Quality of Service Management
Techniques for Publish/Subscribe-based
Distributed Real-time & Embedded
Systems

http://www.dre.vanderbilt.edu/~jhoffert/proposal.pdf

Joe Hoffert
jhoffert@dre.vanderbilt.edu

Institute for Software Integrated Systems
Vanderbilt University
Nashville, Tennessee
• Problem Statement
• Focus Area 1: QoS Configuration Management
• Focus Area 2: Evaluation of QoS Mechanisms
• Focus Area 3: QoS Support in Dynamic Environments
• Concluding Remarks
Context: QoS-enabled Publish/Subscribe for DRE Systems

- Pub/Sub enables separation of concerns - decouples senders & receivers
- QoS enables finer-grained control of system behavior/properties

Client-server technology for DRE systems is being enhanced (or replaced) by pub-sub middleware
  - i.e., client-server & pub/sub are complementary technologies

Characteristics of Pub/Sub
- Decouples location via anonymous pub/sub
- Decouples time via asynchronous, time-independent data distribution
- Decouples redundancy via unbounded # of senders/receivers

Manifesto for QoS-enabled Pub/Sub
The right data…to the right place…at the right time
Example: QoS-enabled Pub/Sub DRE Systems

• Net-centric & large-scale “systems of systems”
  • e.g., satellite systems, shipboard computing environments, emergency response systems

• Satisfying tradeoffs between multiple (often conflicting) QoS demands
  • e.g., secure, real-time, reliable

• Regulating & adapting to (dis)continuous changes in runtime environments
  • e.g., online prognostics, dependable upgrades, availability of critical tasks, dynamic resource management

DRE systems increasingly realized by composing loosely-coupled services, e.g., pub/sub
Challenges in Realizing DRE Pub/Sub Systems

Variability in the **solution space** (systems integrator role)
- Diversity in platforms, languages, protocols & tool environments
- Enormous accidental & inherent complexities
- Continuous evolution & change
- Management of diverse QoS requirements

Focus on QoS Management
QoS Configuration Development: Complexity #1

Conventional QoS configuration design performed manually at system design time

Manual QoS Configuration Techniques

System Lifecycle Timeline

(System Deployment)
QoS Configuration Development: Complexity #1

QoS configurations are not evaluated until run-time.

- Manual QoS Configuration Techniques
- QoS Configuration Validation

System Lifecycle Timeline

- System Deployment
QoS Configuration Development: Complexity #1

Disconnect between design & validation of QoS configuration adds complexity:
- Modification time & cost is increased
- It is difficult to associate configuration problems with the design
Evaluating QoS mechanisms for pub/sub middleware in different environments increases complexity:

- Developers need to instrument pub/sub middleware for evaluation
- Developers need to evaluate multiple QoS concerns simultaneously

A single pub/sub DRE system can be deployed in many different operating environments.

Developers must thoroughly understand how QoS mechanisms in a pub/sub context support *multiple* QoS concerns under different environments.

QoS Configuration Development: Complexity #2

QoS Mechanism Space in Operating Environment A

- Adequate QoS mechanism

QoS Mechanism Space in Operating Environment B

- Adequate QoS mechanism
- Inadequate QoS mechanism

Evaluating QoS mechanisms for pub/sub middleware in different environments increases complexity:

- Developers need to instrument pub/sub middleware for evaluation
- Developers need to evaluate multiple QoS concerns simultaneously
QoS Configuration Development: Complexity #3

- QoS mechanisms are determined for the operating environment before system execution.

- As operating environment changes initial QoS mechanisms might not longer be adequate.

Updating QoS mechanisms in dynamic environments increases complexity:
- An adequate QoS mechanism needs to be found.
- Transitioning to a new QoS mechanism needs to be done in a timely way.
Presentation Outline

• Problem Statement
• Focus Area 1: QoS Configuration Management
• Focus Area 2: Evaluation of QoS Mechanisms
• Focus Area 3: QoS Support in Dynamic Environments
• Concluding Remarks
Focus Area 1: QoS Configuration Management

Pub/sub QoS configurations have often been validated at run-time, implying:

1. A lengthy iteration time for QoS configuration design
2. A disconnect between discovering the problem & solving it, i.e., loss of context
3. A coupling of QoS design with implementation, i.e., manipulating code to fix faulty design

```
// Create the Datareader.
// Add keep_all_history and reliability
// to the data reader to get all
// the data.
DDS::DataReaderQos dr_qos;
sub->get_default_datareader_qos(dr_qos);

dr_qos.history.kind = DDS::KEEP_ALL_HISTORY_QOS;
dr_qos.reliability.kind = DDS::RELIABLE_RELIABILITY_QOS;
dr_qos.reliability.max_blocking_time.sec = 100;
dr_qos.reliability.max_blocking_time.nanosec = 0;
dr_qos.durability.kind = DDS::TRANSIENT_LOCAL_DURABILITY_QOS;
```

QoS design & implementation coupled together

System Lifecycle Timeline

- Manual QoS Configuration Techniques
- QoS Configuration Validation
- System Deployment
Benefits of design-time QoS configuration validation include:

1. Shortening iteration time for QoS configuration design
2. Maintaining context between discovering & solving the problem
3. Decoupling QoS design from implementation, i.e., separating concerns
4. Generating implementation from design, i.e., reducing accidental complexity

Design-time QoS validation increases developer productivity & reduces accidental complexity for pub/sub DRE systems
Motivating Example: Magnetospheric Multiscale Mission

Scenario

The NASA Magnetospheric Multiscale (MMS) Mission

- Study various aspects of the earth’s magnetosphere
  - turbulence in key boundary regions
  - magnetic reconnection
  - charged particle acceleration

