



An Evaluation of a Real-Time SOA Implementation for the U.S. Surface Navy

Fred Weindelmayer, NSWCDD
Dr. Frank Coyle, Southern Methodist University
Paul Haynes, NSWCDD*

Goal

Goal

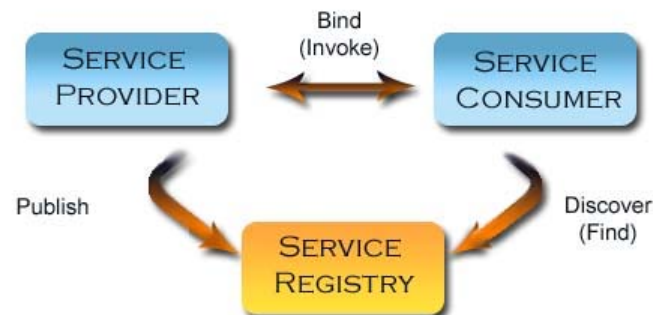
- Investigate if a real-time Service Oriented Architecture (SOA) application can be created using Web Service components* synthesized with real-time technologies, resulting in deterministic performance

* *Web Services are traditionally non-real-time in nature*

Background

What is SOA?

- What is SOA?
 - SOA is an architectural framework, not a technology
 - Applications in an enterprise are designed to be standards-based, interoperable, and discoverable services
 - Follows the publish, find, and bind model
 - Current adoption by industry to enhance distributed enterprise computing



Web Services and SOA

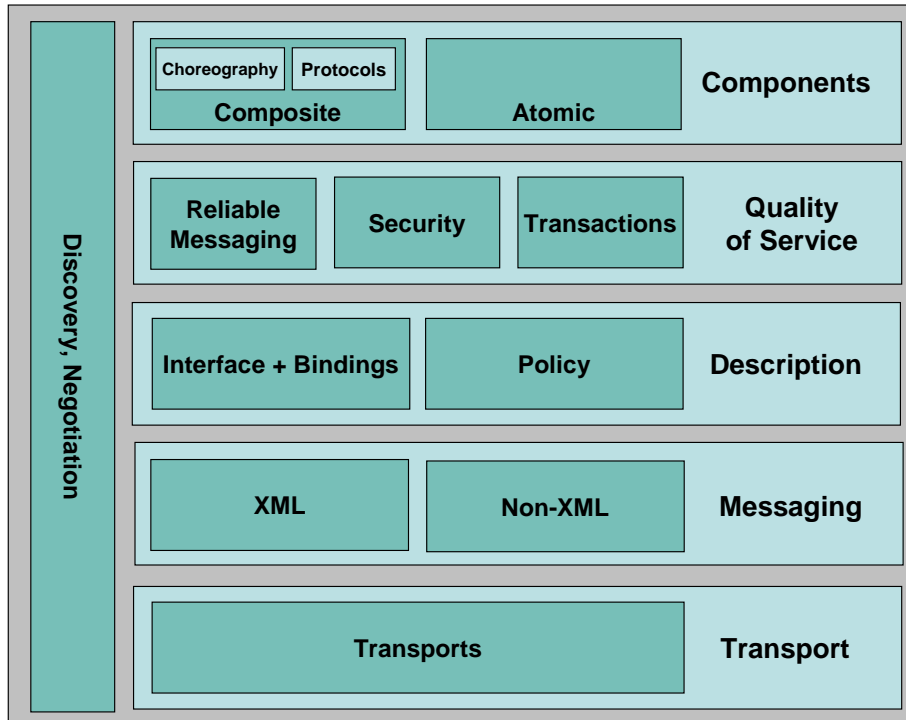
- Web Services is currently the technology base of choice to implement SOAs
- W3C definition of Web Service
 - “A **Web Service** is a software system designed to **support interoperable machine-to-machine interaction** over a network. It has an interface described in a **machine-processable format** (specifically **WSDL**). Other systems interact with the Web service in a manner prescribed by its description **using SOAP-messages**, typically conveyed **using HTTP** with an **XML serialization** in conjunction with other Web-related standards”*
- Web Services provides a popular and accepted set of standards, frameworks, and products for realizing SOAs
 - Over 150 standards known collectively as WS-*
 - WS-* controlled by standards groups including the W3C, OASIS, and WS-I
 - Quality of Service provided via WS-Security, WS-ReliableMessaging, etc.
 - Products include IBM WebSphere, BEA Weblogic, Apache Axis2, etc.

