the data values to be between -1 and 1 vs. no scaling). SVM-RBF has the highest average response times. This result is not surprising since the RBF kernels create more complicated calculations to determine the appropriate transport protocol. The standard deviations for the response times are fairly consistent regardless of scaling. SVM-RBF exhibits an unusually high standard deviation when the data is not scaled. This data might show an anomaly with the libSVM library with respect to timeliness since we also needed to modify the implementation to remove sources of unbounded time complexity for testing timeliness (*e.g.*, replace calls to `malloc()` with statically allocated memory since

`malloc()` provides no bounded time guarantees).

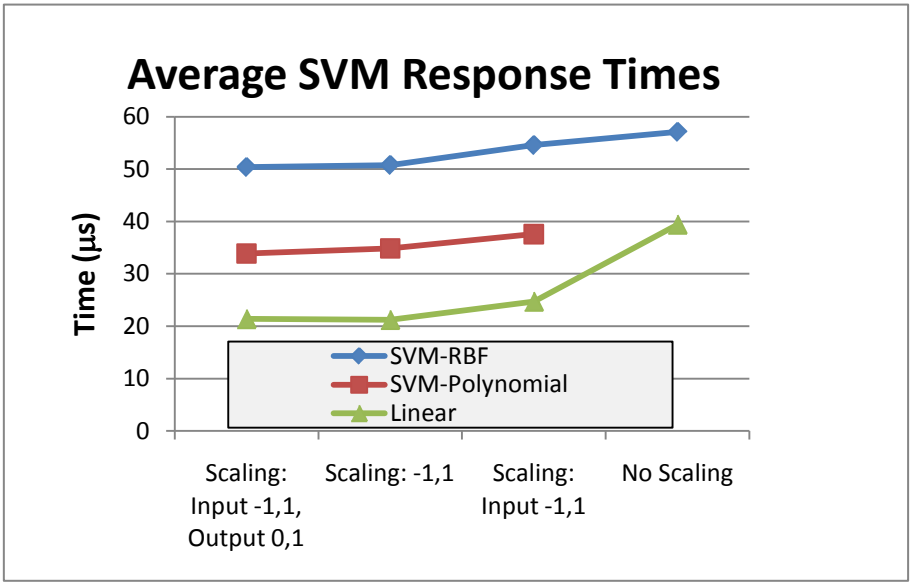

Figure 17. Linear Classifier and SVM Average Response Times (µsecs)

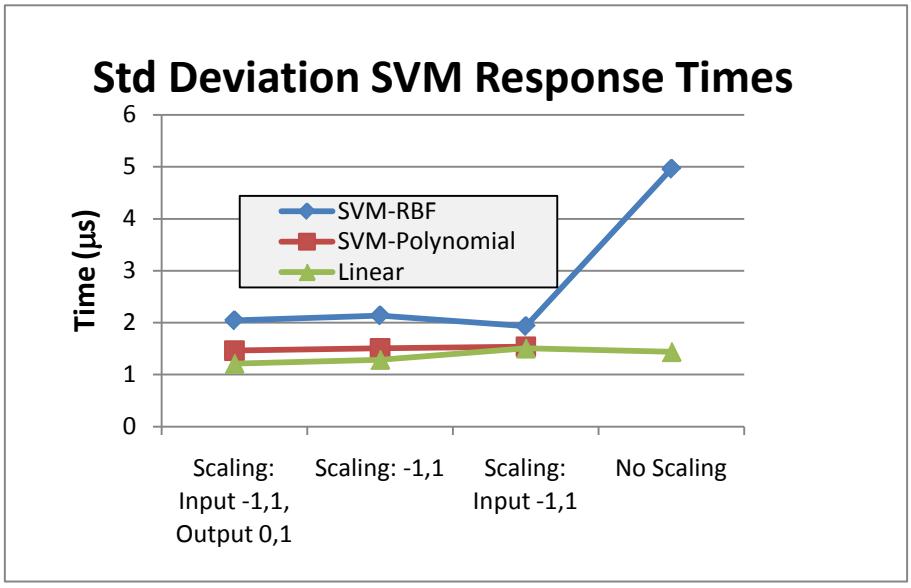

Figure 18. Standard Deviation for Linear Classifier and SVM Response Times (µsecs)

