Maintaining Publish/Subscribe Middleware QoS in Dynamic Environments *

Joe Hoffert
Adviser: Douglas C. Schmidt
Institute for Software Integrated Systems, Dept of EECS
Vanderbilt University, Nashville, TN 37203
{jhoffert}@dre.vanderbilt.edu

ABSTRACT

Quality of service (QoS)-enabled publish/subscribe (pub/-sub) middleware provides powerful support for 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 slow human response times and the complexity of managing multiple interrelated QoS settings.

Autonomic adaptation provides a promising approach to maintaining QoS properties of QoS-enabled pub/sub middleware in dynamic environments. By monitoring key system resources and environment aspects, changes to middleware mechanisms (*e.g.*, associations of publishers and subscribers to transport protocols) can be made autonomically to maintain specified QoS. This paper describes the research we are conducting on highly configurable QoS-enabled middleware, adaptive transport protocols, system monitoring, supervised machine learning techniques, and autonomic adaptation to maintain specified QoS as the system environment and configuration dynamically changes.

Categories and Subject Descriptors

C.2.4.b [**Distributed Systems**]: Distributed applications; C.4.f [**Performance of Systems**]: Reliability, availability, and serviceability

Keywords

Pub/sub Middleware, Event-based Distributed Systems, Real-time and embedded systems, Quality of Service

1. INTRODUCTION

Emerging trends and challenges. The number and type of distributed systems that utilize publish/subscribe (pub/sub) technologies are growing due to the advantages of performance, cost, and scale as compared to single computers [8, 9]. 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 fractionated spacecraft constellations. 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 as an atomic unit).

While tunable policies provide fine-grained control of system QoS, several challenges emerge when developing pub/-sub systems deployed in dynamic environments. Mechanisms used by the middleware to ensure certain QoS properties for a given environment configuration may not be applicable for a different environment configuration. For example, a simple unicast protocol (such as UDP) may provide adequate QoS regarding latency 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 publishers sending UDP messages 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 (again such as UDP) provide low overhead but no end-to-end reliability. Other protocols (such as TCP) provide reliability, but incur unbounded latencies due to acknowledgment-based retransmissions. Still other protocols balance reliability and low latency, but provide benefit over other protocols only for specific environment configurations. Determining when to modify parameters of a particular transport protocol or switch from one transport protocol to another can be a complex decision. Moreover, human intervention is often not responsive enough to meet system timeliness requirements.

Research approach. The remainder of this document

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

DEBS 2009, July 6-9, 2009, Nashville, TN, USA

describes the research we are conducting to address the challenges of maintaining QoS in dynamic environments by integrating and enhancing (1) QoS-enabled pub/sub middleware, (2) adaptive transport protocols, (3) environment monitoring, (4) supervised machine learning, and (5) autonomic adaptation of transport protocols. Section 2 shows the challenges in more detail. Section 3 presents related work and the shortcomings of this work for the challenges presented. Section 4 outlines our proposed research to fully address the challenges. Section 5 details the status of our current research. Section 6 presents concluding remarks.

2. MOTIVATING EXAMPLE - SEARCH AND RESCUE (SAR) OPERATIONS FOR DISASTER RECOVERY

To motivate the need for autonomic adaptation of QoS-enabled pub/sub middleware, this section describes the research challenges associated with search and rescue (SAR) operations. These operations help locate and extract survivors in a large metropolitan area after a regional catastrophe, such as a hurricane, earthquake, or tornado. SAR operations utilize 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.

Figure 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 then sent

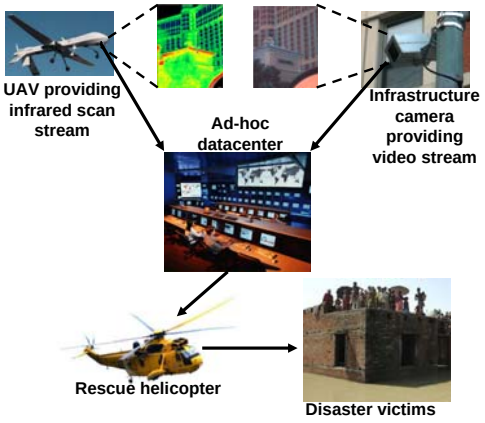

Figure 1: Search and Rescue Motivating Example

to a datacenter, where they are processed by fusion applications to detect survivors. Once a survivor is detected the application can develop a three dimensional view and highly accurate position information so that rescue operations can commence.