* <http://www.w3.org/TR/ws-gloss/>

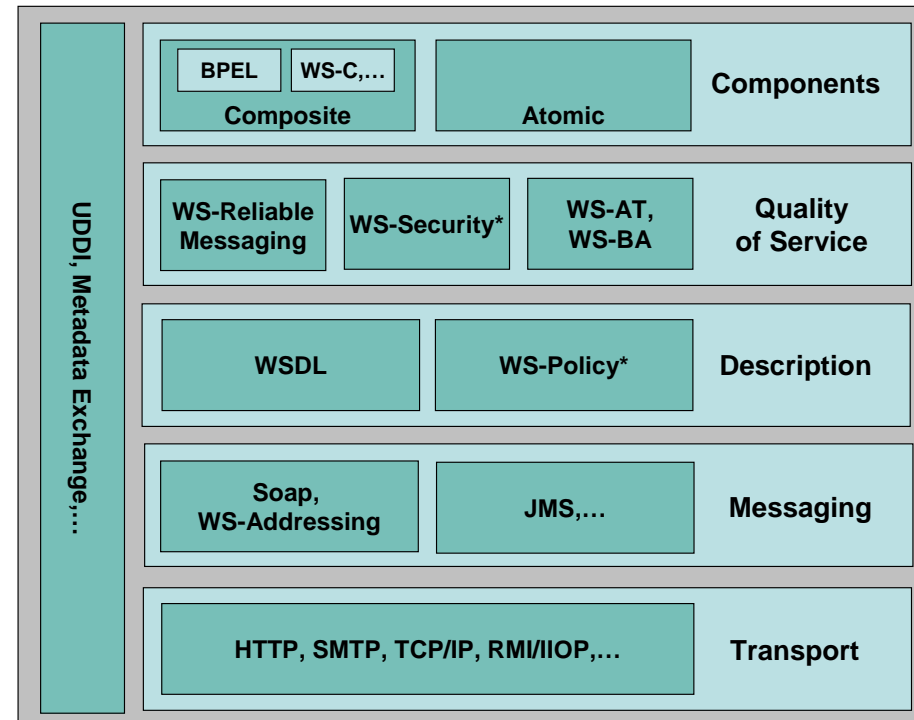
Web Services and SOA (cont).

Comparison of the SOA and Web Services stacks

SOA Stack



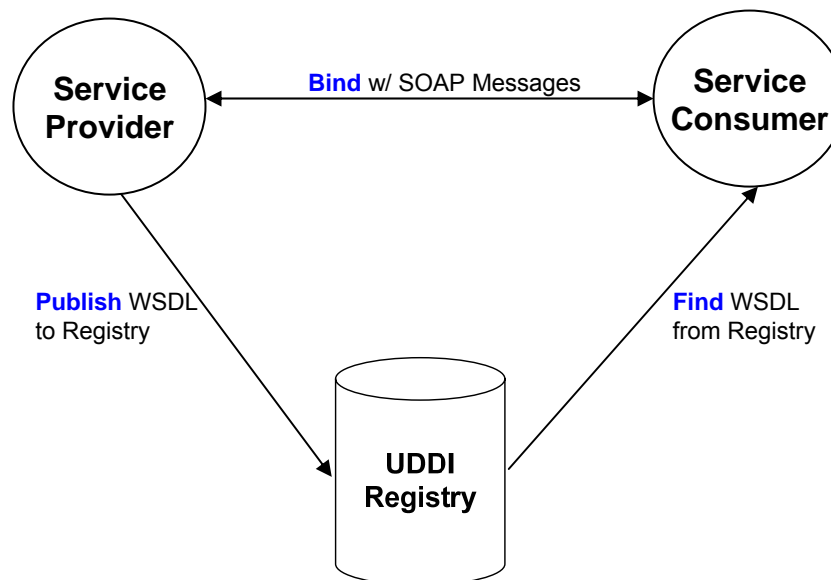
Web Services Stack



There is a one-to-one correspondence between the stacks

Publish-Find-Bind in Web Services

- Universal Description, Discovery, and Integration (UDDI) registry
 - Stores service providers' interface and data description metadata
 - Service providers **publish** WSDL metadata to the UDDI registry
 - Service consumers **find** service providers and how to “connect” to them via the service registry
- Services consumers **bind** to service providers and exchange SOAP messages based on the provider's WSDL over HTTP



Web Services and Problems in Real-time

- Web Services traditionally have been built upon non-real-time technologies
- Messaging using SOAP
 - More expensive to transport ASCII XML over the wire than binary
 - XML can have an expansion factor of 6-8 times that of a binary representation
 - Parsing XML adds further processing latency
- HTTP is the common transport used in Web Services
 - HTTP is built upon TCP which can produce delays from packet reordering, retransmission timeouts, etc.
 - HTTP connections can timeout and a response may never reach its destination
 - As a request-response protocol, may not be a good match for message-based communications with SOAP in a real-time system
 - “Future Web Services are likely to be asynchronous in nature and HTTP may cease to be the default choice anymore”*
- Many Web Services technologies are built with Java which has its own set of problems for use in real-time