4.4 Analyzing the Experimental Results

The ANN with 24 hidden nodes and a stopping error of 0.0001 produced the most accurate result for experiments dealing with environments known *a priori*. The ANN with 36 hidden nodes was comparable in its accuracy for environments known *a priori*. When considering the timeliness advantage of 24 hidden nodes compared to 36 hidden nodes, however, the ANN with 24 hidden nodes seems the most appropriate for environments known *a priori*.

With the 10-fold cross-validation experiments, the ANN with 24 hidden nodes and a stopping error of 0.0001 produced the highest accuracy over the other ANN configurations, the linear classifier, and the SVMs. This result is somewhat surprising since SVMs are de-

signed to be more general and handle unknown input well whereas over-fitted ANNs are not expected to do as well. However, with the 2-fold cross-validation experiments, the ANN with 36 hidden nodes and a stopping error of 0.0001 produced more accurate results averaged over 10 training runs than the ANN with 24 hidden nodes and the same stopping error. These differing results indicate that if only ANNs are used for machine learning, we should use one type of ANN for environments known *a priori* (*i.e.*, 24 hidden nodes and stopping error of 0.0001) and one type for environments unknown until runtime (*i.e.*, 36 hidden nodes and a stopping error of 0.0001). We speculate that the excluded environment configurations for the 10-fold cross-validation was a small enough percentage of the overall data that it was not different enough to cause a different ANN configuration or an SVM to be more accurate. The results of the 2-fold cross-validation support this conclusion.

The experimental results shown in Sections 4.2 and 4.3 illustrate that different supervised machine learning techniques provide different levels of accuracy depending on whether the environment configuration is known a priori or unknown until runtime. Based on these experimental data, we have chosen to use an ANN with 24 hidden nodes and a stopping error of 0.0001 for environments known *a priori* due to its (1) perfect accuracy and (2) low latency constant classification time. We have also chosen to use an SVM with a polynomial kernel for environments unknown until runtime since this technique (1) provided the highest accuracy for all the configurations of ANNs and SVMs tested with the most generalized data for validation (*i.e.*, 2-fold cross validation data) and (2) provided lower response latencies than the SVM with the RBF kernel. As noted in Section 3.2, we combine both of these machine learning techniques to increase the overall accuracy of ADAMANT and leverage constant-time perfect hashing to determine if the environment configuration is known *a priori* or unknown until runtime and use the machine learning that will provide the highest accuracy.

5 Related Work

This section compares our work integrating multiple machine learning approaches for adapting QoS-enabled pub/sub middleware in ADAMANT with related work.

The use of multiple data driven methods in concert to construct a system for classification and detection is a concept that falls into the general description of the term *Meta-Learning* [24]. Meta-Learning has been used in machine learning for building scalable systems that can possess multiple learning algorithms and effectively utilize them all to improve classification. ADAMANT is designed in a similar method with using multiple machine learning techniques: an over-fitted ANN for environments known *a priori* and an SVM for environments unknown until runtime. The intelligent design and application of these Meta-Learning systems is imperative to improving classification accuracy. ADAMANT extends Meta-Learning by improving classification accuracy and addressing the timeliness requirements of a dynamic DRE system. The ADAMANT Meta-Learning approach of combining multiple machine learning techniques is similar to the work on combining classifiers as proposed by Kittler [25]. However, with Kittler's work there is no consideration for timeliness which is crucial for DRE systems. Also, ADAMANT uses a very coarse grained, constant-time complexity classification (*e.g.*, perfect hashing) to determine which machine

learning technique to use for classification whereas Kittler's approach fuses multiple classifiers via combining results into multiple stages without regard for the timeliness of the individual classifiers or the fusion of the classifiers.

