# Using Machine Learning to Maintain Pub/Sub System QoS in Dynamic Environments*

Joe Hoffert, Daniel Mack, and Douglas Schmidt
Department of Electrical Engineering & Computer Science, Vanderbilt University, Nashville, TN
jhoffert@dre.vanderbilt.edu, dmack@isis.vanderbilt.edu, schmidt@dre.vanderbilt.edu

## ABSTRACT

Quality-of-service (QoS)-enabled publish/subscribe (pub/sub) middleware provides powerful support for scalable data dissemination. It is hard, however, to maintain specified QoS properties (such as reliability and latency) in dynamic environments (such as disaster relief operations or power grids). For example, managing QoS manually is often not feasible in dynamic systems due to (1) slow human response times, (2) the complexity of managing multiple interrelated QoS settings, and (3) the scale of the systems being managed. For certain applications the systems must be able to reflect on the conditions of their environment and adapt accordingly.

Machine learning techniques provide a promising adaptation approach to maintaining QoS properties of QoS-enabled pub/sub middleware in dynamic environments. These techniques include decision trees, neural networks, and linear logistic regression classifiers that can be trained on existing data to interpolate and extrapolate for new data. By training the machine learning techniques with system performance metrics in a wide variety of configurations, changes to middleware mechanisms (*e.g.*, associations of publishers and subscribers to transport protocols) can be driven by machine learning to maintain specified QoS.

This paper describes how we are applying machine learning techniques to simplify the configuration of QoS-enabled middleware and adaptive transport protocols to maintain specified QoS as systems change dynamically. The results of our work thus far show that decision trees and neural networks can effectively classify the best protocols to use. In particular, decision trees answer questions about which measurements and variables are most important when considering the reliability and latency of pub/sub systems.

## Categories and Subject Descriptors

C.2.4.b [**Distributed Systems**]: Distributed applications

## General Terms

Performance

*This work is supported in part by the AFRL/IF Pollux project and NSF TRUST.


*ARM 2009*, December 1, 2009, Urbana Champaign, Illinois, USA.


## Keywords

QoS, Pub/Sub, Machine learning, Dynamic environments

## 1. INTRODUCTION

**Emerging trends and challenges.** The number and type of distributed systems that utilize publish/subscribe (pub/sub) technologies have grown due to the advantages of performance, cost, and scale as compared to single computers [8, 10]. Examples of pub/sub middleware include Web Services Brokered Notification (`www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsn`), the Java Message Service (JMS) (`java.sun.com/products/jms`), the CORBA Event Service (`www.omg.org/technology/documents/formal/event_service.htm`), and the Data Distribution Service (DDS) (`www.omg.org/spec/DDS`). These technologies support data propagation throughout a system using an anonymous subscription model that decouples event suppliers and consumers.

Pub/sub middleware is used in a wide spectrum of application domains, ranging from shipboard computing environments to grid computing. The middleware supports policies that affect the end-to-end QoS of the system. Common policies across different middleware include *persistence* (*i.e.*, saving data for current subscribers), *durability* (*i.e.*, saving data for subsequent subscribers), and *grouped data transfer* (*i.e.*, transmitting a group of data atomically).

While tunable policies provide fine-grained control of system QoS, several challenges emerge when developing pub/sub systems deployed in dynamic environments. Middleware mechanisms used to ensure certain QoS properties for one environment configuration may not be applicable for another configuration. For example, a simple unicast protocol (such as UDP) may provide adequate latency QoS when a publisher sends to a small number of subscribers. UDP could incur too much latency, however, when used for a large number of subscribers due to the publisher needing to send to each individual subscriber.

Challenges also arise when managing multiple QoS policies that interact with each other. For example, a system might specify low latency QoS and reliability QoS, which can affect latency due to data loss discovery and recovery. Certain transport protocols (such as UDP) provide low overhead but no end-to-end reliability. Other protocols (such as TCP) provide reliability but unbounded latencies due to acknowledgment-based retransmissions. Still other protocols, (such as lateral error correction protocols [1]) balance reliability and low latency, but only provide benefits over other protocols for specific environment configurations.

Determining when to modify parameters of a particular transport protocol or switch from one transport protocol to another is hard. Moreover, human intervention is often not responsive enough for the timeliness requirements of the system. The problem of timely

response is exacerbated by increasing the scale of the system, *e.g.*, increasing the number of publishers or subscribers.