* http://www.jaxmag.com/itr/online_artikel/psecom,id,747,nodeid,147.html

DoD Interest in SOA

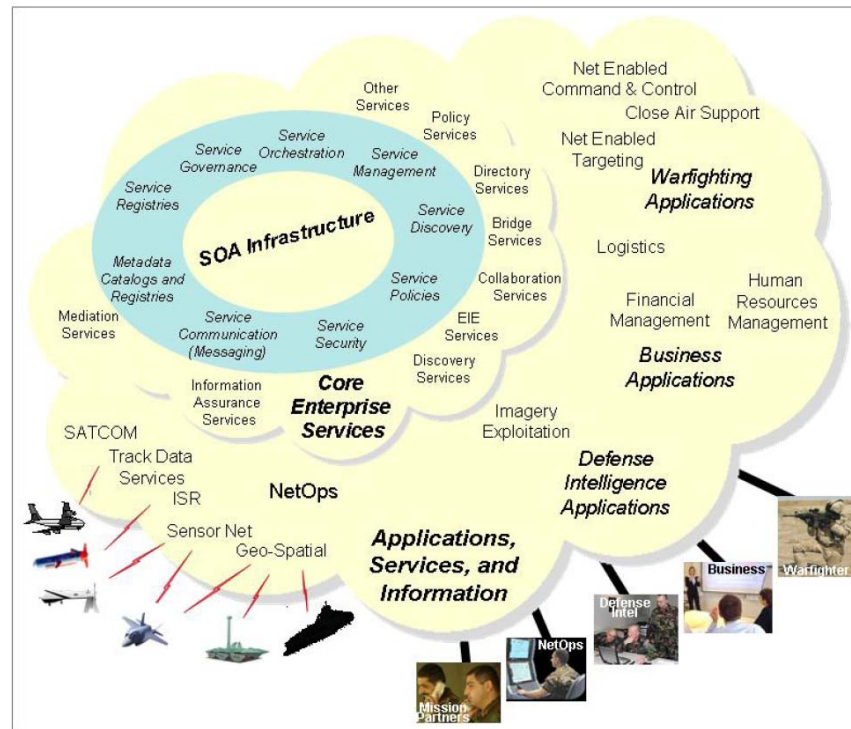
Department of Defense and SOA

- The Department of Defense (DoD) has embraced the notion of SOA as a way to:
 - Reduce stovepipes
 - Increase software interoperability, reuse, and agility, resulting in significant IT cost savings
 - Enable net-centricity through the DoD's Global Information Grid
- DoD SOA Policy
 - 2005 DoD Enterprise Architecture Technical Reference Model
 - 2007 DoD Global Information Grid Architecture Vision
 - 2007 DoD Net-Centric Services Strategy
- Department of the Navy (DoN) is following the DoD vision for using SOAs
 - In 2006, the DoN CIO created a SOA Transformation Charter to implement SOAs in the Navy using web-centric technologies
- The Defense Advanced Research Projects Agency's (DARPA) Strategic Technology Office put out a Broad Agency Announcement (BAA) in 2006 soliciting research for strategic and tactical networks
 - Within this domain the solicitation sought, “novel approaches that **enable mission-centric network enabled real-time service-oriented architectures (SOA)...**”*

* <http://www.darpa.mil/sto/solicitations/BAA07-01/index.html>

Department of Defense and SOA (cont.)

- The DoD vision for SOA permeates the entire enterprise
 - From warfighting applications at the tactical edge with bandwidth and processing limitation to business applications with less stringent constraints