Several challenges that arise with SAR operations in dynamic environments are summarized below. Section 4 then describes research techniques we are using to address these challenges.

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 take advantage of these 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.

3.1 Adaptive Middleware

Mobility Support Service (MSS). MMS [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 to be 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 solely focuses on supporting pub/sub mobility, however, and therefore does not address Challenge 2 in Section 2.2. Moreover, MSS does not address Challenge 4 in Section 2.4 since it does not present a standardized and robust interface for QoS.

Gridkit. 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 will operate 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, however, and does not address Challenge 1 in Section 2.1. Within Gridkit no consideration is given to making timely adaptations based on 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.

SAFRAN. 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 configurations 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. Additionally, SAFRAN does not address Challenge 4 in Section 2.4 since it provides no standard QoS specification.

3.2 Machine Learning in Support of Autonomic Adaptation

Vienne and Sourrouille [11] 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 provides a customized QoS specification that does not address Challenge 4 in Section 2.4. 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.

3.3 Infrastructure for Autonomic Computing

Grace *et al.* [6] describe an architecture metamodel for adapting components that implement coordination for reflective middleware distributed across peer devices. This work also investigates supporting reconfiguration types in various environmental conditions. The proposed architecture metamodel, however, only provides proposed infrastructure for autonomic adaptation and reconfiguration and does not directly address the challenges in Section 2.

Valetto *et al.* [10] developed network features in support of service awareness to enable autonomic behavior. Their work targets communication services within a Session Initiation Protocol (SIP) enabled network to communicate monitoring, deployment, and advertising information. As an autonomic computing infrastructure, however, this work does not directly address any of the challenges in Section 2.

3.4 Autonomic Adaption of Service Level Agreements

Herssens *et al.* [7] 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.

3.5 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. The framework includes (1) high level descriptions of policies and objectives, (2) a multicast topology management protocol supported by network nodes, (3) measurement and monitoring infrastructure, and (4) a control component that autonomously manipulates the protocol and infrastructure to reduce variance. ARMDS does not address Challenge 2 in Section 2.2, however, nor does it address Challenge 4 in Section 2.4.

4. RESEARCH APPROACH

This section describes how our research combines and enhances the following technologies to resolve the challenges presented in Section 2 and address the gaps in related work described in Section 3.

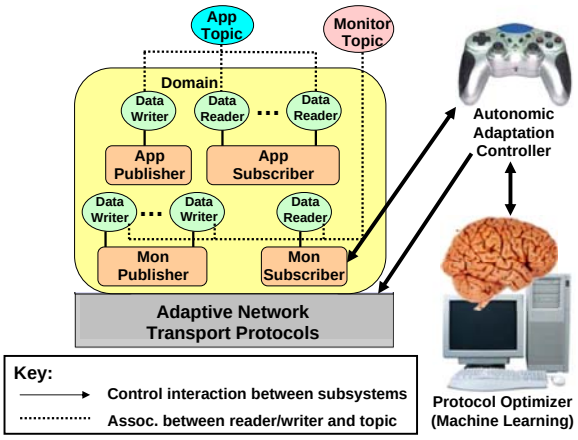

Figure 2: ADAMANT Architecture

• Standard QoS-enabled pub/sub middleware addresses the scalability of Challenge 3 in Section 2.3 by decoupling data senders from data receivers. Applications interested in published data can receive it any time without knowledge of the data sender. Moreover, standard QoS-enabled middleware addresses the QoS standardization of Challenge 4 in Section 2.4.

• Supervised machine learning helps 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. The machine learning component includes features for several different environment configurations and supervised training, such as decision trees, multilayer perceptrons, and support vector machines, to learn the correct protocol and parameters. The machine learning will interpolate and extrapolate its learning based on the current environment configuration, which might not have been included in the supervised training.

• Adaptive network transport protocols work to address Challenge 1 in Section 2.1 and Challenge 2 in Section 2.2 by providing the infrastructure 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. Adaptive network transport protocols support not only fine grained control of a protocol's parameters but also switching from one protocol to another to provide the adaptation needed within dynamic environments.