**Solution approach → ADAptive Middleware And Network Transports (ADAMANT).** ADAMANT is QoS-enabled pub/sub middleware that uses machine learning techniques to (1) adaptively configure transport protocols and (2) manage specified QoS by reflecting on a system's dynamically changing environment and adapting accordingly. By utilizing machine learning techniques trained on data collected from multiple configurations, ADAMANT adjusts middleware mechanisms to provide transport protocol and parameter settings that maintain the specified QoS. The remainder of this document describes how ADAMANT addresses key challenges of maintaining QoS for pub/sub systems in dynamic environments via machine learning. In particular, we evaluate several machine learning techniques for providing adaptation guidance.

## 2. MOTIVATING EXAMPLE

Search and rescue (SAR) operations help locate and extract survivors in a large metropolitan area after a regional catastrophe, such as a hurricane, earthquake, or tornado. Figure 1 shows an example SAR scenario where infrared scans along with GPS coordinates are provided by unmanned aerial vehicles (UAVs) and video feeds are provided by existing infrastructure cameras 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. These infrared scans and video feeds are then sent to a datacenter, where they are processed by fusion applications to detect survivors and develop three dimensional views and highly accurate position information for rescue operations.

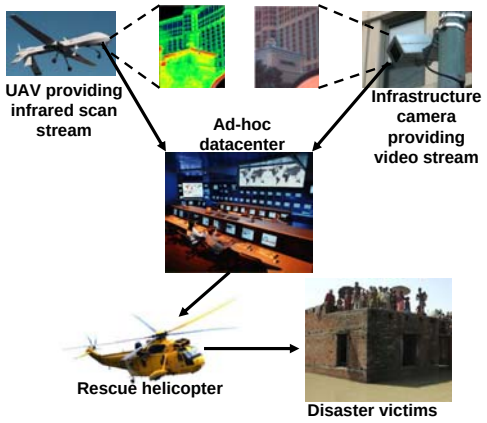


**Figure 1: Search and Rescue Motivating Example**

To motivate the need for integrating machine learning techniques with QoS-enabled pub/sub middleware, this section describes the following key research challenges associated with SAR operations in dynamic environments:

### 2.1 Challenge 1: Timely Adaptation to Dynamic Environments

Due to the dynamic environment inherent in the aftermath of a disaster, SAR operations must adjust in a timely manner as the environment changes. If SAR operations cannot adjust quickly enough they will fail to perform adequately given a shift in resources. If resources are lost or withdrawn—or demand for information increases—SAR operations must be configured to accommodate these changes with appropriate responsiveness to maintain a minimum level of service. If resources increase or demand decreases, SAR operations should adjust as quickly as possible to provide higher fidelity or more expansive coverage. Manual modification is often too slow and error-prone to maintain QoS.

### 2.2 Challenge 2: Managing Interacting QoS Requirements

SAR operations must manage multiple QoS requirements that interact with each other, *e.g.*, data reliability so that enough data is received to be useful and low latency for soft realtime data so that infrared scans from UAVs or video from cameras mounted atop traffic lights do not arrive after they are needed. The streamed data must be received soon enough so that successive dependent data can be used as well. For example, MPEG I frame data must be received in a timely manner so that successive dependent B and P frame data can be used before the next I frame makes them obsolete. Otherwise, not only is the data unnecessary, but sending and processing the data has consumed limited resources.

### 2.3 Challenge 3: Scaling to Large Numbers of Receivers

For a regional or national disaster, a multitude of organizations would register interest not only in the individual video and infrared scans for various applications, but also in the fused data for the SAR operations. For example, fire detection applications and power grid assessment applications can use infrared scans to detect fires and working HVAC systems respectively. Likewise, security monitoring and structural damage applications can use video stream data to detect looting and unsafe buildings respectively. Moreover, federal, state, and local authorities would want to register interest in the fused SAR data to monitor the status of current SAR operations.

### 2.4 Challenge 4: Specifying Standardized and Robust QoS

SAR applications should be developed with the focus on application logic rather than on complex or custom formats for specifying QoS. Time spent learning a customized or complex format for QoS is time taken from developing the SAR application itself. Moreover, learning a custom format will not be applicable for other applications that use a different QoS format. Application developers also need support for a wide range of QoS to handle dynamic environments.

