MicroQoSCORBA: A Configurable Middleware Framework for Small Embedded Systems that Supports Multiple Quality of Service Properties

Dave Bakken
Oregon Graduate Institute, 26 September 2003

MicroQoSCORBA Research Team

Professors: Dave Bakken (lead PI), John Shovic (adjunct)
Students: A. David McKinnon, Kevin Dorow, Tarana Damania, Olav Haugan, Wes Lawrence, Thor Skaug, Eivind Næss, Kim Swenson, Ryan Johnston

This research is supported in part by two Cisco University Research Program Donations and grant NSF-CISE EHS-0209211 from the National Science Foundation’s Embedded and Hybrid Systems Program.
Outline

• **Introduction**
  – Middleware
  – Embedded systems
  – Quality of Service

• Related Work

• **MicroQoSCORBA**
  – Middleware Architectural Design Taxonomy
  – Fine-grained Configurable Middleware Framework
  – Embedded System Security
  – Security Subsystem Design & Implementation
  – Experimental Evaluation

• Conclusions
Middleware

“A layer of software above the operating system but below the application program that provides a common programming abstraction across a distributed system”
Middleware: Heterogeneity & Transparency

- Middleware’s programming building blocks mask heterogeneity
  - Makes programmer’s life much easier!!
- Kinds of heterogeneity masked by middleware
  - Heterogeneity in network technology always masked
  - Heterogeneity in host CPU always masked
  - Heterogeneity in operating system (or family thereof) usually masked
  - Heterogeneity in programming language usually masked
  - Heterogeneity in vendor implementations sometimes masked
- Middleware can provide transparency with respect to distribution:
  - Location transparency
  - Replication transparency
  - Mobility transparency
  - Concurrency transparency
  - Failure transparency
- Masking heterogeneity and providing transparency makes programming distributed systems much easier to do!
Existing Middleware Frameworks

• Support is lacking for
  – Small memory footprint
  – Generality to a wide range of hardware devices
  – Power awareness
  – Multi-property QoS (esp. non-RT properties)
  – Fine-grained configurability
  – Software Engineering & Analysis tools
Embedded Systems Market

• 11 Billion CPUs per year
• System size varies
  – Aircraft, PDAs, Home appliances
• Application volume varies
  – Radios, TVs, Satellites
• Application constraints vary
  – Business applications
  – Sensor Networks
    • Single-purpose sensors → High-end signal processing
    • Environmental monitoring, battlefield networks, …
Quality of Service

• Real-World applications have real-world tradeoffs!

• QoS Properties
  – Security
    • Multiple strengths/Algorithms
  – Fault Tolerance
    • Quantity and types of faults tolerated
  – Real-time Behavior
    • Scheduling algorithms, Network performance

• Resource Issues
  – Memory footprint
  – Power awareness
Outline

• Introduction

• **Related Work**

• **MicroQoSCORBA**
  – Middleware Architectural Design Taxonomy
  – Fine-grained Configurable Middleware Framework
  – Embedded System Security
  – Security Subsystem Design & Implementation
  – Experimental Evaluation

• Conclusions
Related Work—CORBA

• MinimumCORBA
  – Removes support for dynamic interfaces, etc
  – Reduces the memory footprint of an ORB (~in half)
• Real-time CORBA
  – Provides tools to better predict time delays
  – Enables hard real-time CORBA applications
• Fault Tolerant CORBA
• CORBA Security Service
• Smart Transducers Interface
• Stand-alone specifications—they do not compose
Related Work, cont.

• Java Remote Method Invocation (RMI)
  – Lacks cross-language support, configurability, QoS mechanisms

• Small Footprint
  – legORB — UIUC, ~ 6 Kb Client IIOP engine
  – e*ORB — Vertel, ~ 35 Kb Client ORB
  – ORBlite — HP Labs: evolvability flexibility, subsetting
  – Stripped-down, *Point solutions*

• VEST
  – Application specific operating systems
  – Lightweight components
  – Strong analysis toolkit
  – Aspects for RT performance & dependability
Related Work, cont.