Experiment

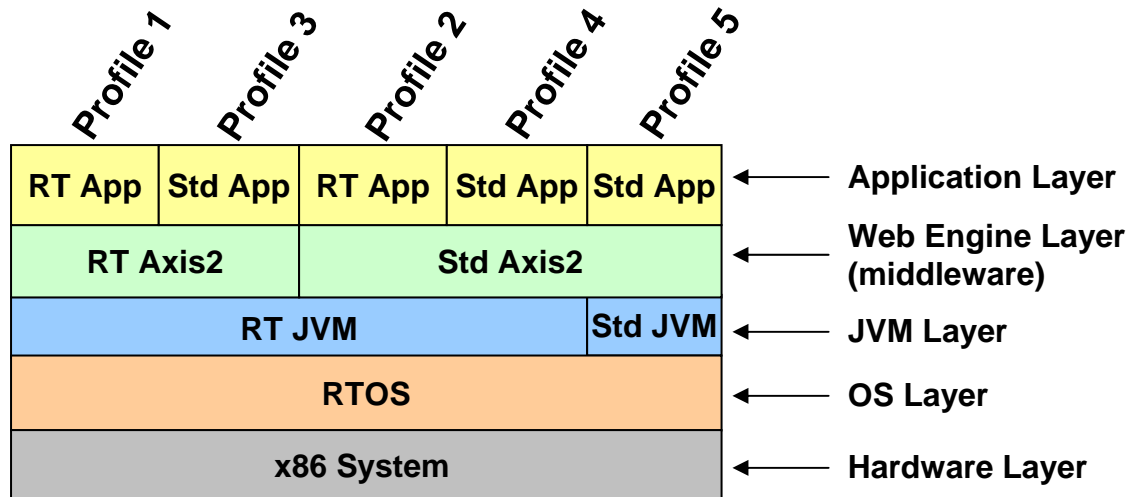
Objectives of the Experiment

- Investigate modifying a non-real-time open source Web Services middleware with real-time Java constructs
 - How difficult is it to modify the existing threading model in such middleware with RTSJ RealtimeThreads?
 - Which threads should be modified?
- Assess the determinism and performance of a Web Services application configured with different real-time mechanisms running on the modified Web Services middleware
 - Compare the internal processing latencies and end-to-end processing latencies of a Web Service application using a RTSJ RealtimeThread vs. the same app using a standard Java Thread
 - Compare the predictable performance of these applications using different profiles of the Web Services middleware (i.e., standard threading version vs. real-time threading version) and JVMs (real-time JVM [RTSJ w/ RTGC] vs. standard JVM)
- Determine if these modifications at the middleware layer and application layer yield sufficient benefit to enable the use of Web Services in real-time SOAs

Methodology of the Experiment

- Use Apache Axis2 as the open source Web Services middleware to modify with real-time Java constructs
 - Axis2 is a popular Web Services SOAP Engine used to host Web Services applications and process SOAP messages
 - Used in IBM's WebSphere J2EE Application Server Community Edition as its JAX-WS engine
- Perform simple “real-time surgery” on Axis2's threading model
 - Replace some standard Java threads with RTSJ RealtimeThreads
 - `org.apache.axis2.transport.http.server.DefaultThreadFactory.java`
 - `org.apache.axis2.engine.AxisEngine.java`
 - Use the Priority Scheduler to manage the RealtimeThreads
 - Rely upon the RTGC to work in harmony with the RealtimeThreads
- Create Web Service applications to measure latencies
 - Message Producer Service which sends out messages at a given rate
 - Message Consumer Service
 - Deployed on the Axis2 SOAP Engine to receive and process messages
 - Records internal and end-to-end processing latencies

Architectural Profiles to Evaluate



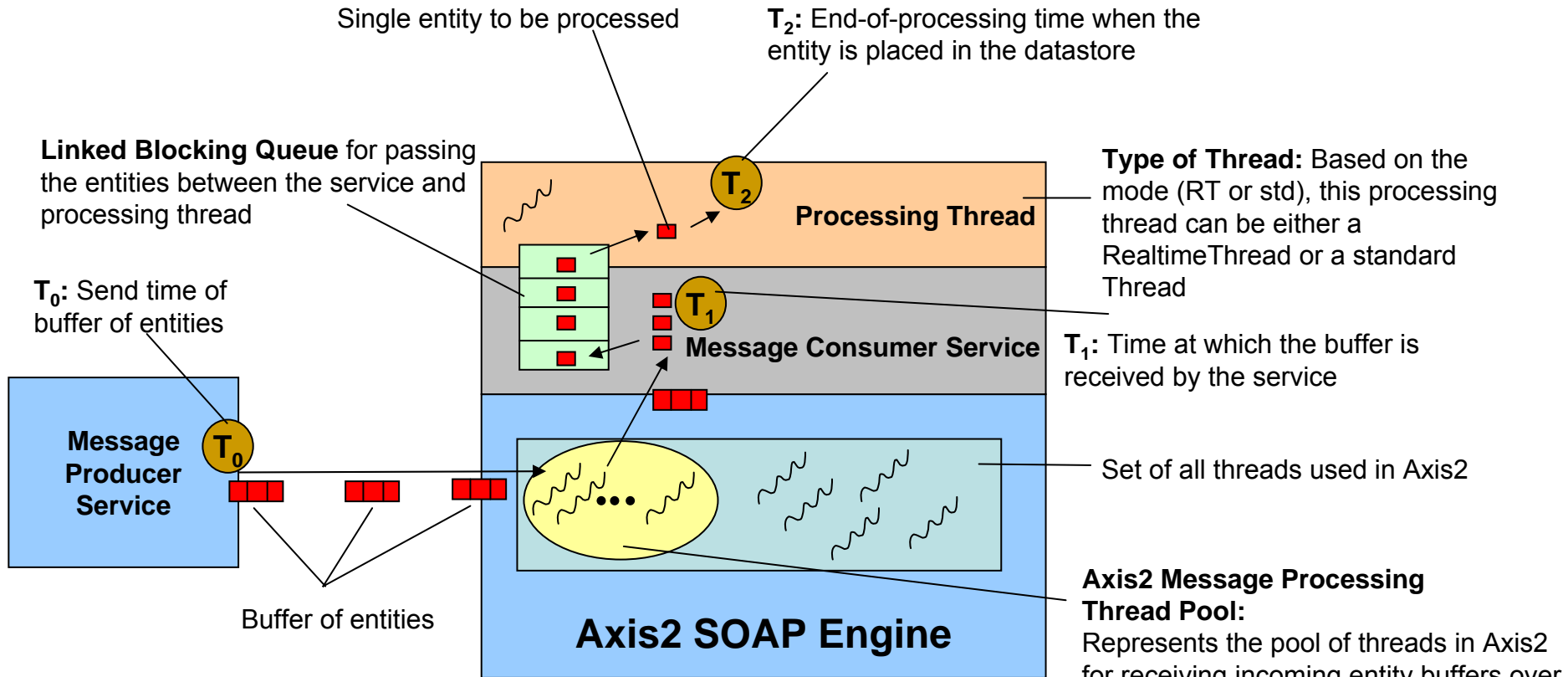
Term	Definition
RT JVM	IBM WebSphere Real Time V1.0 – A real-time JVM that implements the RTSJ and has an RTGC
Std JVM	Sun’s J2SE 1.5 – A standard JVM
RT Axis2	A version of the Axis2 v1.3 Engine with part of the threading model modified to use RealtimeThreads
Std Axis2	An unmodified version of Axis2 v1.3 Engine which uses standard threads in its model
RT App	The Message Consumer Web Service which uses a single RealtimeThread to process incoming messages
Std App	The Message Consumer Web Service which uses a single standard Thread to process incoming messages