## 3. OVERVIEW OF RELATED WORK

This section analyzes related research efforts in light of the challenges presented in Section 2.

**Support for adaptive middleware.** The Mobility Support Service (MSS) [3] provides a software layer on top of pub/sub middleware to enable endhost mobility. The purpose of MSS is to support the movement of clients between access points of a system using pub/sub middleware. In this sense, MSS adapts the pub/sub middleware used in a mobile environment. Mobile clients notify MSS when mobility starts and ends. MSS buffers messages and manages connections while the client moves to a different access point. MSS is designed to support multiple pub/sub technologies, *e.g.*, implementations of JMS, and adapt to the technology-specific characteristics.

MSS is solely focused on supporting mobility of pub/sub, however, and therefore does not address Challenge 2 in Section 2.2. Moreover, MSS fails to address Challenge 4 in Section 2.4 since it does not present a standardized and robust interface for QoS.

Gridkit [5] is a middleware framework that supports reconfigurability of applications dependent upon the condition of the environment and the functionality of registered components. Gridkit focuses on grid applications which are highly heterogeneous in nature. For example, these applications will run on many types of computing devices and across different types of networks.

To register components, application developers use Gridkit's API which is based on binding contracts. Gridkit then uses the contract information along with a context engine to determine which components to include in the application. The context engine takes into account the context of the host machines, *e.g.*, battery life, network connectivity.

Gridkit focuses on reconfiguration for installing an application and does not address Challenge 1 in Section 2.1. Within Gridkit no consideration is given to making timely adaptations based on the environment changing for a single application installation. Moreover, Gridkit fails to address Challenge 4 in Section 2.4 as it provides no standardized QoS specification.

David and Ledoux have developed SAFRAN [4] to enable applications to become context-aware themselves so that they can adapt to their contexts. SAFRAN provides reactive adaptation policy infrastructure for components using an aspect-oriented approach. SAFRAN follows the structure of a generic AOP system by supporting (1) a base program which corresponds to a configuration of components, (2) point-cuts which are invoked in response to internal events (*e.g.*, invocations on interfaces) and external events (*e.g.*, change in system resources), (3) advices which define functionality to be executed for point-cuts, and (4) adaptation which uses adaptation policies to link join points to advices.

The SAFRAN component framework, however, only provides development support of maintaining specified QoS. The adaptive policies and component implementation are the responsibility of the application developer. Moreover, SAFRAN does not specifically address Challenge 3 in Section 2.3 since it does not focus on scalability. SAFRAN also does not address Challenge 4 in Section 2.4 since it provides no standard QoS specification.

**Machine learning in support of autonomic adaptation.** Vienne and Sourrouille [12] present the Dynamic Control of Behavior based on Learning (DCBL) middleware that incorporates reinforcement machine learning in support of autonomic control for QoS management. Reinforcement machine learning not only allows DCBL to handle unexpected changes but also reduces the overall system knowledge required by the system developers. System developers provide an XML description of the system, which DCBL then uses together with an internal representation of the managed system to select appropriate QoS dynamically.

DCBL's customized QoS specification, however, does not address Challenge 4 in Section 2.4 and DCBL focuses on single computers rather than addressing scalable distributed systems, as outlined with Challenge 3 in Section 2.3. Moreover, DCBL requires developers to specify in an XML file the selection of operating modes given a QoS level along with execution paths, which leaves handling Challenge 2 in Section 2.2 to developers.

Tock et al [11] utilize machine learning for data dissemination in their work on Multicast Mapping (MCM). MCM hierarchically clusters data flows so that multiple topics are mapped onto a single session and multiple sessions are mapped onto a single reliable multicast group. MCM's approach manages the scarce availability of multicast addresses in large-scale systems. MCM leverages machine learning to adapt as user interest and message rate change during the day. MCM is just designed to address the scarce resource of IP multicast addresses in large-scale systems, however, rather than Challenge 2 in Section 2.2 or Challenge 4 in Section 2.4.

**Autonomic adaption of service level agreements.** Herssens *et al.* [6] describe work that centers around autonomically adapting service level agreements (SLAs) when the context of the specified service changes. This work acknowledges that both offered and the requested QoS for Web services might vary over the course of the interaction and accordingly modifies the SLA between the client and the server as appropriate. This work does not address Challenge 1 in Section 2.1, but rather negotiates the QoS agreement to fit the dynamic environment.