- Comprised of five co-orbiting & coordinated satellites, instrumented identically
Motivating Example: Magnetospheric Multiscale Mission

Systems like MMS can leverage OMG’s Data Distribution Service (DDS)

Provides flexibility, power, & modular structure by decoupling:
• **Location** – anonymous pub/sub
• **Redundancy** – any # of readers/writers
• **Time** – asynchronous, time-independent data distribution

• **Platform** – platform-independent IDL
• **Protocol** – standard data exchange
• **Quality of Service** – fine-grained QoS control
Motivating Example: Magnetospheric Multiscale Mission

• Data-centric Pub-Sub (DCPS) entities (e.g., topics, data readers/writers) configurable via nearly two dozen QoS policies

• Example QoS tailored to data distribution for DRE systems, such as MMS

  • **DEADLINE**
    • Establishes contract regarding rate at which periodic data is refreshed

  • **LATENCY_BUDGET**
    • Establishes guidelines for acceptable end-to-end delays

  • **TIME_BASED_FILTER**
    • Mediates exchanges between slow consumers & fast producers

  • **RESOURCE_LIMITS**
    • Controls resources utilized by service

  • **RELIABILITY (BEST_EFFORT, RELIABLE)**
    • Enables use of real-time transports for data

  • **HISTORY (KEEP_LAST, KEEP_ALL)**
    • Controls which (of multiple) data values are delivered

  • **DURABILITY (VOLATILE, TRANSIENT, PERSISTENT)**
    • Determines if data outlives time when they are written

  • … & many more …

• Request/offered compatibility checked by DDS **at runtime**

• Consistency checked by DDS **at runtime**
Motivating Example: Magnetospheric Multiscale Mission

MMS Requirements

- Multiple QoS properties, e.g.,
  - Prioritization for high-value data
  - Reliability for mission critical data
- Valid QoS configurations for proper execution
  - Correctly specified QoS properties
  - Correctly managed related & interacting QoS
  - Implementation artifacts that accurately represent design

Key:
- = MMS ground station
- = other satellites
- = MMS spacecraft
- = data flow

Inter-arrival data spacing
Data with time deadlines
Potentially conflicting QoS
Data reliability
Data priority
Data redundancy
Provisioning of data resources
Ordered data
Determining liveness

// Add keep_all history and reliability
// to the data reader to get all the samples.
dr_qos.history.kind = DDS::KEEP_ALL_HISTORY_QOS;
dr_qos.reliability.kind = DDS::RELIABLE_RELIABILITY_QOS;
dr_qos.reliability.max_blocking_time.sec = 100;
dr_qos.reliability.max_blocking_time.nanosec = 0;
dr_qos.durability.kind = DDS::TRANSIENT_LOCAL DURABILITY
QoS Policy Configuration Challenges (1/4)

Challenge 1: Reduce accidental complexities with QoS policy configurations early in development lifecycle, ideally when designed

- QoS Policy Variability
  - i.e., only certain associations are valid & only certain parameters, parameter types, values are valid
QoS Policy Configuration Challenges (2/4)

Challenge 2: Resolve QoS configuration incompatibilities *early* in developmental lifecycle

- **QoS Policy Compatibility**
  - i.e., QoS policies for the communicating entities must be compatible between what’s requested & offered

Key:
- = MMS ground station
- = MMS spacecraft
- = intended data flow

Deadline of 10 ms specified
Deadline of 5 ms requested

**Data will not be transferred**

X
QoS Policy Configuration Challenges (3/4)

Challenge 3: Resolve QoS configuration inconsistencies *early* in developmental lifecycle

- **QoS Policy Consistency**
  - i.e., QoS policies for any one entity must be consistent with each other

**Key:**
- 📥 = MMS ground station
- 🚀 = MMS spacecraft
- → = intended data flow
QoS Policy Configuration Challenges (4/4)

Challenge 4: Reduce accidental complexity of manual transformation for QoS configuration

- Faithful Transformation
  - i.e., implementation must accurately reflect design

```java
PacketTypeSupport_var pts = new PacketTypeSupportImpl();

if (DDS::RETCODE_OK != pts->register_type(participant.in (), "") ){
  cerr << "register_type failed." << endl;
  exit(1); 
}

CORBA::String_var type_name = pts->get_type_name();

DDS::TopicQos topic_qos;
participant->get_default_topic_qos(topic_qos);
topic_qos.reliability.kind = DDS::RELIABLE_RELIABILITY_QOS;
topic_qos.reliability.max_blocking_time = 200;
DDS::Topic_var topic = 
  participant->create_topic("Movie Discussion List",
                    type_name.in (),
                    topic_qos,
                    DDS::TopicListener::_nil());

if (CORBA::is_nil(topic.in())) {
  cerr << "create_topic failed." << endl;
  exit(1); 
}
```
### Related Research


# QoS Configuration Management: Related Work

## Related Research

<table>
<thead>
<tr>
<th>Description</th>
<th>Details</th>
</tr>
</thead>
</table>

Technique is good for *run-time* exploration of QoS adaptation methods
### Related Research

<table>
<thead>
<tr>
<th>Author(s)</th>
<th>Title</th>
<th>Conference</th>
<th>Pages</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>Related Research</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>-------------------</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## Properties for QoS Configuration Management

What properties help us assess research to manage QoS configurations of pub/sub middleware for DRE systems?