Architectural Profiles to Evaluate (cont.)

- Five different profiles composed of a particular combination of:
 - JVM (real-time or standard)
 - Axis2 thread model (real-time or standard)
 - Web Service application thread model (real-time or standard)
- Will the control of the real-time JVM extend through the middleware and application layer resulting in a positive deterministic effect?

	JVM	Axis2	Message Consumer Service (Web Service)
Profile 1	RTSJ J2SE 1.5 w/ RTGC	RT Axis2 (RealttimeThreads used)	RealttimeThread used
Profile 2	RTSJ J2SE 1.5 w/ RTGC	Standard Axis2	RealttimeThread used
Profile 3	RTSJ J2SE 1.5 w/ RTGC	RT Axis2 (RealttimeThreads used)	Standard Thread used
Profile 4	RTSJ J2SE 1.5 w/ RTGC	Standard Axis2	Standard Thread used
Profile 5	Standard J2SE 1.5	Standard Axis2	Standard Thread used

Experiment Design: High-Level Diagram



Instrumentation Points

T_0 : Message Producer sends messages to Axis2 SOAP Engine

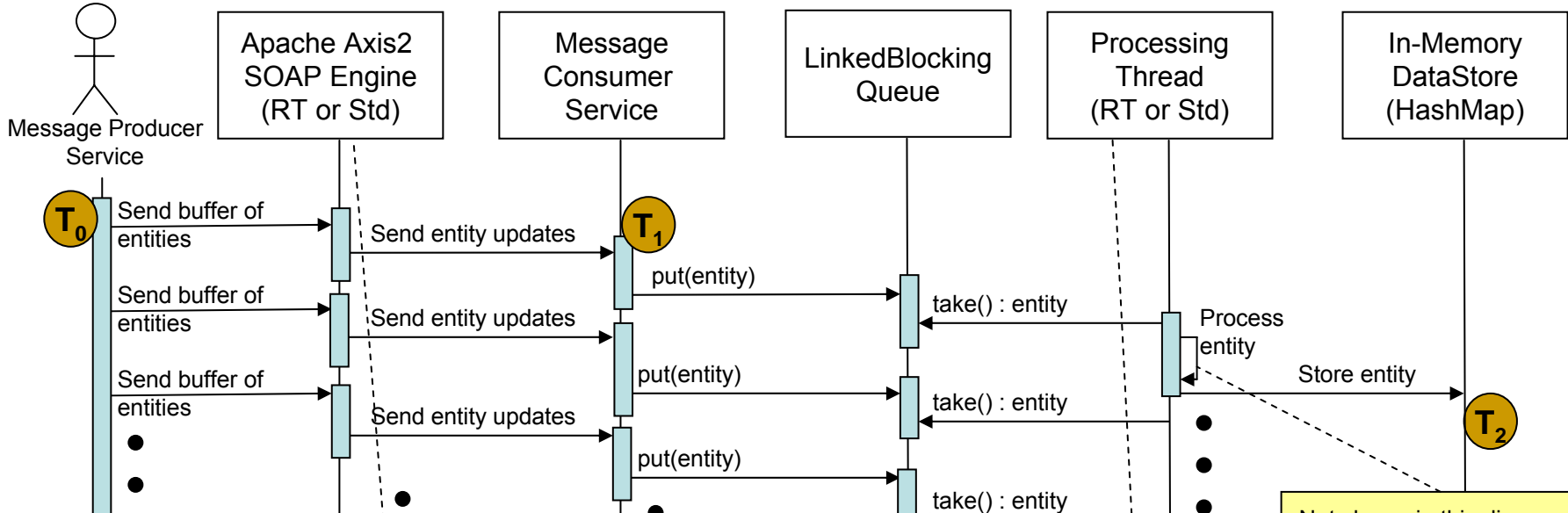
T_1 : Message Service receives messages from Axis2 SOAP Engine