**Autonomic adaption of networks.** The Autonomic Real-time Multicast Distribution System (ARMDS) [2] is a framework that focuses on decreasing excessive variance in service quality for multicast data across the Internet. The framework supports the autonomic adaptation of the network nodes forming the multicast graph so that the consistency of service delivery is enhanced. ARMDS does not address Challenge 2 in Section 2.2, however, nor does it address Challenge 4 in Section 2.4.

# 4. ADAMANT OVERVIEW AND RESULTS

This section describes ADAMANT, our experimental setup, and the results for evaluating machine learning techniques in providing adaptation guidance to select the most appropriate protocol and configuration settings for a particular dynamic environment.

## 4.1 Overview of ADAMANT

The ADAMANT QoS-enabled pub/sub middleware uses machine learning techniques to adjust the underlying transport protocols and associated parameter settings to maintain specified end-to-end QoS. ADAMANT addresses the challenges presented in Section 2 to resolve gaps in related work described in Section 3 via the following integrated techniques.

- The **Adaptive Network Transports** (ANT) framework addresses Challenge 1 in Section 2.1 and Challenge 2 in Section 2.2 by providing the flexibility to maintain interrelated QoS even within dynamic environments. For some environment configurations one particular transport protocol provides the required QoS. For other environment configurations a different transport protocol provides the specified QoS. ANT not only supports fine grained control of a protocol's parameters, but also switching from one protocol to another to provide the adaptation needed within dynamic environments. Moreover, ANT works to address Challenge 3 in Section 2.3 by supplying appropriate transport protocols and protocol settings as the number of senders and receivers in the system fluctuate.

We chose ANT due to its infrastructure for composing transport protocols. ANT builds upon the properties provided by the scalable reliable multicast-based Ricochet transport protocol [1]. It also provides a modular framework whereby protocol modules can be tuned, enhanced, and replaced to maintain specified QoS.

- **Machine learning techniques** help address Challenge 1 in Section 2.1 and Challenge 2 in Section 2.2 by selecting in a timely manner an appropriate transport protocol and protocol parameters given specified QoS and a particular environment configuration. Machine learning can interpolate and extrapolate its learning based on the current environment configuration, which might not have been included originally. Thus, machine learning provides increased flexibility over manual or policy driven approaches.

The first learning technique investigated is a decision tree (DT). This algorithm attempts to create a tree where a set of decisions leads down to a leaf node that can accurately classify a new example. A DT will attempt to produce the shortest and smallest tree possible while maintaining accuracy by looking for features that best split the data as completely as possible and use them closer to the root. DTs are designed for data sets with more than a binary set of classes, *i.e.*, where there are more than two possible classifications of an appropriate transport protocol and parameter settings.

The second technique we investigated is an Artificial Neural Network (ANN). ANNs work well on sets with a small number of features and can produce highly accurate results with medium sized data sets. ANNs also can generally produce results in less time

than other machine learning techniques. The learning produced when using ANNs is not as accessible as a DT, however, since the factors that are used and the importance placed on each factor are difficult to present in a human understandable form.

The third technique we investigated is a Linear Logistic Regression Classifier (LLRC), which uses a weighting of the various collected metrics to determine the appropriate protocol and parameters. The results from LLRCs have increased comprehensibility as compared to ANNs since how the environment configuration influences the selection of transport protocol and parameter settings is less opaque. Moreover, LLRCs can be optimized to reduce the time to determine an optimal protocol and settings.

• **The OMG Data Distribution Service** (DDS) middleware (`www.omg.org/spec/DDS`) addresses the scalability of Challenge 3 in Section 2.3 by decoupling data senders from data receivers. DDS enables applications to communicate by publishing information they have and subscribing to information they need in a timely manner. DDS is standards-based anonymous QoS-enabled pub/sub middleware for exchanging data in event-based distributed systems. It provides a global data store in which publishers and subscribers write and read data, respectively.

DDS provides flexibility and modular structure by decoupling: (1) *location*, via anonymous publish/subscribe, (2) *redundancy*, by allowing any numbers of readers and writers, (3) *time*, by providing asynchronous, time-independent data distribution, and (4) *platform*, by supporting a platform-independent model that can be mapped to different platform-specific models.