• Environment monitoring works to address Challenge 1 in Section 2.1 by providing environment configuration information. Relevant environment configuration values are monitored as needed such as the number of subscribers, the percentage of network packet loss, and the sending rate of the data. Our architecture leverages the pub/sub middleware that is being autonomically adapted by creating a monitoring topic. The adaptation infrastructure then uses this topic to disseminate monitoring information to interested subscribers. In our architecture, the subscriber interested in the monitoring data is the autonomic adaptation controller.

• Autonomic adaptation addresses Challenge 1 in Section 2.1 and Challenge 2 in Section 2.2 by (1) subscribing to the monitoring topic and querying relevant values from the environment monitoring, (2) determining when QoS is not being met, (3) activating the machine learning component which will determine an appropriate transport protocol and parameters, (3) retrieving the recommended protocol settings, and (4) transitioning the adaptive network transport protocols to use the recommended settings.

5. CURRENT STATUS AND PLAN

Figure 3 shows how we integrated and enhanced the OpenDDS implementation (`www.opendds.org`) of the OMG Data Distribution Service (DDS) standard (`www.omg.org/spec/DDS`) with the Adaptive Network Transports (ANT) framework. Additionally, Figure 3 highlights the capture of key metrics into data files and the offline classification of middleware behavior using the WEKA 3 data mining software (`www.cs.waikato.ac.nz/ml/weka`).

OpenDDS 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, so applications can communicate by publishing information they have and subscribing to information they need in a timely manner. OpenDDS provides support for 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 OpenDDS due to its open-source availability (which facilities modification and experimentation) and its support for a *pluggable transport framework* that allows application developers to create custom transport protocols for sending/receiving data. We chose the ANT framework

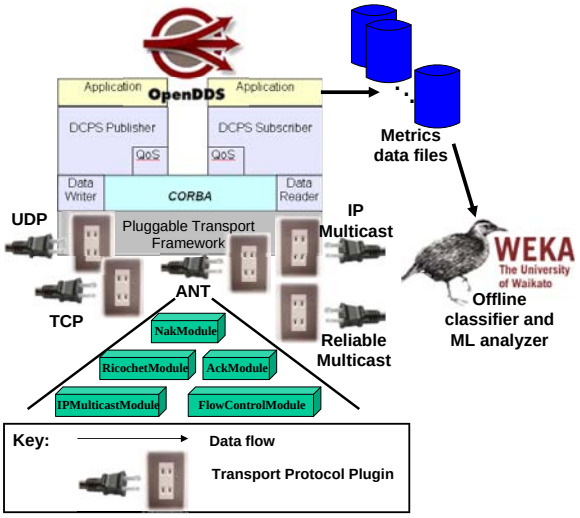

Figure 3: Current Research Prototype

due to its infrastructure for composing transport protocols. ANT builds upon the properties provided by the scalable reliable multicast-based Ricochet transport protocol [1]. ANT also provides a modular framework whereby protocol modules can be tuned, enhanced, and replaced to maintain specified QoS. Some of the modules supported by ANT include (1) an IP multicast module to efficiently disseminate data from one publisher to multiple subscribers, (2) NAKcast, a

NAK-based reliable multicast module, and (3) a Ricochet module which provides a trade-off between latency and reliability. NAKcast and Ricochet also allow modification of parameters to affect latency, reliability, and bandwidth usage.

We chose the Weka data mining software due to its user-friendly interface, 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 are capturing (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, min/max/avg bandwidth usage). We input these collected metrics and configuration information for the environment and transport protocol used into Weka. We are classifying and analyzing 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.

As shown in Figure 4 we are currently running experiments and collecting 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. Moreover, Emulab provides facilities to capture network bandwidth usage.