• MMLite
  – object oriented DCOM components (coarse grained)
  – primary QoS property is RT via scheduling comp.

• QoS and/or Reflection
  – QuO — QoS contracts, adaptive distributed applications
  – MULTE — wide range of latency & bandwidth
  – Open-ORB Python — reflection, component based
  – dynamicTAO — dynamic adaptation, replaceable concurrency, scheduling, & security
Outline

• Introduction
• Related Work
• MicroQoSCORBA
  – Middleware Architectural Design Taxonomy
  – Fine-grained Configurable Middleware Framework
  – Embedded System Security
  – Security Subsystem Design & Implementation
  – Experimental Evaluation
• Conclusions
MicroQoSCORBA Objectives

• Support wide ranges of (deeply) embedded H/W
  – Resources vary widely (memory, power, etc.)
  – Home appliances, Sensor networks

• Tailor middleware to both application and hardware constraints, with fine granularity

• Develop a multi-property QoS enabled MW framework

• Maintain CORBA interoperability
  – Develop an IDL based framework that interoperates with other ORBs, rather than just another IIOP engine
## Lifecycle Time Epochs

<table>
<thead>
<tr>
<th>Lifecycle Epoch</th>
<th>Constraint Bound</th>
<th>Representative Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Design</strong></td>
<td>HW Heterogeneity</td>
<td>Symmetric, asymmetric</td>
</tr>
<tr>
<td></td>
<td>HW Choice</td>
<td>X86, TINI, ColdFire</td>
</tr>
<tr>
<td></td>
<td>Communications HW</td>
<td>Ethernet, Serial, Infrared</td>
</tr>
<tr>
<td></td>
<td>Processing Capability</td>
<td>50 Mhz, 1 GHz, 8bit, 32bit</td>
</tr>
<tr>
<td></td>
<td>System size</td>
<td>small, medium, large (e.g., transducers to jets)</td>
</tr>
<tr>
<td></td>
<td>Power Usage</td>
<td>line, battery, and/or parasitic power</td>
</tr>
<tr>
<td><strong>IDL Compilation</strong></td>
<td>Communication Style</td>
<td>Passive, Pro-active, Push, Pull</td>
</tr>
<tr>
<td></td>
<td>Stub/Proxy Generation</td>
<td>Inline vs. library usage</td>
</tr>
<tr>
<td></td>
<td>Message Lengths</td>
<td>Fixed, variable length messages</td>
</tr>
<tr>
<td></td>
<td>Parameter Marshalling</td>
<td>Fixed Formats</td>
</tr>
<tr>
<td><strong>Application</strong></td>
<td>Space/Time Optimizations</td>
<td>Loop unrolling, code migration, function and proxy inlining</td>
</tr>
<tr>
<td></td>
<td>Library Usage</td>
<td>Static vs. dynamic library linkage</td>
</tr>
<tr>
<td><strong>Compilation</strong></td>
<td>Device Initialization</td>
<td>Serial port baud rate, handshaking</td>
</tr>
<tr>
<td></td>
<td>Network Startup</td>
<td>Bootp, dhcp</td>
</tr>
<tr>
<td></td>
<td>Major QoS adaptation</td>
<td>Select between QoS modules</td>
</tr>
<tr>
<td><strong>Run Time</strong></td>
<td>Minor QoS adaptation</td>
<td>Adjust QoS parameters</td>
</tr>
</tbody>
</table>