The DDS architecture consists of two layers: (1) the *data-centric pub/sub* (DCPS) layer that provides APIs to exchange topic data based on chosen QoS policies and (2) the *data local reconstruction layer* (DLRL) that makes topic data appear local. Our work 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. Moreover, *Publishers* manage groups of data writers and *Subscribers* manage groups of data readers.

Additionally, utilizing DDS addresses the QoS standardization of Challenge 4 in Section 2.4. Table 1 summarizes the DDS QoS policies. DDS provides 22 QoS policies applicable to various entity types. Each QoS policy has $\sim$2 attributes with the majority of the attributes having a large number of possible values, *e.g.*, an attribute of type long or character string.

Figure 2 also shows how we integrated and enhanced the OpenDDS implementation (`www.opendds.org`) of the OMG Data Distribution Service (DDS) with ANT, which supports various transport protocol properties, such as NAK-based and ACK-based reliability and flow control. ADAMANT leverages ANT to appropriately modify transport protocols and parameters settings as needed to maintain QoS.

OpenDDS provides a standards-based anonymous QoS-enabled pub/sub middleware for exchanging data in event-based distributed systems. It provides a global data store in which publishers and subscribers write and read data, respectively, so applications can communicate by publishing information they have and subscribing to information they need in a timely manner. OpenDDS supports various transport protocols, including TCP, UDP, IP multicast, and reliable multicast. OpenDDS also provides a pluggable transport framework that allows integration of custom transport protocols within OpenDDS. We chose the OpenDDS implementation due to (1) its source code being freely available, facilitating modification

| DDS QoS Policy | Description |
|---|---|
| User Data | Attaches application data to DDS entities |
| Topic Data | Attaches application data to topics |
| Group Data | Attaches application data to publishers, subscribers |
| Durability | Determines if data outlives the time when written or read |
| Durability Service | Details how durable data is stored |
| Presentation | Delivers data as group and/or in order |
| Deadline | Determines rate at which periodic data is refreshed |
| Latency Budget | Sets guidelines for acceptable end-to-end delays |
| Ownership | Controls writer(s) of data |
| Ownership Strength | Sets ownership of data |
| Liveliness | Sets liveness properties of topics, data readers, data writers |
| Time Based Filter | Mediates exchanges between slow consumers and fast producers |
| Partition | Controls logical partition of data dissemination |
| Reliability | Controls reliability of data transmission |
| Transport Priority | Sets priority of data transport |
| Lifespan | Sets time bound for "stale" data |
| Destination Order | Sets whether data sender or receiver determines order |
| History | Sets how much data is kept to be read |
| Resource Limits | Controls resources used to meet requirements |
| Entity Factory | Sets enabling of DDS entities when created |
| Writer Data Lifecycle | Controls data and data writer lifecycles |
| Reader Data Lifecycle | Controls data and data reader lifecycles |

**Table 1: DDS QoS Policies**

and experimentation and (2) its pluggable transport framework allowing integration of OpenDDS with the ANT framework.

• The **Waikato Environment for Knowledge Analysis** (Weka) is data mining software (`www.cs.waikato.ac.nz/ml/weka`) leverages key metrics captured from the ADAMANT prototype as shown in Figure 2. We use Weka to analyze ADAMANT's behavior for various transport protocols. Specifically, Weka captures (1) data update latency times (*i.e.*, the time from when the data writer writes the data to the time the data reader receives the data), (2) the number of updates received compared to the number of updates sent, and (3) network bandwidth usage statistics (*e.g.*, total bytes on the network and min/max/avg bandwidth usage).

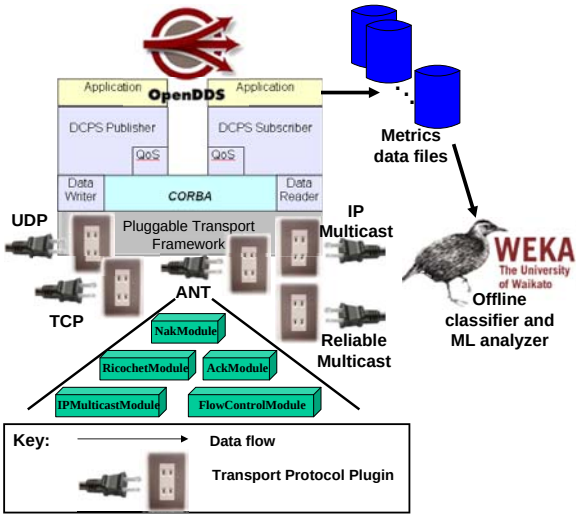


**Figure 2: ADAMANT System Architecture**

We chose the Weka data mining software due to its intuitive in-

terface, ease of use, robust analysis tools, and support for a wide range of machine learning techniques. These techniques include decision trees, multilayer perceptrons, and support vector machines. We input collected metrics and configuration information for the environment and transport protocol used into Weka. We have classified and analyzed the data using the various machine learning techniques to determine which techniques provide the best guidance in selecting a transport protocol for a given environment.