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Shorten design &amp; validation period</strong></td>
<td>Does the technique shorten the time between QoS design &amp; QoS validation?</td>
</tr>
<tr>
<td>** Maintain context between problem &amp; solution**</td>
<td>Does the technique help maintain the context between when a QoS configuration problem is found &amp; when it’s addressed?</td>
</tr>
<tr>
<td><strong>Decouple design from implementation</strong></td>
<td>Is the technique only applicable to a particular implementation?</td>
</tr>
<tr>
<td><strong>Auto generate implementation</strong></td>
<td>Can the technique automatically generate implementation artefacts from a validated design?</td>
</tr>
</tbody>
</table>
QoS Configuration Management: Related Work

Does the technique support auto-generation of the implementation?

Automated

Manual

PrismTech

PrismTech, RTI

Ye et al.

Li et al.

Carzaniga et al.

Design-time validation

Run-time validation

Implementation specific

Implementation agnostic
QoS Configuration Management: Related Work

When does the technique validate the design?

- Automated
  - Design-time validation
  - Run-time validation

- Manual
  - Implementation specific
  - Implementation agnostic

Ye et al., PrismTech
Carzaniga et al.
Li et al., PrismTech, RTI
QoS Configuration Management: Related Work

Is the technique tied to a particular implementation?

- Automated
  - Design-time validation
    - Ye et al.
  - Run-time validation
    - Li et al.

- Manual
  - Implementation specific
  - Implementation agnostic

PrismTech
PrismTech, RTI
Carzaniga et al.
QoS Configuration Management: Related Work

Gap: Design-time validation, automatically generated implementation, implementation agnostic

Current gap makes it hard for developers to accurately design & implement QoS configurations
Solution Approach: Distributed QoS Modeling Language

Solution: domain-specific modeling language for QoS configurations (via GME)

- Automated
  - Design-time validation
  - Run-time validation
- Manual
  - Design-time validation
  - Run-time validation

Implementation specific

Implementation agnostic

PrismTech

PrismTech, RTI

Ye et al.

Li et al.

Carzaniga et al.

DQML
Addressing Challenges of QoS Management (1/4)

DQML addresses challenge 1 by allowing only valid associations, parameters, types, & values

DQML manages QoS configuration variability:

- Associations
- QoS policy parameters
- Parameter types
- Parameter values
DQML supports QoS compatibility constraint checking:

![Constraint Violations](image)

DQML addresses challenge 2 by checking for incompatibilities at **design-time**
DQML addresses challenge 3 by checking for inconsistencies at *design-time*.

DQML supports QoS consistency constraint checking:

**Constraint Violations**

- **Constraint**: `meta::DataReader::Consistent_Deadline_Timebased`
- **Description**: For consistency, the `DeadlineQosPolicy`'s deadline must be greater than or equal to the `TimeBasedFilterQosPolicy`'s minimum separation.

**Variable**

- **self**: `DataReader { kind: meta::DataReader; path: /Root Folder/NewDDSQoS; }`
- **dr_deadline**: `ocl::Integer { 5 }`
- **dr_deadline_policies**: `ocl::Set { size: 1; }
- **dr_min_sep**: `ocl::Integer { 10 }

---

**Diagram:**

- **TimeBasedFilter**: 10ms
- **DataReader**
- **Deadline**: 5ms
DQML supports the MMS requirements of
(1) Correctly specified QoS properties,
(2) Correctly managed related & interacting QoS, and
(3) Implementation artifacts that accurately represent design.

DQML addresses challenge 4 by accurately transforming design to implementation.

DQML generates correct implementation artifacts:

Addressing Challenges of QoS Management (4/4)
Comparison: QoS Configuration Management

<table>
<thead>
<tr>
<th>Technique</th>
<th>Design-time validation</th>
<th>Pub/sub</th>
<th>Implementation agnostic</th>
<th>Automate implementation</th>
</tr>
</thead>
<tbody>
<tr>
<td>DQML</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Ye, J.</td>
<td></td>
<td></td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Krishna, A.</td>
<td>X</td>
<td></td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>McKinnon, D.</td>
<td>X</td>
<td></td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Carzaniga, A.</td>
<td></td>
<td>X</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Li, G.</td>
<td></td>
<td></td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>RTI</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PrismTech</td>
<td>X</td>
<td></td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

Current limitation of DQML:

- A valid QoS configuration does not provide run-time assurance of QoS
- Addressed by focus area 2 (existing research)
Performed quantitative analysis of DQML in addressing transformation challenge

**Minimal MMS scenario contains:**
- 5 MMS satellites
- 1 non-MMS satellite
- 1 ground station

Each satellite/ground station contains 1 data reader & 1 data writer for data dissemination

**DDS Benchmarking Environment (DBE):**
- Evaluates DDS implementations
- Deploys MMS data readers & data writers onto platforms
- Uses text files for QoS settings
DQML’s DBE interpreter
(1) traverses the QoS configuration model & (2) automatically generates the DBE QoS settings files:

```cpp
// Check that there is only one QoS policy as there should be
if (comms.size() > 1)
{
    std::string error_msg("ERROR: Multiple ");
    error_msg += qos.name + "QoS Policies specified for ";
    error_msg += entity_name + " " + dds_entity->get_name();
    error_msg = ",\nQoS Policy Ignored";
   AFXMessageBox(error_msg.c_str());
} else {

    // Get the one and only "qos_name" QoS Policy
    // and print out its attributes
    std::multiset<ConnectionEnd>::const_iterator iter(comms.begin());
    ConnectionEnd endPt = *iter;

    // If we make it here then the end is an eco
    std::set<Attribute> attrs = eco->getAttributes();
    std::set<Attribute>::const_iterator attr_iter(attrs.begin());

    // Total C++ statements developed for DBE interpreter: 160
```

C++ interpreter statement effort ≤ configuration statement effort considering
1. Variability concerns
2. Compatibility/consistency concerns
3. Transformation concerns
**Analysis: DQML, DBE, & MMS Scenario (3/4)**