Our use of an over-fitted ANN takes advantage of the back-propagation algorithm [26] used in training ANNs. Research in deep-belief networks [27] helps improve the accuracy of ANNs by allowing for more complex error functions and more complex architectures (*e.g.*, the number of layers for hidden nodes and their construction). The deep-belief networks provide the advantage of allowing a larger range of classification and thus can improve accuracy in certain areas. The time complexity of deep-belief networks, however, is not constant so it is not applicable for DRE systems.

The SVM algorithm commonly used for classification produces a classifier that separates the decision space of the environment configurations as equitably as possible [28]. A common technique to affect accuracy is to map the features of the data into another decision space, in hopes of finding a classifier that provides increased accuracy. This technique for producing non-linear classifiers is called a *kernel trick* [29]. Among the possibilities of kernel tricks is translating the data to a polynomial kernel, as performed in our work. Other kernels are available, such as the *Gaussian Kernel* [30], which can produce even more complex mappings. As the complexity increases, the data becomes increasingly altered to the point where mapping to the original data for human understanding becomes hard. We chose to only use RBF and polynomial kernels since the benefits of additional kernels were unclear and added complexity to the testing and analysis.

The *Dynamic Control of Behavior based on Learning* (DCBL) middleware [31] incorporates reinforcement machine learning in support of autonomic control for QoS management. Reinforcement machine learning allows DCBL to handle unexpected changes, as well as reduce the overall system knowledge required by the system developers. System developers are required to create an XML description of the system, which DCBL then uses together with an internal representation of the managed system to select appropriate QoS dynamically.

In contrast to ADAMANT, however, DCBL is applicable to a single computer, rather than scalable QoS-enabled pub/sub middleware for DRE systems. Moreover, reinforcement learning used by DCBL does not provide constant time or even bounded time complexities unlike our work, which provides fast, constant-time complexity decision making. DCBL also requires developers to specify in an XML file the selection of operating modes given a QoS level along with execution paths, which increases accidental development complexity. In contrast, ADAMANT ameliorates this complexity by having supervised machine learning techniques determine the appropriate operating modes for a given environment.

RAC [12] incorporates reinforcement learning to aid in the configuration of web services by autonomically configuring the services via performance parameter settings to change both the services' workload and the virtual machine configurations. The reinforcement learning component of RAC is enhanced with an additional runtime initialization period at system startup. As with DCBL, however, RAC's use of reinforcement learning causes unbounded response times due to online exploration of the solution space and modification of decisions

while the system is running. In contrast, ADAMANT leverages constant-time complexity supervised machine learning to provide predictable and fast decision making.

Bellavista *et al.* [32] have conducted research centering on their embedded middleware called Mobile agent-based Ubiquitous multimedia Middleware (MUM). The purpose of MUM is to handle the complexities of wireless hand-off management for wireless devices moving among different points of attachment to the Internet. In this sense, MUM adapts its functionality for an application as it moves through its environments. In contrast, our work on ADAMANT focuses on the needs of DRE pub/sub systems to adapt in a low latency, bounded time manner based on the QoS of the application and the resources presented in the environment.

Boonma *et al.* [33] have specialized a DDS implementation called TinyDDS for the demands of wireless sensor networks (WSNs). TinyDDS defines a subset of DDS interfaces for efficiency and simplicity for WSNs. Additionally, TinyDDS includes a pluggable framework for non-functional properties, *e.g.*, power-efficient routing capability, event correlation and filtering mechanisms, and data aggregation functionality. However, TinyDDS provides no guidance for adapting to a change in environment resources as is the case with ADAMANT.