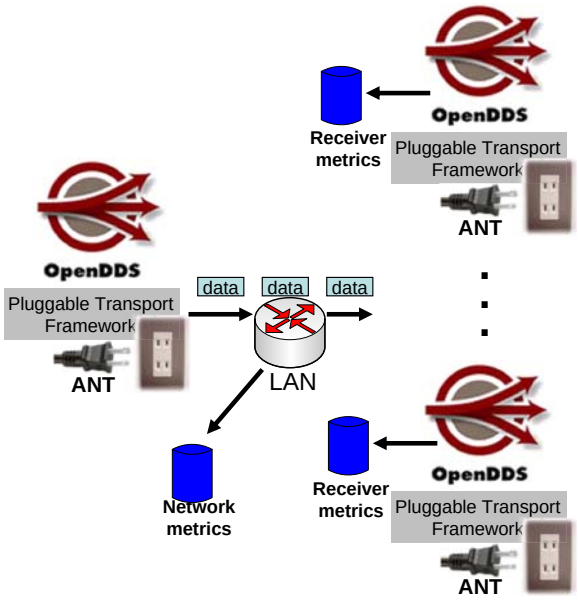

Figure 4: Emulab Experimental Setup

Table 1 outlines the points of variability for the Emulab experiments while Table 2 outlines the data that is being collected to classify and evaluate middleware performance. The NAKcast timeout value listed in Table 1 is the value in seconds that NAKcast waits before it sends out negative acknowledgments for data packets that is has detected are missing. The Ricochet R value listed in Table 1 is the number of packets that Ricochet receives before it sends out recovery information to the other receivers.

For example, if Ricochet's R value is 4 then Ricochet will wait until it receives 4 data packets before it creates and sends out a recovery packet based on the 4 data packets received. Ricochet also provides a parameter to adjust the number of receivers that receive the recovery packet. Currently we are using the default value of 3 but plan to run experiments where this value is varied as well.

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

Table 1: Emulab Experiment Variables

Metrics	Units
Number of data updates received	integer
Latency of data updates	microseconds
Network bandwidth usage	bytes, bytes/sec

Table 2: Metrics Captured from Experiments

Figures 5, 6, and 7 show that capturing reliability metrics (in the number of updates received), latency times, and network bandwidth usage showcases important trade-offs between transport protocols in general and NAKcast and Ricochet in particular. Modifying NAKcast's timeout value directly affects the latency of recovered data, the amount of network bandwidth used, and to a low probability the number of updates received. Likewise, modifying Ricochet's R and C values has an direct impact on the latency of recovered data, the number of data updates received, and the network bandwidth used.

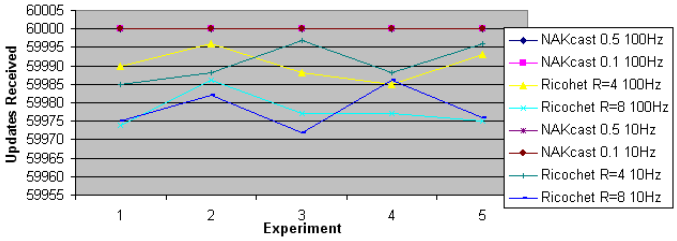

Figure 5: Updates Received for NAKcast w/ timeouts of 0.5, 0.1 sec; Ricochet w/ R=4, R=8; sending rates of 10Hz, 100Hz; 3 receivers, 1% endhost packet loss

Moreover, the protocol itself has an effect on the distribution of the latency times since NAKcast sends out NAKs only when needed while Ricochet regularly send out recovery packets. Since we are capturing the raw latency times we can use statistical methods on these times to provide fine-grained analysis of the middleware's behavior as outlined in Figure 8.

We are also integrating ANT with the OpenSplice DDS implementation (`www.opensplice.com`) due to its (1) recent availability as open source, (2) robust QoS and scalability support, (3) commercial and military deployments, and (4) support for a network plug-in framework that provides an efficient C API and allows application developers to customize transport protocols via an XML configuration file. We plan to run identical ADMANT experiments as was done with OpenDDS. Collecting metrics using both OpenDDS