<table>
<thead>
<tr>
<th>Evaluation Criteria</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>(H1) Reducing Configuration Effort</td>
<td>Hypothesize for systems with 7 or more data readers &amp; writers, DQML reduces development of DBE implementation artifacts vs. manual methods</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>QoS Policy</th>
<th># of Parameters</th>
<th>Parameter Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Deadline</td>
<td>1</td>
<td>int</td>
</tr>
<tr>
<td>Destination Order</td>
<td>1</td>
<td>enum</td>
</tr>
<tr>
<td>Durability</td>
<td>1</td>
<td>enum</td>
</tr>
<tr>
<td>Durability Service</td>
<td>6</td>
<td>5 ints, 1 enum</td>
</tr>
<tr>
<td>History</td>
<td>2</td>
<td>1 int, 1 enum</td>
</tr>
<tr>
<td>Latency Budget</td>
<td>1</td>
<td>int</td>
</tr>
<tr>
<td>Lifespan</td>
<td>1</td>
<td>int</td>
</tr>
<tr>
<td>Liveliness</td>
<td>2</td>
<td>1 int, 1 enum</td>
</tr>
<tr>
<td>Ownership</td>
<td>1</td>
<td>enum</td>
</tr>
<tr>
<td>Ownership Strength</td>
<td>1</td>
<td>int</td>
</tr>
<tr>
<td>Reliability</td>
<td>2</td>
<td>1 int, 1 enum</td>
</tr>
<tr>
<td>Resource Limits</td>
<td>3</td>
<td>3 ints</td>
</tr>
<tr>
<td>Transport Priority</td>
<td>1</td>
<td>int</td>
</tr>
<tr>
<td>User Data</td>
<td>1</td>
<td>string</td>
</tr>
<tr>
<td>Writer Data Lifecycle</td>
<td>1</td>
<td>bool</td>
</tr>
<tr>
<td><strong>Data Writer Parameters</strong></td>
<td><strong>25</strong></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>QoS Policy</th>
<th># of Parameters</th>
<th>Parameter Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Deadline</td>
<td>1</td>
<td>int</td>
</tr>
<tr>
<td>Destination Order</td>
<td>1</td>
<td>enum</td>
</tr>
<tr>
<td>Durability</td>
<td>1</td>
<td>enum</td>
</tr>
<tr>
<td>History</td>
<td>2</td>
<td>1 int, 1 enum</td>
</tr>
<tr>
<td>Latency Budget</td>
<td>1</td>
<td>int</td>
</tr>
<tr>
<td>Lifespan</td>
<td>1</td>
<td>int</td>
</tr>
<tr>
<td>Ownership</td>
<td>1</td>
<td>enum</td>
</tr>
<tr>
<td>Reader Data Lifecycle</td>
<td>2</td>
<td>2 ints</td>
</tr>
<tr>
<td>Reliability</td>
<td>2</td>
<td>1 int, 1 enum</td>
</tr>
<tr>
<td>Resource Limits</td>
<td>3</td>
<td>3 ints</td>
</tr>
<tr>
<td>Time Based Filter</td>
<td>1</td>
<td>int</td>
</tr>
<tr>
<td>User Data</td>
<td>1</td>
<td>string</td>
</tr>
<tr>
<td><strong>Data Reader Parameters</strong></td>
<td><strong>18</strong></td>
<td></td>
</tr>
</tbody>
</table>

Total parameters relevant to all MMS data readers & data writers: \(7 \times (25 + 18) = 301\)

We used DQML’s DBE interpreter to generate QoS settings files.
We are able to validate hypothesis (H1) comparing development of DBE interpreter to manual methods.

Manual effort:
- 301 lines
- 7 MMS data readers & writers

Interpreter effort:
- 160 lines

Breakeven point

Lines of Code

# of DDS Entities
(Data Readers, Data Writers, or Data Reader/Data Writer pair)
Related Publications & Presentations

**Book Chapters**

**Conference Publications**

**Poster Publications**
Presentation Outline

- Problem Statement
- Focus Area 1: QoS Configuration Management
- **Focus Area 2: Evaluation of QoS Mechanisms**
- Focus Area 3: QoS Support in Dynamic Environments
- Concluding Remarks

QoS Mechanism Space in Operating Environment A

- Adequate QoS mechanism
- Inadequate QoS mechanism

QoS Mechanism Space in Operating Environment B

- Adequate QoS mechanism
- Inadequate QoS mechanism
Focus Area 2: Evaluation of QoS Mechanisms

• A single pub/sub DRE system can be deployed in many different operating environments

• The QoS of a pub/sub DRE system is affected by the operating environment, e.g.,
  • # of senders/receivers
  • sending rate
  • pub/sub middleware

• Network transport protocols provide QoS for pub/sub DRE systems, e.g., reliability, latency, jitter
R&D Problem: Run-time QoS Guidance & Flexibility

Traditionally, transport protocols have been evaluated apart from pub/sub DRE QoS concerns while pub/sub middleware leverage a single protocol:

1. Trade-offs of protocols in various environments not highlighted
2. Impact of protocol properties & multiple pub/sub QoS concerns not known

![Diagram showing protocol comparison]
Motivating Example: Search & Rescue Missions

Scenario

- Regional disasters (e.g., hurricane, flooding)
- Survivors trapped
- Search & rescue (SAR) mission initiated
- Search application fuses multiple sensor streams
  - Thermal scans from unmanned aerial vehicles (UAVs)
  - Video from existing camera infrastructure
  - Data streams sent to datacenter for fusion
- Datacenter leverages DDS for coordination between search applications
Motivating Example: Search & Rescue Missions