Dave Bakken
## Middleware Architectural Taxonomy

<table>
<thead>
<tr>
<th>Embedded Hardware</th>
<th>Roles</th>
<th>SW I/O</th>
<th>IDL Subsetting</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>System Comp.</strong></td>
<td><strong>Control Flow</strong></td>
<td><strong>Data Flow</strong></td>
<td><strong>Interaction Style</strong></td>
</tr>
<tr>
<td>Homogenous</td>
<td>Connection Setup</td>
<td>Data Direction</td>
<td>Sync (Send/Recv)</td>
</tr>
<tr>
<td>Asymmetric</td>
<td>Initiates Setup</td>
<td>Bits In</td>
<td>Async (One-Way Msgs)</td>
</tr>
<tr>
<td></td>
<td>Receive Setup</td>
<td>Bits Out</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Requests</td>
<td>Bits In/Out</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Service Location</td>
<td>1 Message in Transit</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Hardwired Logic</td>
<td>N Messages in Transit</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Config. File</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Name Service</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Other</td>
<td></td>
<td></td>
</tr>
<tr>
<td>HW I/O Support</td>
<td><strong>Parallelism</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Serial, 1-Wire,</td>
<td>1 Message in</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Parallel, Digital,</td>
<td>Transit</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Ethernet, IrDA,</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Bluetooth, GSM, GPRS</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Resources</td>
<td><strong>Data Direction</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Memory</td>
<td>Bits In</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Power</td>
<td>Bits Out</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Processing</td>
<td>Bits In/Out</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Capabilities</td>
<td>1 Message in</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Transit</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>N Messages in</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Transit</td>
<td></td>
<td></td>
</tr>
<tr>
<td>System Comp.</td>
<td><strong>Service Location</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Homogenous</td>
<td>Hardwired Logic</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Asymmetric</td>
<td>Config. File</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Name Service</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Other</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Resources</td>
<td><strong>Interaction Style</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Memory</td>
<td>Bits In</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Power</td>
<td>Bits Out</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Processing</td>
<td>Bits In/Out</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Capabilities</td>
<td>1 Message in</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Transit</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>N Messages in</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Transit</td>
<td></td>
<td></td>
</tr>
<tr>
<td>System Comp.</td>
<td><strong>SW I/O</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Homogenous</td>
<td>Data Representation</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Asymmetric</td>
<td>CORBA CDR</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>MQC CDR</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>...</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Resources</td>
<td><strong>Message Payload</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Memory</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Power</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Processing</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Capabilities</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Outline

• Introduction
• Related Work
• MicroQoS CORBA
  – Middleware Architectural Design Taxonomy
  – Fine-grained Configurable Middleware Framework
  – Embedded System Security
  – Security Subsystem Design & Implementation
  – Experimental Evaluation
• Conclusions
Architectural Considerations

• Fine-grained composability
• Baseline Functionality
  – CORBA IDL support
  – Autogenerated Code (e.g., stubs & skeletons)
• QoS Functionality
  – Security (to be presented later)
  – Fault Tolerance
  – Timeliness
• Development Tools
## Fault Tolerance Mechanisms

<table>
<thead>
<tr>
<th>Redundancy</th>
<th>Reliability</th>
<th>Ordering</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Temporal</td>
<td>• Group Communication</td>
<td>• Sender FIFO</td>
</tr>
<tr>
<td>– Multiple</td>
<td>– Best Effort</td>
<td>• Causal</td>
</tr>
<tr>
<td>transmits</td>
<td>– Reliable</td>
<td>– Logical Timestamping</td>
</tr>
<tr>
<td>• Spatial</td>
<td>– Atomic</td>
<td>• Total</td>
</tr>
<tr>
<td>– Multiple</td>
<td>– Uniform</td>
<td>– Sequencer / Token based</td>
</tr>
<tr>
<td>Channels</td>
<td></td>
<td></td>
</tr>
<tr>
<td>– Replicated</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Servers</td>
<td></td>
<td></td>
</tr>
<tr>
<td>• Value</td>
<td>• Failure Detection</td>
<td></td>
</tr>
<tr>
<td>– Checksums,</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CRC</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Ordering:
- **Sender FIFO**
- **Causal**
  - Logical Timestamping
- **Total**
  - Sequencer / Token based
Development Tools