## 4.2 Evaluation Setup

To evaluate the behavior of ADAMANT with various transport protocols and protocol configuration setttings, we ran experiments and collected metrics using the Emulab network testbed (`www.emulab.net`). Emulab provides computing platforms and network resources that can be easily configured with the desired computing platform, OS, network topology, and network traffic shaping. It also provides facilities to capture network bandwidth usage.

Table 2 outlines the points of variability for the Emulab experiments. The NAKcast timeout period configures the amount of time that elapses before a receiver notifies the sender of lost packets. The Ricochet R value determines the number of packets received by an individual receiver before error correction data is sent to other receivers. The Ricochet C value determines the number of receivers to which an individual receiver sends error correction data. Table 3 outlines the data that is being collected to classify and evaluate middleware performance.

| Point of Variability | Values |
|---|---|
| # of data receivers | 3 - 25 |
| Frequency of sending data | 10Hz, 25 Hz, 50 Hz, 100Hz |
| % end-host network loss | 0 to 5 % |
| Processor speed | 850 MHz, 3 GHz |
| Network speed | 100 Mb/s, 1 Gb/s |
| Protocols used | NAKcast, Ricochet |
| NAKcast timeout | 0.5, 0.1, 0.05, 0.025 seconds |
| Ricochet R value | 4, 8 |
| Ricochet C value | 3, 6 |

**Table 2: Emulab Experiment Variables**

| Metrics | Units |
|---|---|
| Number of data updates received | integer |
| Latency of data updates | microseconds |
| Std. deviation of latency | microseconds |
| Maximum network bandwidth usage | bytes/sec |
| Minimum network bandwidth usage | bytes/sec |
| Average network bandwidth usage | bytes/sec |
| Network bandwidth usage | bytes |
| ReLate2 value | integer |

**Table 3: Metrics Captured from Experiments**

The ReLate2 value [7] is a metric that evaluates both reliability determined by the number of packets received by an application and packet latency. ReLate2 is calculated by multiplying the average latency by the percent packet loss.

## 4.3 Empirical Evaluations of Machine Learning Techniques for ADAMANT

This section presents our analysis of the experimental data and the results of the machine learning techniques described in Section 4.1. For each experiment we *subsampled* the collected data by selecting the transport protocol and settings that performed the best for an experiment run. We then used the machine learning techniques described in Section 4.1 on the subsampled data from all the experiments. The evaluation of the machine learning techniques included two criteria: (1) *basic accuracy*, which is the number of correctly learned protocols and protocol parameters for a given environment configuration divided by the total overall number of protocols and protocol parameters and (2) *area under the curve*, which plots *sensitivity* (*i.e.*, true positive rate) vs. *specificity* (*i.e.*, false negative rate) of the learning technique. These two criteria provide more complete analysis of the results vs. using a single criterion.

### *4.3.1 Subsampling the Data*

Since ADAMANT is concerned with reliability and latency, we focus on the ReLate2 value discussed in Section 4.2 to provide relevant subsampling. For a given environment configuration we selected the transport protocol and parameter settings that performed the best, *i.e.*, had the lowest ReLate2 value. For a set of 5 experiment runs for each environment configuration with varying transport protocols and parameter settings, the protocol and settings that performed the best remained. More specifically, 5 data points are left per experimental setup when including only the data from the protocol and parameters that produced the lowest ReLate2 value for the 5 runs. Across all experiment configurations, the collected metrics were reduced to roughly 150 data points that we used to train the machine learning techniques.

With this reduction, the learning techniques can start to select the most appropriate transport protocol and parameters settings. Using the training data the machine learning will accurately identify a protocol that produced the lowest ReLate2 values. Conversely, the machine learning will identify when a protocol and its parameterization will offer the best ReLate2 values, given an environment configuration.