Requirements

• Operate in diverse *ad hoc* environments
  • Support multiple missions & applications
  • Leverage resources available in the environment
  • Understand how environment affects QoS

• Support multiple “antagonistic” QoS
  • Soft real-time, multimedia
  • Reliability vs. low latency
Challenges for Evaluating QoS Mechanisms (1/2)

Challenge 1: Leverage new, custom transport protocols with middleware (e.g., to support multiple “antagonistic” QoS)

New, custom protocols can provide required QoS properties (e.g., balance reliability & low latency)

However, integration of protocols with middleware is complex
- Different APIs for different middleware
- No support for common transport functionality
  - Packet tracking
  - Multicasting
Challenge 2: Quantitatively evaluate multiple (e.g., “antagonistic”) QoS concerns for QoS-enabled pub/sub systems in multiple environments

Individual QoS concerns can be evaluated quantitatively. However, manually combining QoS evaluations is subjective & error-prone.
<table>
<thead>
<tr>
<th>Related Research</th>
</tr>
</thead>
</table>

Good for evaluating various pub/sub middleware technology *w/o regard to transport protocols*
## Evaluation of QoS Mechanisms: Related Research

### Related Research

<table>
<thead>
<tr>
<th>Reference</th>
<th>Details</th>
</tr>
</thead>
</table>

Good for **developing** customized protocols
# Evaluation of QoS Mechanisms: Related Research

<table>
<thead>
<tr>
<th>Related Research</th>
</tr>
</thead>
</table>

Good for evaluating **individual protocols**
### Properties for Evaluating QoS Mechanisms

What properties help us assess research to evaluate QoS mechanisms in various environments?

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Provide composite metrics for QoS</strong></td>
<td>Does the technique provide metrics to evaluate multiple QoS concerns?</td>
</tr>
<tr>
<td><strong>Evaluate QoS mechanisms empirically in pub/sub context</strong></td>
<td>Does the technique provide a pub/sub experimentation testbed for quantitative empirical evaluations to highlight trade-offs between QoS mechanisms?</td>
</tr>
<tr>
<td><strong>Provide flexible framework for QoS mechanisms</strong></td>
<td>Does the technique provide a flexible framework to include transport protocols with pub/sub middleware?</td>
</tr>
</tbody>
</table>
Is the technique used & evaluated in the context of pub/sub middleware?
Evaluation of QoS Mechanisms: Related Work

- Does the technique support empirical evaluations?
  - No empirical evaluations
  - Empirical evaluations

- Context
  - Standalone
  - Pub/sub middleware

- Transport Protocol
  - Static
  - Flexible protocol

- Framework
  - Balakrishnan
  - Floyd
  - Kirschberg
  - Sachs
  - Bellavista
  - Xiong
  - van Renesse

- Middlewares
  - Pub/sub
Evaluation of QoS Mechanisms: Related Work

Does the technique provide a flexible framework for transport protocols?

Pub/sub middleware context

Standalone context

van Renesse

Floyd

Kirschberg

Cheng

Bateman

Balakrishnan

Xiong

Bellavista

Sachs

Static transport protocol

Flexible protocol framework

No empirical evaluations

Empirical evaluations

Empirical evaluations

Does the technique provide a flexible framework for transport protocols?
Evaluation of QoS Mechanisms: Open Issues

Gap: Empirically evaluated pub/sub middleware with flexible protocol framework

Current gap makes it hard for developers to assess & leverage transport protocols to address pub/sub QoS concerns
Solution Approach: FLEXible Middleware & Transports

Solution: Flexible evaluation testbed for pub/sub middleware & transports

Pub/sub middleware context

Standalone context

Static transport protocol

Flexible protocol framework

No empirical evaluations

Empirical evaluations

FLEXMAT

van Renesse

Floyd

Xiong

Cheng

Kirschberg

Bellavista

Sachs

Balakrishnan

Floyd

Bellavista

Sachs
Addressing Challenges of Evaluating QoS Mechanisms (1/2)

FLEXMAT addresses challenge 1 of leveraging protocols by providing framework support for multiple pub/sub middleware platforms:

FLEXMAT’s Adaptive Network Transport (ANT) framework provides infrastructure support for protocols for fast incorporation:

- Multicast, mux/demux, sequencing support
- Flexible event-based subscription to supporting infrastructure

Integrated with the two most well-established open-source DDS implementations:

- PrismTech’s OpenSplice DDS implementation
- Object Computing, Inc.’s OpenDDS implementation
Addressing Challenges of Evaluating QoS Mechanisms (2/2)

FLEXMAT addresses challenge 2 by providing an experimentation framework & composite metrics that quantitatively evaluate multiple QoS concerns

Composite QoS metrics

- **ReLate2 metric**
  - Includes reliability & latency
  - Sets 10% loss as unacceptable (i.e., increases by order of magnitude)

- Other ReLate2 metrics add jitter, network usage
  - \( ReLate2_{\text{Jit}} = ReLate2 \times \text{std deviation} \)
  - \( ReLate2_{\text{Net}} = ReLate2 \times \text{avg b/w usage} \)

FLEXMAT supports the SAR requirements of
(1) Supporting multiple “antagonistic” QoS via new, custom protocols &
(2) Understanding how environments affect multiple QoS concerns
**Comparison: Evaluation of QoS Mechanisms Properties**

<table>
<thead>
<tr>
<th>Technique</th>
<th>Pub/Sub Middleware Evaluation</th>
<th>Flexible Protocol Frameworks</th>
<th>Transport Protocol Evaluations</th>
<th>Composite Metrics</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>FLEXMAT</strong></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Xiong, M.</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sachs, K.</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Floyd, S.</td>
<td></td>
<td></td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>van Renesse, R.</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Balakrishnan, M.</td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Kirschberg, J.</td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
</tbody>
</table>