- Not all developers are created equal
- **Goal:** Make it easy for the casual programmer
  - Domain expert, but QoS novice
  - Lifecycle support personnel
  - Temporary/contract employees
- Tools choose compatible components based upon
  - QoS requirements
  - Resource configuration
- Application and Hardware specific configuration file for the IDL Compiler
  - IDL compiler custom-generates stub/skeleton code
Architecture Overview
Outline

• Introduction
• Related Work
• MicroQoSCORBA
  – Middleware Architectural Design Taxonomy
  – Fine-grained Configurable Middleware Framework
  – **Embedded System Security**
  – Security Subsystem Design & Implementation
  – Experimental Evaluation
• Conclusions
Embedded System Security

- Security must be designed in
- Deep coupling with a physical environment
  - Exposure to the elements, tampering, etc.
- Significant tradeoffs between
  - Security
  - Cost & Resource Usage
  - Generality / Adaptability
- Relatively limited computation power
- Denial of Service attacks are more acute
## Security Design Space

<table>
<thead>
<tr>
<th>Confidentiality</th>
<th>Integrity</th>
<th>Availability</th>
<th>Accountability</th>
</tr>
</thead>
<tbody>
<tr>
<td>Physical</td>
<td>Message Digests*</td>
<td>Service Continuity</td>
<td>Authentication</td>
</tr>
<tr>
<td>— Dedicated Wire</td>
<td>— MD4/5</td>
<td>— See Fault Tolerance</td>
<td>— Physical Tokens</td>
</tr>
<tr>
<td>— Secure Network</td>
<td>— SHA</td>
<td>Disaster Recovery</td>
<td>— Shared Secrets*</td>
</tr>
<tr>
<td>Encryption</td>
<td>Message Authentication Codes*</td>
<td></td>
<td>— Passwords</td>
</tr>
<tr>
<td>— Symmetric*</td>
<td>— HMAC</td>
<td></td>
<td>— Challenge/Resp.</td>
</tr>
<tr>
<td>AES</td>
<td>Error Control/Correction Codes</td>
<td></td>
<td>Authorization</td>
</tr>
<tr>
<td>DES</td>
<td>— CRC32*</td>
<td></td>
<td>— Access Controls</td>
</tr>
<tr>
<td>Rot13</td>
<td>Digital Signatures</td>
<td></td>
<td>— Data Protection</td>
</tr>
<tr>
<td>— Public Key</td>
<td>— DSA</td>
<td></td>
<td>Audit</td>
</tr>
<tr>
<td>RSA</td>
<td>— RSA</td>
<td></td>
<td>Non-Repudiation</td>
</tr>
<tr>
<td>Elliptic Curves</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Some aspects of this design space are beyond the scope of MicroQoSCORBA (e.g., Dedicated networks, authentication tokens, PKI infrastructure)

* Currently Implemented
Outline

• Introduction

• Related Work

• MicroQoSCORBA
  – Middleware Architectural Design Taxonomy
  – Fine-grained Configurable Middleware Framework
  – Embedded System Security
  – Security Subsystem Design & Implementation
  – Experimental Evaluation

• Conclusions
Configurable Security Subsystem

• Initial Prototype
  – Caesar & AES Ciphers

• Implementing additional mechanisms
  – Reused existing cryptographic mechanisms
    • Cryptix Java Cryptography Extensions (JCE) mechanisms
    • Substantially rewrote the Cryptix JCE class hierarchy
  – Implemented “low-cost” mechanisms
    • XOR cipher
    • Parity & CRC message digests

• Security mechanisms are enabled/disabled via MicroQoSCORBA’s macro mechanisms
MicroQoSCORBA Security Mechanisms

• Supported Ciphers
  – XOR, Caesar, CAST5, DES, 3DES, IDEA, MARS, RC2, RC4, AES, Serpent, SKIPJACK, Square, Twofish