### *4.3.2 Analysis of Results*

To explore which protocols and protocol parameters perform best under different configuration environments, we used the three machine learning techniques described in Section 4.1. The techniques utilized the reduced data set where the best performing protocols and protocol parameters were selected. All three techniques were trained using *n-fold cross validation*, where each fold partitions the data into a training set and testing set. The cross validation then averages the accuracy of the technique over all *n* folds, which provides greater coverage and increases the technique's robustness.

We selected 10 as the number of folds. The learning techniques were then trained and tested on each of the 10 folds. Since the reduced data points are not evenly distributed among the different experiment variables, n-fold cross validation is the best approach to maximizing data coverage while not skewing the learning results toward a particular transport protocol or parameter setting [9].

We used two metrics for evaluating the effectiveness of a machine learning technique. The first metric is the basic accuracy (also known as *1-loss accuracy*) which captures how well the technique determines the appropriate protocol and parameters. Basic accuracy has its greatest utility when the number of different environment configurations and protocols used for experiments are evenly distributed across all the types of experiments. The experimental data we collected, however, was not evenly distributed, *i.e.*, there were some protocol parameters that were used in more environment configurations than others. For example, we ran more experiments with NAKcast having a timeout value of 0.025 seconds than with a timeout value of 0.5 seconds.

The second metric for evaluating the effectiveness of machine learning techniques to determine appropriate protocols and parameters is the *area under the curve* (AUC), which plots sensitivity vs. specificity. For comparison, a learning technique that would select a transport protocol and parameters at random would graph as a straight line with a slope of 1 and the AUC would be 0.5. As a learning technique improves, its AUC increases. AUC attempts to provide some balance between learning techniques for ADAMANT since ADAMANT requires more complexity than a simple boolean yes/no response from the learning technique, *i.e.*, the specific trans-

port protocol and parameters to use. Moreover, a higher AUC value provides an indication of greater robustness.

Applying the three machine learning techniques outlined in Section 4.1 on the reduced set of 150 data points, we see clear differences in the results. The DT produced the best basic accuracy for determining appropriate transport protocols and parameters at 87% and the worst AUC score at .925. The ANN produced an accuracy that was lower at 85.3%, but provided the highest AUC at .966. The LLRC posted the lowest accuracy at 80% but also a higher AUC than DT at .935.

In general, all three techniques provide high accuracy results. While the differences in accuracy between the techniques are low, they are still significant. In machine learning with high levels of accuracy, a non-trivial amount of effort is required to modify a less accurate technique to match a more accurate technique. Our results indicate that ANNs are the most robust learning technique for ADAMANT, but DTs post a higher base accuracy. In particular, DTs exhibit some brittleness in being able to handle new, untrained environment configurations, whereas ANNs are more likely to provide ADAMANT resiliency for environment configurations not previously encountered. We are collecting more data to explore this assessment. LLRC appears to provide the worst results with the lowest base accuracy and only slightly better AUC than DTs.

While it appears ADAMANT might leverage ANNs initially with LLRC being the least useful in determining appropriate protocols and parameters, the results from DTs answer and raise interesting questions. Due to the nature of DTs, one can look at the implications the tree finds about the features. As shown in Figure 3, the tree utilizes relatively few features, *e.g.*, the amount of bandwidth used in bytes, the controlled variable of packet loss, and the controlled variable of number of receivers in the environment. While some of the controlled variables of # of receivers, % packet loss, and sending rate are used, the most important discriminator of the measured environment is the bandwidth usage.

```
J48 pruned tree
------------------

network_bytes <= 25612604
|   percent_packet_loss <= 1
|   |   network_bytes <= 11024041
|   |   |   percent_packet_loss <= 0
|   |   |   |   network_bytes <= 3275210: NAKcast-0.05 (3.0)
|   |   |   |   network_bytes > 3275210: NAKcast-0.025 (11.0)
|   |   |   percent_packet_loss > 0
|   |   |   |   num_receivers <= 3
|   |   |   |   |   network_bytes <= 4260177: NAKcast-0.05 (6.0)
|   |   |   |   |   network_bytes > 4260177
|   |   |   |   |   |   duration <= 2127.917476: NAKcast-0.025 (2.0)
|   |   |   |   |   |   duration > 2127.917476: NAKcast-0.1 (2.0)
|   |   |   |   num_receivers > 3: NAKcast-0.05 (4.0)
|   |   network_bytes > 11024041: NAKcast-0.025 (17.0/1.0)
|   percent_packet_loss > 1: NAKcast-0.025 (37.0/3.0)
network_bytes > 25612604
|   network_bytes <= 25729972: Ricochet-R8C3 (2.0)
|   network_bytes > 25729972
|   |   num_receivers <= 20: Ricochet-R4C3 (62.0)
|   |   num_receivers > 20
|   |   |   std_dev <= 5439.952831: Ricochet-R4C3 (2.0)
|   |   |   std_dev > 5439.952831: Ricochet-R8C3 (2.0)

Number of Leaves  :     12

Size of the tree :      23
```