**Current limitation of FLEXMAT:**

- Can evaluate QoS mechanisms in diverse environments but can not support modifying the mechanisms dynamically as the environment is changing
- Addressed by focus area 3 (proposed research)
FLEXMAT Analysis

Evaluate Ricochet protocol with multiple operating environments using FLEXMAT testbed:
- OpenSplice, OpenDDS
- Various # of senders, % loss, sending rate

Two approaches for integrating Ricochet with DDS implementations
- Integrate directly into DDS implementations
FLEXMAT Analysis

Evaluate Ricochet protocol with multiple operating environments using FLEXMAT testbed:

- OpenSplice, OpenDDS
- Various # of senders, % loss, sending rate

Two approaches for integrating Ricochet with DDS implementations

- Integrate directly into DDS implementations
- Leverage FLEXMAT testbed integrated with DDS implementations

Two approaches for evaluating multiple QoS concerns

- Manually compose & integrate results
Evaluate Ricochet protocol with multiple operating environments using FLEXMAT testbed:

- OpenSplice, OpenDDS
- Various # of senders, % loss, sending rate

Two approaches for integrating Ricochet with DDS implementations

- Integrate directly into DDS implementations
- Leverage FLEXMAT testbed integrated with DDS implementations

Two approaches for evaluating multiple QoS concerns

- Manually compose & integrate results
- Leverage composite QoS metrics
## Analysis: FLEXMAT & Protocol Integration (1/2)

<table>
<thead>
<tr>
<th>Evaluation Criteria</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>(H1) Reducing Integration Effort</strong></td>
<td>Hypothesize that leveraging FLEXMAT will reduce Ricochet integration effort with OpenDDS &amp; OpenSplice by &gt; 50%</td>
</tr>
</tbody>
</table>

### Effort Analysis

- **Effort for Ricochet**: \( \frac{\text{Effort for direct integrations}}{2} \)
- **Effort for FLEXMAT Integration**: \( \frac{\text{Effort for Ricochet direct integrations}}{2} \)
- **Total Effort**: \( 2 \)

![Diagram of FLEXMAT and Protocol Integration](image)
Analysis: FLEXMAT & Protocol Integration (2/2)

We are able to validate hypothesis (H1) comparing direct integration of protocol vs. integration leveraging FLEXMAT

<table>
<thead>
<tr>
<th>Integration</th>
<th>C++ Statements</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ricochet OpenDDS</td>
<td>733</td>
</tr>
<tr>
<td>Ricochet OpenSplice</td>
<td>+ 353</td>
</tr>
<tr>
<td><strong>Total effort</strong></td>
<td>1086</td>
</tr>
<tr>
<td>Ricochet FLEXMAT</td>
<td>83</td>
</tr>
</tbody>
</table>

Integration effort

- \((1086 - 83) \div 1086 = 92\%\) reduction leveraging FLEXMAT
- Ricochet integration with FLEXMAT leverages protocol support (e.g., mux/demux, sequencing, multicast)
(H2) Reducing QoS Evaluation Comparisons

Hypothesize that using ReLate2 composite metric will reduce QoS evaluation comparisons by 50%.

Experiment environment for generating comparison results:
- Emulab 850 MHz processors
- Fedora Core 6 w/ RT 11 patches
- OpenDDS 1.2.1
- 1 data writer, 3 data readers
- 12 byte data payloads, 20K packets sent
- 1% network loss at endhosts
- 25 Hz sending rate
- Ricochet parameter values, R=4,8, C=3
- NAKcast timeout values (0.05, 0.025 s)
Protocols good for reliability

Protocols good for reliability & low latency

Protocols good for low latency
We are able to validate hypothesis (H2) evaluating # of comparison points w/ & w/o ReLate2 metric.

- **80 comparisons needed, requires subjectivity**
- **40 comparisons needed, maintains objectivity**
Relevant Publications & Presentations

Conference Publications


Workshop Publications


Presentation Outline

- Problem Statement
- Focus Area 1: QoS Configuration Management
- Focus Area 2: Evaluation of QoS Mechanisms
- Focus Area 3: QoS Support in Dynamic Environments
- Concluding Remarks

System Execution Timeline

QoS Mechanism 1 (adequate)  
(initial environment)

QoS Mechanism 2 (adequate)  
(environment modification)

QoS Mechanism 3 (adequate)  
(environment modification)

QoS Mechanism 2 (inadequate)

QoS Mechanism 1 (inadequate)
Focus Area 3: QoS Support in Dynamic Environments

These areas have been addressed by my research results described earlier.
Focus Area 3: QoS Support in Dynamic Environments

This area has not yet been addressed & is what I propose to do to complete my dissertation
Focus Area #3: QoS Support in Dynamic Environments

- As the operating environment changes QoS mechanisms might no longer be effective
- QoS might no longer be maintained
- Building on FLEXMAT we’re focusing on the QoS mechanisms of transport protocols

Transport protocol 1
(adequate)

Transport protocol 1
(inadequate)

(initial environment)  

System Execution Timeline  

(environment modification)
R&D Problem: Manage QoS in Dynamic Environments

As environment changes & QoS not being met:

- Select an appropriate QoS mechanism to support QoS
- Select the mechanism in a timely manner
  - Manual modification tedious, error-prone, & untimely for DRE systems

System Execution Timeline

Transport protocol 1 (adequate)

(Initial environment)

Transport protocol 2 (adequate)

(Transport protocol 2 (inadequate))

Transport protocol 3 (adequate)

(Transport protocol 2 (inadequate))

Transport protocol 1 (inadequate)

(Transport protocol 1 (inadequate))