• Support Message Digests
  – Parity, CRC32, MD2, MD4, MD5, RIPEMD, RIPEMD128, RIPEMD160, SHA0, SHA1, SHA256, SHA384, SHA512, Tiger

• Supported Message Authentication Codes
  – HMAC is supported with the above MDs
Outline

• Introduction
• Related Work
• MicroQoSCORBA
  – Middleware Architectural Design Taxonomy
  – Fine-grained Configurable Middleware Framework
  – Embedded System Security
  – Security Subsystem Design & Implementation
  – Experimental Evaluation
• Conclusions
Supported Platforms

• Linux Workstation
  – Pentium 4, 1.5 GHz, 256 MB RAM

• Systronix SaJe
  – 100 MHz aJile aJ-100 CPU, 1 MB RAM
  – Native Java execution

• TINI
  – 40 MHz DS80C390 CPU, 512 KB RAM
  – (~equiv. 100 MHz 8051)
  – Emulated JVM (slow)

• PDAs (soon)
Automated Tools

• Necessitated by MicroQoSCORBA’s fine-grained configurability of both functional and QoS properties
  – (i.e., literally *hundreds* of configurations were evaluated)

• Complex Makefile targets
  – Update configurations, Execute IDL compiler, Build configurations, Archive builds for later execution

• Expect scripts
  – Automate the performance testing of multiple configurations
  – Scripts developed for the Linux, SaJe, and TINI platforms

• Analysis Routines
  – Common file formats, Expect logs
Example Application

• Timing Example
  – Very simple
  – Note: MicroQoSCORBA has not been completely optimized for memory usage or run time performance

• CORBA IDL

```idl
module timing {
  interface foo {
    long bar(in long arg1);
  };
};
```
Memory & File Size Comparisons

- MicroQoSCORBA and JacORB are Java based
- TAO is a C++ ORB

<table>
<thead>
<tr>
<th>Sizes (on Linux)</th>
<th>MicroQoSCORBA</th>
<th>JacORB</th>
<th>TAO</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Client</td>
<td>Server</td>
<td>Client</td>
</tr>
<tr>
<td>Application</td>
<td>4,222 B</td>
<td>2,476 B</td>
<td>6,591 B</td>
</tr>
<tr>
<td>Java Mem.</td>
<td>153,560 B</td>
<td>160,648 B</td>
<td>222,968 B</td>
</tr>
<tr>
<td>Linux RSS</td>
<td>9.95 MB</td>
<td>9.62 MB</td>
<td>13.31 MB</td>
</tr>
</tbody>
</table>

Baseline Application Size & Memory Usage

<table>
<thead>
<tr>
<th>MicroQoSCORBA Java Class Size (bytes)</th>
<th>Linux Client</th>
<th>Linux Server</th>
<th>SaJe Client¹</th>
<th>SaJe Server¹</th>
<th>TINI Client</th>
<th>TINI Server</th>
</tr>
</thead>
<tbody>
<tr>
<td>Baseline</td>
<td>63,607</td>
<td>61,062</td>
<td>259,077</td>
<td>254,246</td>
<td>23,867</td>
<td>20,764</td>
</tr>
<tr>
<td>Baseline w/Temp.Red.²</td>
<td>59,478</td>
<td>58,617</td>
<td>258,437</td>
<td>252,819</td>
<td>22,506</td>
<td>19,928</td>
</tr>
<tr>
<td>Value Redundancy</td>
<td>68,687</td>
<td>66,278</td>
<td>262,506</td>
<td>257,726</td>
<td>25,726</td>
<td>22,675</td>
</tr>
<tr>
<td>AES Cipher</td>
<td>85,182</td>
<td>82,634</td>
<td>270,606</td>
<td>265,762</td>
<td>35,156</td>
<td>32,062</td>
</tr>
</tbody>
</table>

Multi-Property QoS Application Size (bytes)

¹ Note: SaJe sizes include runtime  
² Note: impl. restricts to fixed length msgs so simpler code