Li and Nahrstedt [34] present the Middleware Control Framework (MCF) which enhances the effectiveness of QoS adaptation using runtime monitoring, control, and reconfiguration of the parameters of available QoS mechanisms (*e.g.*, network bandwidth usage, CPU usage for object tracking algorithms). The goal of MCF is to satisfy system-wide properties, such as fairness among concurrent applications, as well as application-specific requirements, such as maintaining the performance of services critical to the system. MCF uses a combination of control-theoretic and fuzzy logic to address the system-wide properties and the application-specific requirements respectively. However, unlike ADAMANT which focuses on low latency, constant-time complexity adaptation, MCF does not address the crucial area of bounded adaptation times for DRE systems.

The Dynamic TAO and Open ORB middleware projects [35] outline approaches taken to incorporate reflection into middleware. Reflective middleware uses a collection of components that can be configured and reconfigured by the application. System and application code inspect the internal configuration of the middleware and adapt the middleware in response to changes in the environment. ADAMANT utilizes some of the overarching concepts of reflective middleware in that ADAMANT monitors the QoS and reasons about its current state to make changes as needed. In addition, ADAMANT addresses the critical timeliness concerns of DRE pub/sub systems when making changes unlike the work with Dynamic TAO and Open ORB.

6 Concluding Remarks

This article describes how the *ADAptive Middleware And Network Transports* (ADAMANT) software platform utilizes the strengths of multiple supervised machine learning techniques to adapt the transport protocols of enterprise DRE systems autonomically in a dynamic environment to increase accuracy for environments known *a priori* and unknown until

runtime. The empirical results in this paper show how a combination of over-fitted artificial neural networks (ANNs) and support vector machines (SVMs) help address the development complexity, timeliness, and accuracy of adaptive enterprise DRE systems for both environments known *a prori* and only at runtime. The following is a summary of lessons learned from our experience evaluating machine learning techniques for adapting ADAMANT when executing in dynamic environments:

- Scaling data has an impact on accuracy. With our initial training of ANNs we were only able to achieve 92% accuracy for known environments regardless of the number of hidden nodes or the stopping error. When we scaled the data to be between the values of -1 and 1 we were able to achieve 100% accuracy for environments known *a priori* with several different ANN configurations. Scaling data also has an effect on SVMs and linear classifiers. Unscaled data provided the greatest accuracy for the SVM using RBF kernel for environments known *a priori*. For environments unknown until runtime, however, the greatest accuracy for SVM-RBF was when the input data was scaled between the values of -1 and 1.

- Determining between environments known *a priori* and unknown until runtime based only on ANN output values is challenging. We initially hypothesized that the numeric output of an over-fitted ANN would be able to indicate if a configuration environment was known *a priori* or unknown until runtime. Since the ANN was over-fitted to particular data and provided a low error relative to the expected numerical output for an environment known *a priori*, we assumed that an environment configuration unknown until runtime would produce a numerical output that would distinguish it as not previously known. We have not been able, however, to leverage the numerical output to make this distinction. We are continuing research in this area to eliminate the need for the additional mechanism to determine environments known *a priori* and only at runtime (*e.g.*, perfect hashing).

- More generalized machine learning techniques are more applicable to environments known at runtime (*e.g.*, SVMs do much better than ANNs for 2-fold cross validation). SVMs are particularly designed to generalize their learning and thus be accurate when encountering previously unknown environments. Our empirical results bore out this assumption when there was a large variance between the environments known *a priori* and only at runtime. If the environments unknown until runtime were a small enough subset of the environments known *a priori*, however, then over-fitted ANNs provided the best accuracy.

- ANNs might do well for environments unknown until runtime if these environments are few compared to the environments known *a priori*. Building on the previous lesson learned, an adaptive DRE system might be able to retrain an ANN during runtime using a low priority thread so that the environment configurations unknown until runtime quickly become part of the environments known *a priori* and any unknown configurations backlogged during retraining remain a small subset of the environments already included in training.

Our current work, software downloads, and additional information can be found at www.dre.vanderbilt.edu/~jhoffert/ADAMANT.

References

[1] Agrawal, D., Lee, Kang-Won, Lobo, J., "Policy-based Management of Networked Computing Systems". Communications. Vol. 43, Issue 10. Pp. 69-75. October 2005

[2] Busoniu, L., Babuska, R., De Schutter, B., Ernst, D, *Reinforcement Learning and Dynamic Programming Using Function Approximators*. Boca Raton: CRC Press, 2010.