(System Execution Timeline)
Motivating Example: Ambient Assisted Living (AAL)

Scenario

- Aging population increasing, # of health care workers decreasing
Motivating Example: Ambient Assisted Living (AAL)

Scenario

• Aging population increasing, # of health care workers decreasing
• Increase elderly autonomy in urban areas via coordination of personal equipment & sensing/aware “smart cities”

- Mobility
  - Videocamera
  - PDA
  - High-resolution health monitoring
- Sensory Enhancing Equipment
- Cell phone
- GPS
- Law enforcement
- Doctors
- Surveillance infrastructure
  - Healthcare facilities
  - Smart City
  - EMS
  - Firefighters
Motivating Example: Ambient Assisted Living (AAL)

Scenario

- Aging population increasing, # of health care workers decreasing
- Increase elderly autonomy in urban areas via coordination of personal equipment & sensing/aware “smart cities”
- Utilize personal data center to manage personal & environment data
Motivating Example: Ambient Assisted Living (cont.)

Requirements

• Operate in dynamic environment, e.g.,
  • Varying # of senders, receivers
  • Varying network bandwidth, loss
  • Varying data sending rates (e.g., more updates for critical data)

• Support QoS as environment changes
  • Reliability & latency
    • e.g., high resolution health monitoring
  • Multimedia data
Challenges for QoS in Dynamic Environments (1/2)

**Challenge 1: Determine appropriate transport protocol adjustment as environment changes**

Changes in operating environment can make a once-adequate QoS mechanism inadequate.

- **Low sending rate**: Fewer cameras in vicinity
  - Transport protocol 1 (adequate)
- **High sending rate**: More cameras in vicinity
  - Transport protocol 1 (inadequate)

Elderly person traveling through smart city
Challenges for QoS in Dynamic Environments (2/2)

Challenge 2: Make adjustments in a timely manner

Timeliness concerns need to be addressed while selecting an adequate QoS mechanism.

- More cameras close by
- High sending rate
- Transport protocol 1 (inadequate)
- Transport protocol 1 (adequate)
### Related Research

<table>
<thead>
<tr>
<th>Topic</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>PrismTech’s Tuner application</td>
<td><a href="http://www.opensplice.com">http://www.opensplice.com</a></td>
</tr>
<tr>
<td>Xiangping Bu et al.</td>
<td>“A Reinforcement Learning Approach to Online Web Systems Auto-configuration”, <em>29th IEEE International Conference on Distributed Computing Systems</em>, June 2009, Montreal, Canada</td>
</tr>
</tbody>
</table>

Good for *manually checking* the run-time QoS status
### Related Research

<table>
<thead>
<tr>
<th>Related Research</th>
</tr>
</thead>
<tbody>
<tr>
<td>PrismTech’s Tuner application, <a href="http://www.opensplice.com">http://www.opensplice.com</a></td>
</tr>
<tr>
<td>Xiangping Bu et al. &quot;A Reinforcement Learning Approach to Online Web Systems Auto-configuration“, <em>29th IEEE International Conference on Distributed Computing Systems</em>, June 2009, Montreal, Canada</td>
</tr>
</tbody>
</table>

Good for developing adaptation applications
### Related Research

<table>
<thead>
<tr>
<th>Research Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>PrismTech’s Tuner application, <a href="http://www.opensplice.com">http://www.opensplice.com</a></td>
</tr>
<tr>
<td>Xiangping Bu et al. &quot;A Reinforcement Learning Approach to Online Web Systems Auto-configuration&quot;, <em>29th IEEE International Conference on Distributed Computing Systems</em>, June 2009, Montreal, Canada</td>
</tr>
</tbody>
</table>

**Good for adaptation when timeliness is not a driving concern**
What properties help us assess research to support QoS in dynamic environments?

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Monitor QoS</strong></td>
<td>Does the technique know when QoS is no longer being met?</td>
</tr>
<tr>
<td><strong>Online reconfiguration</strong></td>
<td>Does the technique perform configuration adjustments while system is running?</td>
</tr>
<tr>
<td><strong>Timely adaptation</strong></td>
<td>Can the technique change to a more appropriate protocol in a timely manner?</td>
</tr>
</tbody>
</table>
QoS in Dynamic Environments: Related Work

Does the technique monitor the system for meeting QoS?

Monitor QoS

No monitoring

Timely transition

Best-effort/no transition

Static configuration

Online reconfiguration

RTI
PrismTech

Bu
Violle
Herssens

Grace
David
QoS in Dynamic Environments: Related Work

Does the technique provide timely transition to support QoS?

Monitor QoS

No monitoring

Timely transition

Best-effort/no transition

Static configuration

Online reconfiguration

PrismTech

Bu Vienne Herssens

RTI

Grace David
QoS in Dynamic Environments: Related Work

Does the technique reconfigure while the system is running?

- Monitor QoS
- No monitoring

- Timely transition
- Best-effort/no transition

- Static configuration
- Online reconfiguration

PrismTech

Bu Vienne Herssens

Grace David
QoS in Dynamic Environments: Open Issues

Monitor QoS

No monitoring

Gap: Monitor, analyze, & adapt in a timely manner

Timely reconfiguration

Best-effort/no transition

Static configuration

PrismTech

Bu Vienne Herssens

RTI

Grace David

Current gap makes it hard for DRE systems in dynamic environments to support QoS in a timely manner.
Proposed Approach: Adaptive M/W & Network Transports

Proposed Solution Approach:
Timely autonomic transport protocol adaptation for QoS-enabled pub/sub middleware
ADAMANT proposes to address challenge 1 by using machine learning to determine appropriate transport protocol adjustment.

Machine learning reduces accidental complexity of manually determining response to operating environment factors.