Dave Bakken

MicroQoSCORBA — 31
## Timing Example Latency

### Baseline (non-QoS) Timing Latency (ms)

<table>
<thead>
<tr>
<th>QoS Property</th>
<th>Linux</th>
<th>SaJe</th>
<th>TINI</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>No TR</td>
<td>TR2</td>
<td>TR4</td>
</tr>
<tr>
<td>Baseline</td>
<td>0.170</td>
<td>0.184</td>
<td>0.182</td>
</tr>
<tr>
<td>Value Red.</td>
<td>0.178</td>
<td>0.194</td>
<td>0.195</td>
</tr>
<tr>
<td>AES-128</td>
<td>0.207</td>
<td>0.229</td>
<td>0.222</td>
</tr>
<tr>
<td>AES-192</td>
<td>0.202</td>
<td>0.222</td>
<td>0.222</td>
</tr>
<tr>
<td>AES-256</td>
<td>0.219</td>
<td>0.227</td>
<td>0.229</td>
</tr>
</tbody>
</table>

### Multi-Property QoS Timing Latencies (ms)

Results from best of three runs; TR=Temporal Redundancy
## Security Performance

<table>
<thead>
<tr>
<th>Security Mechanism</th>
<th>Linux 56</th>
<th>Linux 512</th>
<th>Linux 1024</th>
<th>SaJe 56</th>
<th>SaJe 512</th>
<th>SaJe 1024</th>
<th>TINI 56</th>
<th>TINI 512</th>
<th>TINI 1024</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Baseline</strong></td>
<td>0.161</td>
<td>0.247</td>
<td>0.351</td>
<td>3.77</td>
<td>5.88</td>
<td>8.29</td>
<td>134</td>
<td>129</td>
<td>141</td>
</tr>
<tr>
<td><strong>XOR-8</strong></td>
<td>0.167</td>
<td>0.264</td>
<td>0.378</td>
<td>4.02</td>
<td>8.42</td>
<td>13.31</td>
<td>198</td>
<td>918</td>
<td>1,706</td>
</tr>
<tr>
<td><strong>AES-128</strong></td>
<td>0.194</td>
<td>0.406</td>
<td>0.647</td>
<td>5.26</td>
<td>23.04</td>
<td>41.93</td>
<td>647</td>
<td>6,460</td>
<td>12,600</td>
</tr>
<tr>
<td><strong>AES-256</strong></td>
<td>0.199</td>
<td>0.449</td>
<td>0.717</td>
<td>5.64</td>
<td>28.06</td>
<td>51.75</td>
<td>805</td>
<td>8,545</td>
<td>16,690</td>
</tr>
<tr>
<td><strong>DES-56</strong></td>
<td>0.190</td>
<td>0.538</td>
<td>0.914</td>
<td>6.15</td>
<td>37.84</td>
<td>71.53</td>
<td>1,052</td>
<td>12,916</td>
<td>25,500</td>
</tr>
<tr>
<td><strong>3DES-168</strong></td>
<td>0.229</td>
<td>0.997</td>
<td>1.796</td>
<td>10.13</td>
<td>95.28</td>
<td>185.64</td>
<td>2,764</td>
<td>37,720</td>
<td>74,602</td>
</tr>
<tr>
<td><strong>Parity</strong></td>
<td>0.165</td>
<td>0.307</td>
<td>0.467</td>
<td>4.08</td>
<td>8.38</td>
<td>13.11</td>
<td>233</td>
<td>995</td>
<td>1,843</td>
</tr>
<tr>
<td><strong>MD5</strong></td>
<td>0.184</td>
<td>0.318</td>
<td>0.471</td>
<td>6.20</td>
<td>17.35</td>
<td>29.36</td>
<td>919</td>
<td>3,870</td>
<td>7,010</td>
</tr>
<tr>
<td><strong>SHA1</strong></td>
<td>0.202</td>
<td>0.360</td>
<td>0.558</td>
<td>6.70</td>
<td>19.95</td>
<td>34.22</td>
<td>1,202</td>
<td>5,440</td>
<td>9,953</td>
</tr>
<tr>
<td><strong>SHA2-512</strong></td>
<td>0.315</td>
<td>0.878</td>
<td>1.466</td>
<td>15.30</td>
<td>57.32</td>
<td>99.48</td>
<td>4,168</td>
<td>18,713</td>
<td>33,186</td>
</tr>
<tr>
<td><strong>XOR &amp; Parity</strong></td>
<td>0.170</td>
<td>0.247</td>
<td>0.351</td>
<td>4.33</td>
<td>10.92</td>
<td>18.13</td>
<td>299</td>
<td>1,787</td>
<td>3,414</td>
</tr>
<tr>
<td><strong>AES-128 &amp; SHA1</strong></td>
<td>0.235</td>
<td>0.538</td>
<td>0.867</td>
<td>8.99</td>
<td>37.63</td>
<td>68.29</td>
<td>2,002</td>
<td>11,959</td>
<td>22,592</td>
</tr>
<tr>
<td><strong>AES-256 &amp; SHA2</strong></td>
<td>0.375</td>
<td>1.095</td>
<td>1.859</td>
<td>19.83</td>
<td>82.14</td>
<td>145.56</td>
<td>5,892</td>
<td>28,350</td>
<td>51,131</td>
</tr>
</tbody>
</table>