[3] Patterson, D., *Artificial Neural Networks: Theory and Applications*. Upper Saddle River: Prentice Hall PTR, 1998.

[4] Meyer, D., Leisch, D., F. Hornik, K., "The Support Vector Machine Under Test". Neurocomputing. Vol. 55, Issue 1-2. Pp. 169 - 186. 2003

[5] Shankaran, N., Koutsoukos, X, Lu, C., Schmidt, D., Xue, Y., "Hierarchical Control of Multiple Resources in Distributed Real-time and Embedded Systems". Real-Time Systems. Vol. 39, Issue 1-3. Pp. 237 - 282. August 2008.

[6] Object Management Group. Data Distribution Service. January 2007.

[7] Choudhary, A., "Policy Based Management in the Global Information Grid". International Journal of Internet Protocol Technology. Vol. 3, Issue 1. Pp. 72 - 80. 2008

[8] Liu, S., Ding, Y., "An Adaptive Network Policy Management Framework Based on Classical Conditioning", The Seventh World Congress on Intelligent Control and Automation, (WCICA 2008). Chongqing, China, June 25-27, 2008

[9] Breiman, L., Freidman, J., Olshen, R., Stone, C., *Classification and Regression Trees*. Monterey: Wadsworth, 1984.

[10] Hess, A., Nussbaumer, M., Hlavacs, H., Hummel, K., "Automatic Adaptation and Analysis of SIP Headers Using Decision Trees", The Second International Conference on Principles, Systems and Applications of IP Telecommunications (IPTComm 2008). Heidelberg, Germany, July 1-2, 2008.

[11] Hoffert, J., Schmidt, D., Gokhale, A., "Adapting and Evaluating Distributed Real-time and Embedded Systems in Dynamic Environments", The First International Workshop on Data Dissemination for Large scale Complex Critical Infrastructures (DD4LCCI 2010). Valencia, Spain, April 28, 2010.

[12] Bu, X., Rao, J., Xu, C.-Z. "A Reinforcement Learning Approach to Online Web Systems Auto-configuration", The 29th IEEE International Conference on Distributed Computing Systems (ICDCS 2009). Montreal, Canada, June 22-26, 2009.

[13] Dietterich, T., "Overfitting and Undercomputing in Machine Learning". ACM Computing Surveys. Vol. 27, Issue 3. Pp 326-327. September, 1995.

[14] Hoffert, J., Schmidt, D., "Evaluating Supervised Machine Learning for Adapting Enterprise DRE Systems", The 2010 International Symposium on Intelligence Information Processing and Trusted Computing (IPTC 2010). Huanggang, China, October 28 – 29, 2010.

[15] Hoffert, J., Gokhale, A., Schmidt, D., "Evaluating Transport Protocols for Real-time Event Stream Processing Middleware and Applications", The 11th International Symposium on Distributed Objects, Middleware, and Applications (DOA '09). Vilamoura, Algarve-Portugal, November 2-3, 2009.