Machine learning provides robustness over manual methods for operating environments not previously known or encountered.

```cpp
if (network_loss_percent == 1 && num_receivers < 5 && sending_rate < 0.01) {
    transport_framework->use (transport1);
} else if (network_loss_percent == 5 && num_receivers < 5 && sending_rate < 0.01) {
    transport_framework->use (transport2);
} else if (network_loss_percent == 10 && num_receivers < 5 && sending_rate < 0.01) {
    transport_framework->use (transport3);
} else if (network_loss_percent == 1 && num_receivers >= 5 && num_receivers < 10 && sending_rate < 0.01) {
    transport_framework->use (transport4);
} else if (network_loss_percent == 5 && num_receivers >= 5 && num_receivers < 10 && sending_rate < 0.01) {
    transport_framework->use (transport2);
} else if (network_loss_percent == 10 && num_receivers >= 5 && num_receivers < 10 && sending_rate < 0.01) {
    transport_framework->use (transport5);
}
```
ADAMANT proposes to support the AAL requirements to
(1) Determine appropriate protocol adjustment as environment varies &
(2) Make adjustments in a timely manner

ADAMANT proposes to address challenge 2 by using supervised
machine learning to make adjustments in bounded time

Supervised machine learning can provide boundedness over unsupervised
machine learning:

Unsupervised learning traversing solution space
Time complexity: dependent on solution space

Artificial neural network with preconfigured # of layers, nodes
Time complexity: dependent on preconfigured # of layers, nodes
Proposed Experiment: ADAMANT & Dynamic Environments

<table>
<thead>
<tr>
<th>Evaluation Criteria</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>(H1) Adjust for known environment</td>
<td>Hypothesize that ADAMANT will provide adjustment improvement for known environments at least 85% of the time</td>
</tr>
</tbody>
</table>

Leverage FLEXMAT to determine QoS properties of particular transport protocol & settings

Generate training data from collected metrics

Train ADAMANT’s machine learning on training data

Run ADAMANT’s machine learning against training data, collect & measure results
Provisioned Experiment: ADAMANT & Dynamic Environments

<table>
<thead>
<tr>
<th>Evaluation Criteria</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>(H2) Adjust for unknown environment</td>
<td>Hypothesize that ADAMANT will provide adjustment improvement for unknown environments more than 50% of the time</td>
</tr>
</tbody>
</table>

Leverage FLEXMAT to determine QoS properties of particular transport protocol & settings

Split data from FLEXMAT into training & testing data

Train ADAMANT’s machine learning on training data subset

Run ADAMANT’s machine learning against testing data, collect & measure results
**Proposed Experiment: ADAMANT & Dynamic Environments**

<table>
<thead>
<tr>
<th>Evaluation Criteria</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>(H3) Provide bounded, constant time adaptation</td>
<td>Hypothesize that ADAMANT will adjust to new operating environment in bounded constant time (i.e., O(1))</td>
</tr>
</tbody>
</table>

Perform time complexity analysis for ADAMANT

- Determine worst case time complexity for calculating adjustment (i.e., for machine learning)
- Determine worst case time complexity for executing adjustment

Collect empirical results & validate analytical results

- Leverage Emulab/ISISlab testbeds
- Instrument code to determine timings

```c
// Load the configuration values.
amt_load_config_values (amt_network);

// Create the ANT Interface.
amt_network->amt_ = RiscohetIF::instance ();

// Initialize the ANT interface.
amt_network->amt_ = init (amt_network->protocol_,
MULTICAST_MODULE_PORT,
amt_network->drop_ratio_,
amt_network->r_value_,
amt_network->c_value_,
amt_network->nak_timeout_in_sec_);
```
1. Evaluate machine learning techniques for utility, accuracy, & timeliness

2. Collect additional training data

3. Develop monitoring infrastructure

4. Develop controller; evaluate timeliness

5. Integrate monitoring, controller, & machine learning

6. Collect metrics & evaluate

**ADAMANT Remaining Research Timeline**

- **Protocol Optimizer**: (Machine Learning)
- **DDS**
- **ANT Protocols**
- **Additional Training Data**

Timeline:
- November 2009
- June 2010
- December 2010
- April 2011
Relevant Publications & Presentations

Conference Publications

Workshop Publications
Presentation Outline

• Problem Statement
• Focus Area 1: QoS Configuration Management
• Focus Area 2: Evaluation of QoS Mechanisms
• Focus Area 3: QoS Support in Dynamic Environments
• Concluding Remarks
## Doctoral Research Contributions

### Enhancing Productivity & Flexibility for QoS-enabled Pub/Sub DRE Systems

<table>
<thead>
<tr>
<th>Focus Area</th>
<th>Challenge</th>
<th>Approach</th>
<th>Contribution</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Valid QoS Design</strong></td>
<td>• Design-time QoS Validation</td>
<td>• DSML that validates QoS configuration &amp; generates implementation artifacts</td>
<td>• DQML</td>
</tr>
</tbody>
</table>
| **Evaluation of QoS Mechanisms** | • Run-time QoS Guidance & Flexibility          | • Pub/sub middleware with flexible protocol framework  
• Composite metrics & empirical analysis | • FLEXMAT |
| **Autonomic Adapation for QoS** | • Manage QoS in Dynamic Environments | • Autonomic adaptation of transport protocols  
• Timely adaptation based on supervised learning | • ADAMANT |

[www.dre.vanderbilt.edu/~jhoffert/research](http://www.dre.vanderbilt.edu/~jhoffert/research)
Summary of Publications & Presentations

Conference Publications


Workshop Publications


Summary of Publications & Presentations

**Book Chapters**


**Poster Publications**


Questions

Soli Deo Gloria