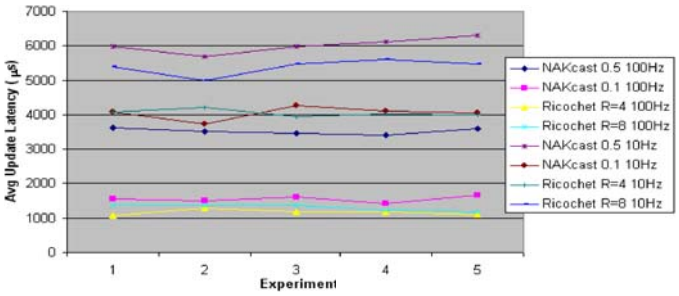

Figure 6: Average Update Latency Times for NAKcast w/ timeouts of 0.5 and 0.1 sec; Ricochet w/ R=4, R=8; sending rates of 10Hz, 100Hz; 3 receivers, 1% endhost packet loss

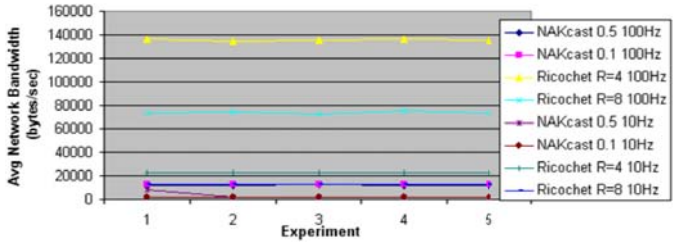

Figure 7: Average Network Bandwidth Usage for NAKcast w/ timeouts of 0.5 and 0.1 sec; Ricochet w/ R=4, R=8; sending rates of 10Hz, 100Hz; 3 receivers, 1% endhost packet loss

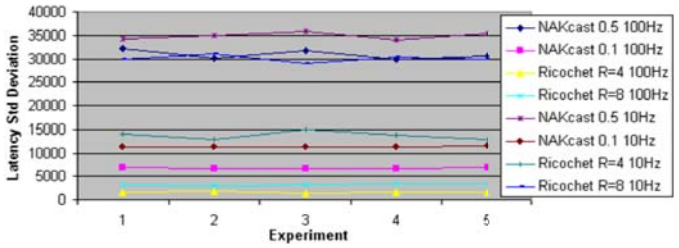

Figure 8: Latency Std Deviation for NAKcast w/ timeouts of 0.5 and 0.1 sec; Ricochet w/ R=4, R=8; sending rates of 10Hz, 100Hz; 3 receivers, 1% endhost packet loss

and OpenSplice will provide additional variability to the machine learning and classification thus increasing its robustness. In addition, OpenSplice's support of the Real-time Pub/Sub (RTPS) protocol allows comparison of NAKcast and Ricochet with RTPS.

6. CONCLUDING REMARKS

Developers of QoS-enabled pub/sub systems face a number of challenges when developing their applications for dynamic environments. To address these challenges, we are researching the use of QoS-enabled pub/sub middleware with adaptive transport protocols, environment monitoring, machine learning, and autonomic adaptation. This combination of technologies provides a unique basis for maintaining specified QoS within changing environments.

SDS

7. REFERENCES

[1] Mahesh Balakrishnan, Ken Birman, Amar Phanishayee, and Stefan 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] Bjorn Brynjulfsson, Gisli Hjalmtysson, Kostas Katrinis, and Bernhard 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.L. 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] Pierre-Charles David and Thomas Ledoux. *Software Composition*, chapter An Aspect-Oriented Approach for Developing Self-Adaptive Fractal Components, pages 82–97. Springer LNCS, Berlin / Heidelberg, 2006.

[5] Paul Grace, Geoff Coulson, Gordon S. Blair, and Barry 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] Paul Grace, Geoff Coulson, Gordon S. Blair, and Barry Porter. A distributed architecture meta-model for self-managed middleware. In *ARM '06: Proceedings of the 5th workshop on Adaptive and reflective middleware (ARM '06)*, page 3, New York, NY, USA, 2006. ACM.

[7] Caroline Herssens, Stéphane Faulkner, and Ivan 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.

[8] Yi Huang and Dennis 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] Sasu Tarkoma and Kimmo Raatikainen. State of the Art Review of Distributed Event Systems. Technical Report C0-04, University of Helsinki, 2006.

[10] Giuseppe Valetto, Laurent Walter Goix, and Guillaume Delaire. Towards service awareness and autonomic features in a sip-enabled network. In *Autonomic Communication*, pages 202–213, Berlin, Heidelberg, 2006. Springer-Verlag.

[11] Patrice Vienne and Jean-Louis 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.