T_2 : Processed message is stored in datastore

$T_2 - T_0$: End-to-end processing latency

$T_2 - T_1$: Internal processing latency to perform processing of received SOAP messages

Experiment Design: Sequence Diagram



Scenario

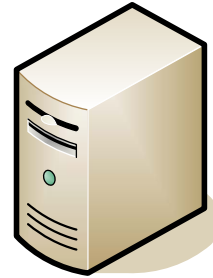
- 100 entities sent from the Message Producer Service to the Message Consumer Service on a 1 second interval
- Message Producer Service sends 5 entities per buffer (e.g., 20 buffers sent per second = 100 entities)
 - Entities sent in buffers in order to reduce the overhead of numerous HTTP connections
- Test duration is 20 minutes
- Nodes time synched to 10's of microseconds with NTP

Instrumentation Points

- T_0 : Message Producer sends messages to Axis2 SOAP Engine
- T_1 : Message Service receives messages from Axis2 SOAP Engine
- T_2 : Processed message is stored in datastore
- $T_2 - T_0$: End-to-end processing latency
- $T_2 - T_1$: Internal processing latency to perform processing of received SOAP messages

level by instantiating (using reflection) a single RealtimeThread or standard Thread which performs the bulk of the business logic on the received messages.

Deployment and Communication Views



Time Server Node

- 2 Intel Xeon 3.06 Ghz CPUs, 1 GB RAM
- Enterprise Linux 5.1 w/ RHEL MRG
- NTP v4

NTP Time Sync

NTP Time Sync



Producer Service Node

Consumer Service Node

SOAP Messages Over
HTTP/1.1

Focus of Experiment

- 1 Intel Core Dual Core 1.86 Ghz CPU, 1 GB RAM
- Enterprise Linux 5.1 w/ RHEL MRG
- IBM WebSphere Real Time V1.0
- Axis2 v1.3 libraries
- Message Producer Service

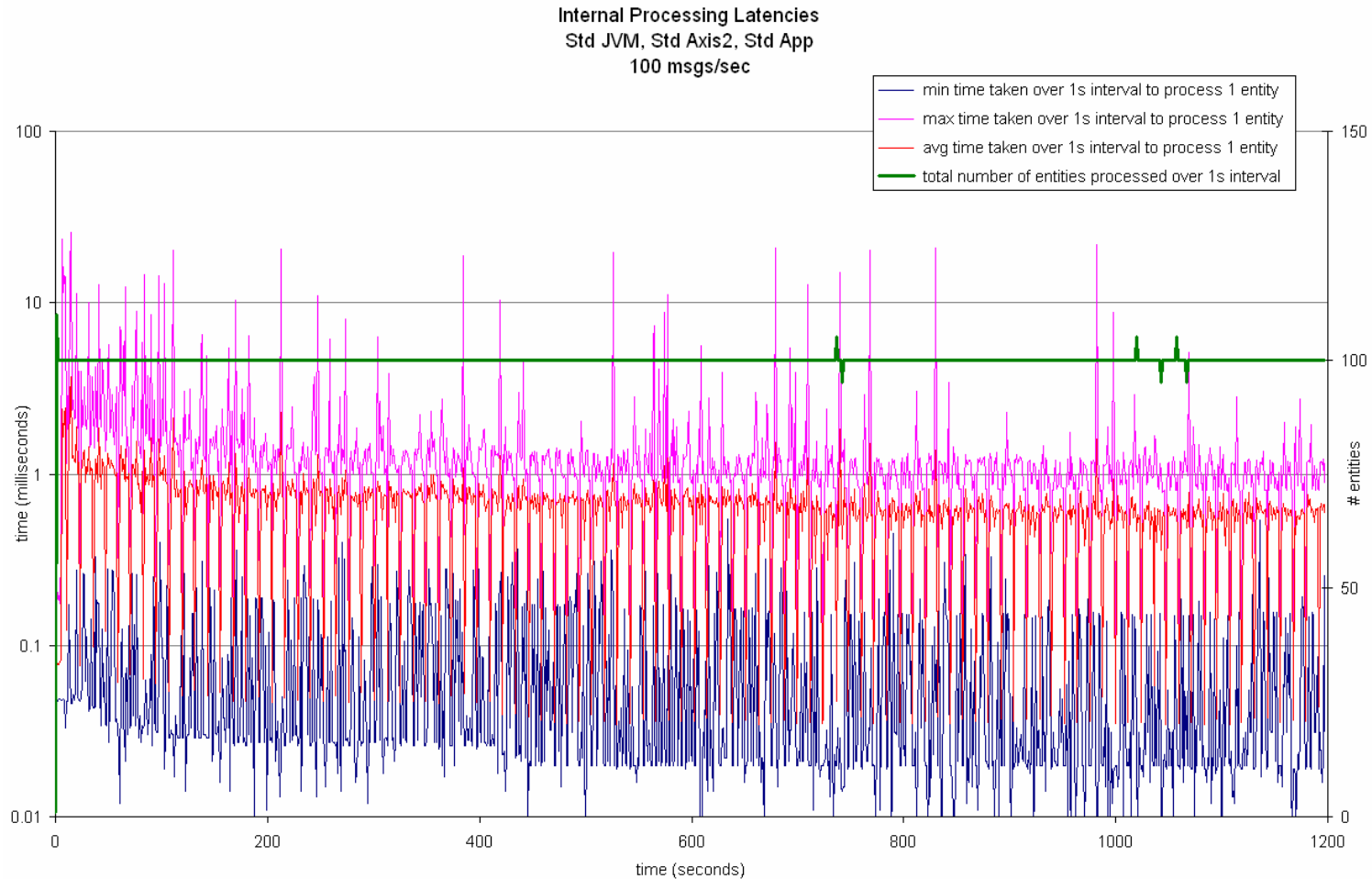
- 2 AMD Athlon 1.8 Ghz CPUs, 2GB RAM
- Enterprise Linux 5.1 w/ RHEL MRG
- JVM
 - IBM WebSphere Real Time V1.0
 - Standard Sun J2SE 1.5
- Axis2 v1.3 SOAP Engine
- Message Consumer Service