Security Mechanism Latencies for 56/512/1024 byte messages (ms)
Performance Impacts

- Java garbage collection and system/network performance impacts best-case performance

Dave Bakken
Performance Impacts (cont.)

- On Linux and SaJe the experiments were repeated with µs timer.
- The ms and µs results motivate the need for “event filtering”.

![Graphs showing performance impacts](image)

Linux µs resolution

SaJe µs resolution
Outline

• Introduction
• Related Work
• MicroQoSCORBA
  – Middleware Architectural Design Taxonomy
  – Fine-grained Configurable Middleware Framework
  – Embedded System Security
  – Security Subsystem Design & Implementation
  – Experimental Evaluation
• Conclusions
Overview of Ongoing/Future Work

• C++ version in progress, late October-ish
• Temporal profiling toolkit
• Additional protocol support (SMTP, IPv6)
• Wireless compensation layer
• IDS mechanisms for embedded middleware
• CASE tools for (in collab. with Prof. Andrews)
  – Generation of instrumentation & validation code
  – Aspect oriented QoS+resource constraint management

Dual Use:
MicroQoS CORBA as embedded middleware and
MicroQoS CORBA for multi-property QoS investigations
External Use

• MicroQoSCORBA is being used at:
  – CMU: Fault tolerance & Real-time mechanism research
  – U. Maryland: Power-aware middleware

• MQC is being considered at
  – TU Berlin: Interested in MQC for their QoS specification research
  – U. Oslo: MQC’s use has been written into Norwegian Research Council proposal
  – Cisco: Interested in MQC for used in low-end and mid-range routers (control plane; uses TAO now)
  – Boeing: MQC for avionics helping with validation
  – WSU: Beginning joint work with Dr. Andrews
  – Lockheed Martin: military
Conclusions

• MicroQoSCORBA the only framework that
  – Is a “bottom-up” rethinking from the device level of what should be configurable, and in what ways
  – Is tailorable for a given application to the wide range of
    • Device constraints, \textit{and}
    • Application-dictated constraints
    • \textit{with} a fine granularity of configuration constraints
  – Supports both “functional” and QoS properties
    • Security and fault tolerance as well as real-time performance are all key QoS properties

• ... one more important conclusion ...
One More Important Conclusion …

Hunt the Ducks!

Drown them Webfeet!

Smack the Quackers!

Eat Beijing Duck!

radioactive phlegm
The **real** SI cover FYI (outside Oregon):

QB, Meet Will Derting!

(Fark courtesy of DobasD03 on www.cougfan.com)