[16] Hoffert, J., Schmidt, D., "Adapting Distributed Real-time and Embedded Publish/Subscribe Middleware for Cloud-Computing Environments", *to appear in the Proceedings of the* ACM/IFIP/USENIX 11th International Middleware Conference (Middleware 2010). Bangalore, India, November 29 – December 3, 2010.
[17] Balakrishnan, M., Birman, K., Phanishayee, A., Pleisch, S., "Ricochet: Lateral Error Correction for Time-Critical Multicast", Fourth Usenix Symposium on Networked Systems Design and Implementation (NSDI 2007). Boston, USA, April 11 – 13, 2007.
[18] Brodnik, A., Munro, J., Ian. "Membership in Constant Time and Minimum Space", in *Algorithms - ESA '94*. Berlin / Heidelberg: Springer LNCS, 1994, Pp. 72-81.
[19] Miller, G., "The Magical Number Seven, Plus or Minus Two: Some Limits on Our Capacity for Processing Information". Psychological Review, Vol. 63. Pp 81-97. 1956.
[20] Georgas, J., Taylor, R., "Policy-Based Architectural Adaptation Management: Robotics Domain Case Studies", in Software Engineering for Self-Adaptive Systems. Berlin / Heidelberg: Springer-Verlag, 2009, Pp. 89-108.
[21] Szydlo, T., Szymacha, R., Zielinski, K., "Policy-based Context-aware Adaptable Software Components for Mobility Computing", The 10th IEEE International Enterprise Distributed Object Computing Conference (EDOC '06). Hong Kong, China, October 16 – 20 2006.
[22] Theodoridis, S., Koutroumbas, K., *Pattern Recognition*. Orlando: Academic Press, Inc., 2006.
[23] Liu, Y., "Create Stable Neural Networks by Cross-Validation", International Joint Conference on Neural Networks, 2006 (IJCNN '06). Vancouver, B.C., July 16-21, 2006.
[24] Chan, P., Stolfo, S., "On the Accuracy of Meta-learning for Scalable Data Mining". Journal of Intelligent Information Systems. Vol. 8, Issue 1. Pp 5 - 28. January/February1997.
[25] Kittler, J., "Combining Classifiers: A Theoretical Framework". Pattern Analysis & Applications. Vol. 1, Issue 1. Pp 18-27. March 1998.
[26] Bryson, A., Ho, Y.-C., "Applied optimal control: optimization, estimation, and control". IEEE Transactions on Systems, Man and Cybernetics. Vol. 9, Issue 6. Pp 366 – 367. June 1979.
[27] Bengio, Y., LeCun, Y., "Scaling Learning Algorithms towards AI", in *Large-Scale Kernel Machines*, Bottou, L., Chapelle, O., Decoste, D., Weston, J., Ed. Cambridge: MIT Press, 2007, pp. 321-359.
[28] Burges, C. "A Tutorial on Support Vector Machines for Pattern Recognition". Data Mining and Knowledge Discovery. Vol. 2, Issue 2. Pp 121 – 167. June 1998.
[29] Boser, B., Guyon, I. Vapnik, V., "A Training Algorithm for Optimal Margin Classifiers", The Fifth Annual ACM Workshop on Computational Learning Theory. Pittsburgh, USA, July 27-29, 1992.
[30] Keerthi, S., Lin, C.-J., "Asymptotic Behaviors of Support Vector Machines with Gaussian Kernel". Neural Computation. Vol. 15, Issue 7. Pp 1667 - 1689. July 2003.
[31] Vienne, P., Sourrouille, J.-L., "A Middleware for Autonomic QoS Management Based on Learning", The 5th International Workshop on Software Engineering And Middleware (SEM 2005). Lisbon, Portugal, September 5-6, 2005.
[32] Bellavista, P., Corradi, A., Foschini, L., "Context-aware Handoff Middleware for Transparent Service Continuity in Wireless Networks", Pervasive and Mobile Computing. Vol. 3,

Issue 4. Pp 439-466. 2007.
[33] Boonma, P., Suzuki, J., “Middleware Support for Pluggable Non-functional Properties in Wireless Sensor Networks”, IEEE Congress on Services - Part I. Washington, D.C., July 8-11, 2008.
[34] Li, B., Nahrstedt, K., “A Control-based Middleware Framework for QoS Adaptations”, IEEE Journal on Selected Areas in Communications. Vol. 17, Issue 9. Pp 1632-1650. 1999.
[35] Kon, F., Costa, F., Blair, G., Campbell, R., “The Case for Reflective Middleware”, Communications of the ACM. Vol. 45, Issue 6. Pp 33-38. 2002.
[36] Kephart, J., Chess, D., “The Vision of Autonomic Computing”, Computer. Vol. 36, Issue 1. Pp 41-50. January 2003.