SW Configuration

- Out-of-the-box settings used for RTOS, RT JVM, Axis2
- Heap of 512m used for JVMs
- TargetUtilization of 85 used for RTGC

Results from the Five Profiles

Profile 5: Internal Processing Latencies

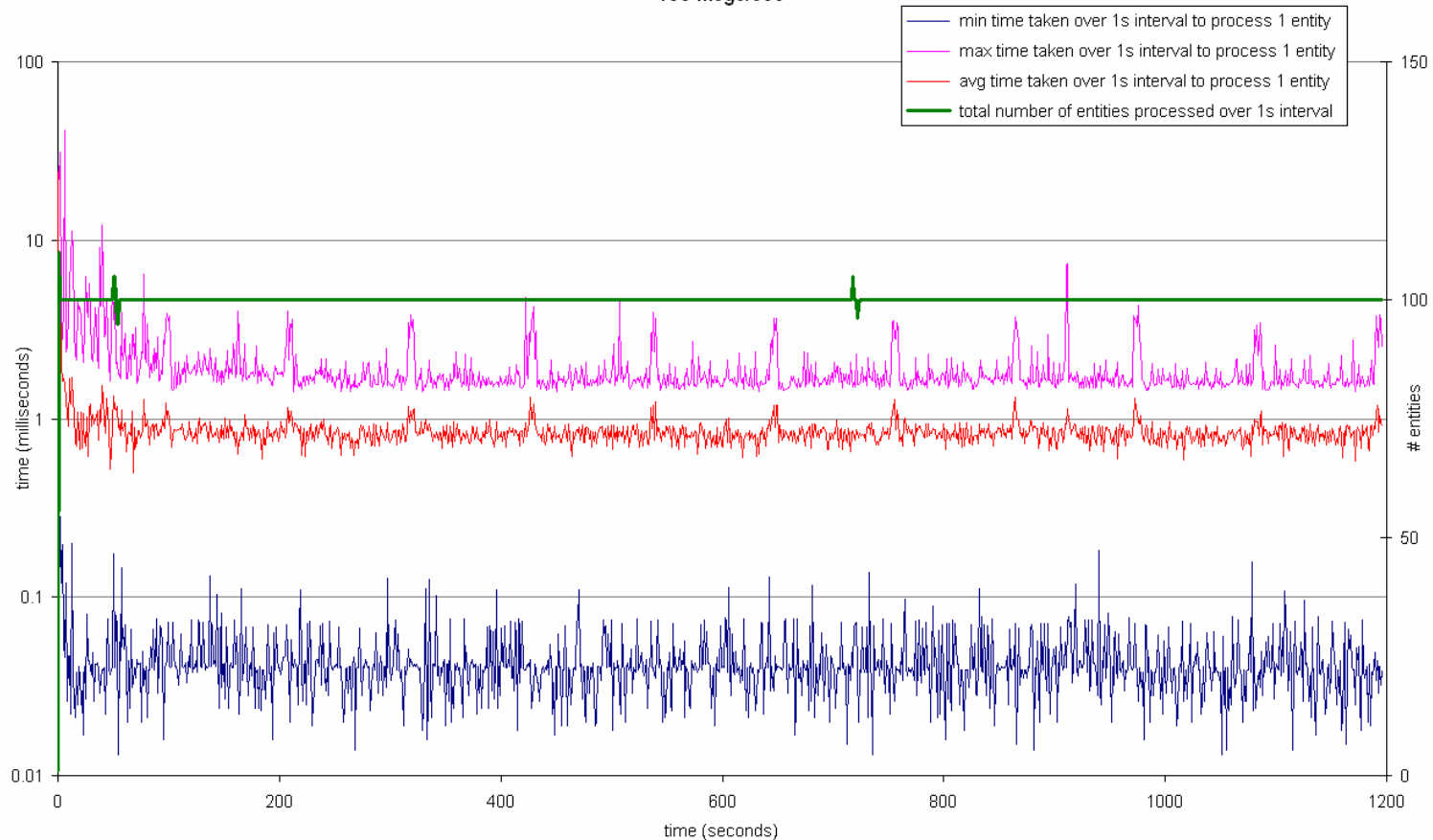


- Many max processing outliers over 1ms and over 10ms
- Many of the averages lie close to the maximums (800 μ s)
- Six instances of jitter in the number of entities processed showing nondeterminism