**Figure 3: Initial Decision Tree**

# 5. CONCLUDING REMARKS

Developers of QoS-enabled pub/sub middleware and applications face a number of challenges in dynamic environments. To address these challenges ADAMANT combines QoS-enabled pub/-sub middleware with adaptive transport protocols and machine learning. This combination of technologies provides a basis for maintaining specified QoS even in dynamic environments. The results presented in this paper indicate that for dynamic environments decision trees and artificial neural networks are promising approaches for classifying the best protocols and protocol parameters to use. In particular, decision trees provide human readable details about which variables are most important to consider. Our future work will empirically evaluate the most appropriate techniques for ADAMANT under various dynamic environment conditions.

# 6. REFERENCES


[1] M. Balakrishnan, K. Birman, A. Phanishayee, and S. Pleisch. Ricochet: Lateral error correction for time-critical multicast. In *NSDI 2007: Fourth Usenix Symposium on Networked Systems Design and Implementation*, Boston, MA, 2007.

[2] B. Brynjulfsson, G. Hjalmtysson, K. Katrinis, and B. Plattner. Autonomic network-layer multicast service towards consistent service quality. In *AINA '06: Proceedings of the 20th International Conference on Advanced Information Networking and Applications*, pages 494–498, Washington, DC, USA, April 2006. IEEE Computer Society.

[3] M. Caporuscio, A. Carzaniga, and A. Wolf. Design and evaluation of a support service for mobile, wireless publish/subscribe applications. *Software Engineering, IEEE Transactions on*, 29(12):1059–1071, Dec. 2003.

[4] P.-C. David and T. Ledoux. *Software Composition*, chapter An Aspect-Oriented Approach for Developing Self-Adaptive Fractal Components, pages 82–97. Springer LNCS, Berlin / Heidelberg, 2006.

[5] P. Grace, G. Coulson, G. S. Blair, and B. Porter. Deep middleware for the divergent grid. In *Middleware '05: Proceedings of the ACM/IFIP/USENIX 2005 International Conference on Middleware*, pages 334–353, New York, NY, USA, 2005. Springer-Verlag New York, Inc.

[6] C. Herssens, S. Faulkner, and I. J. Jureta. Context-driven autonomic adaptation of sla. In *ICSOC '08: Proceedings of the 6th International Conference on Service-Oriented Computing*, pages 362–377, Berlin, Heidelberg, 2008. Springer-Verlag.

[7] J. Hoffert, A. Gokhale, and D. C. Schmidt. Evaluating Transport Protocols for Real-time Event Stream Processing Middleware and Applications. In *Proceedings of the 11th International Symposium on Distributed Objects, Middleware, and Applications (DOA '09)*, Vilamoura, Algarve-Portugal, Nov. 2009.

[8] Y. Huang and D. Gannon. A comparative study of web services-based event notification specifications. *Proceedings of the International Conference on Parallel Processing Workshops*, 0:7–14, 2006.

[9] M. Stone. Cross-validatory choice and assessment of statistical predictions. *Journal of the Royal Statistical Society B*, 36(1):111–147, 1974.

[10] S. Tarkoma and K. Raatikainen. State of the Art Review of Distributed Event Systems. Technical Report C0-04, University of Helsinki, 2006.

[11] Y. Tock, N. Naaman, A. Harpaz, and G. Gershinsky. Hierarchical clustering of message flows in a multicast data dissemination system. In *Proceedings of Parallel and Distributed Computing and Systems (PDCS 2005)*, Nov. 2005.

[12] P. Vienne and J.-L. Sourrouille. A middleware for autonomic qos management based on learning. In *SEM '05: Proceedings of the 5th international workshop on Software engineering and middleware*, pages 1–8, New York, NY, USA, 2005. ACM.