Profile 3: Internal Processing Latencies



Internal Processing Latencies
 RT JVM, RT Axis2, Std App
 100 msgs/sec

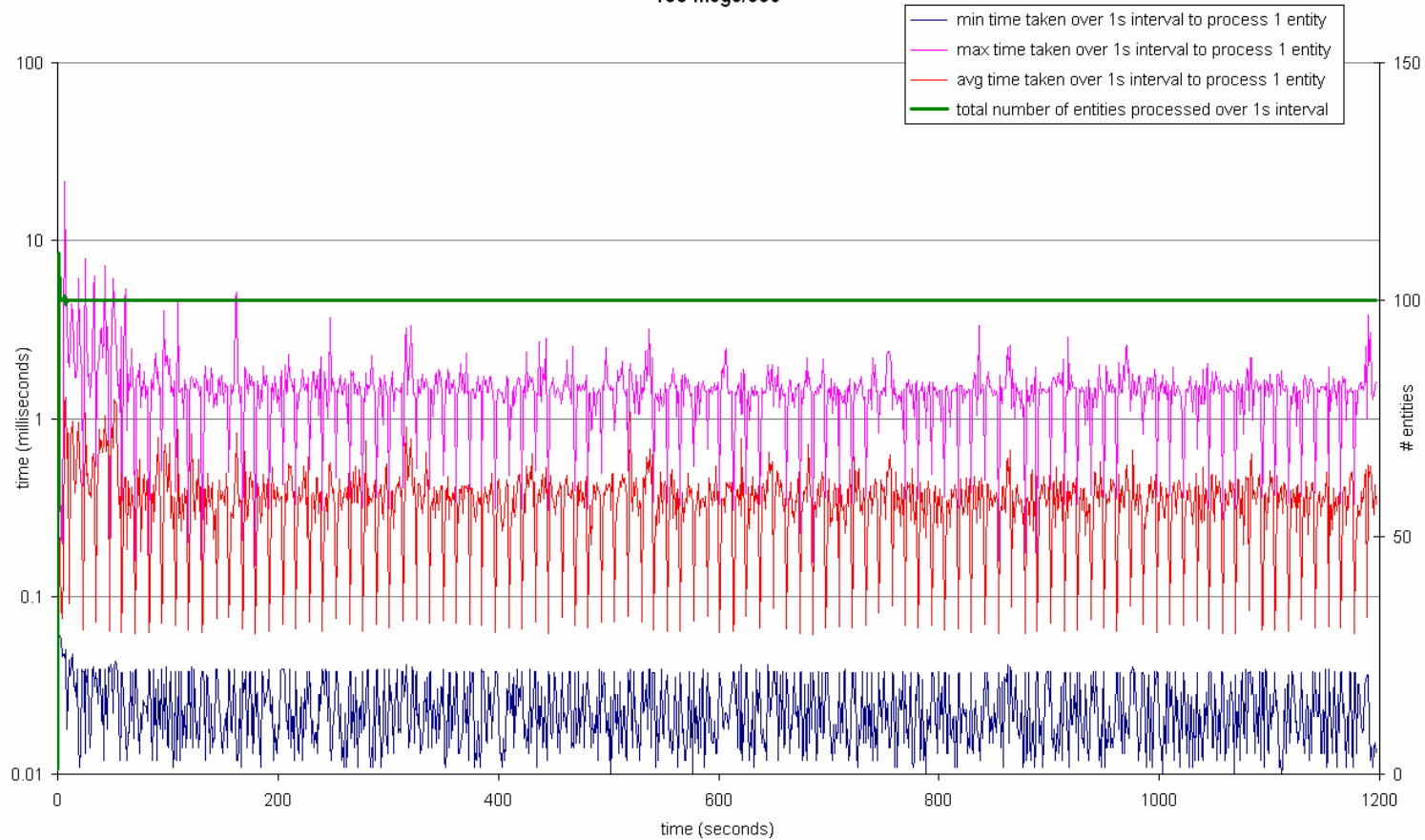


- Majority of maxes between 1.5-2ms. Handful of outliers between 3.5-7ms
- Averages between 700-800 μ s
- Worse determinism and performance makes sense, because standard processing thread is being starved by **23** RT threads in Axis2

Profile 4: Internal Processing Latencies

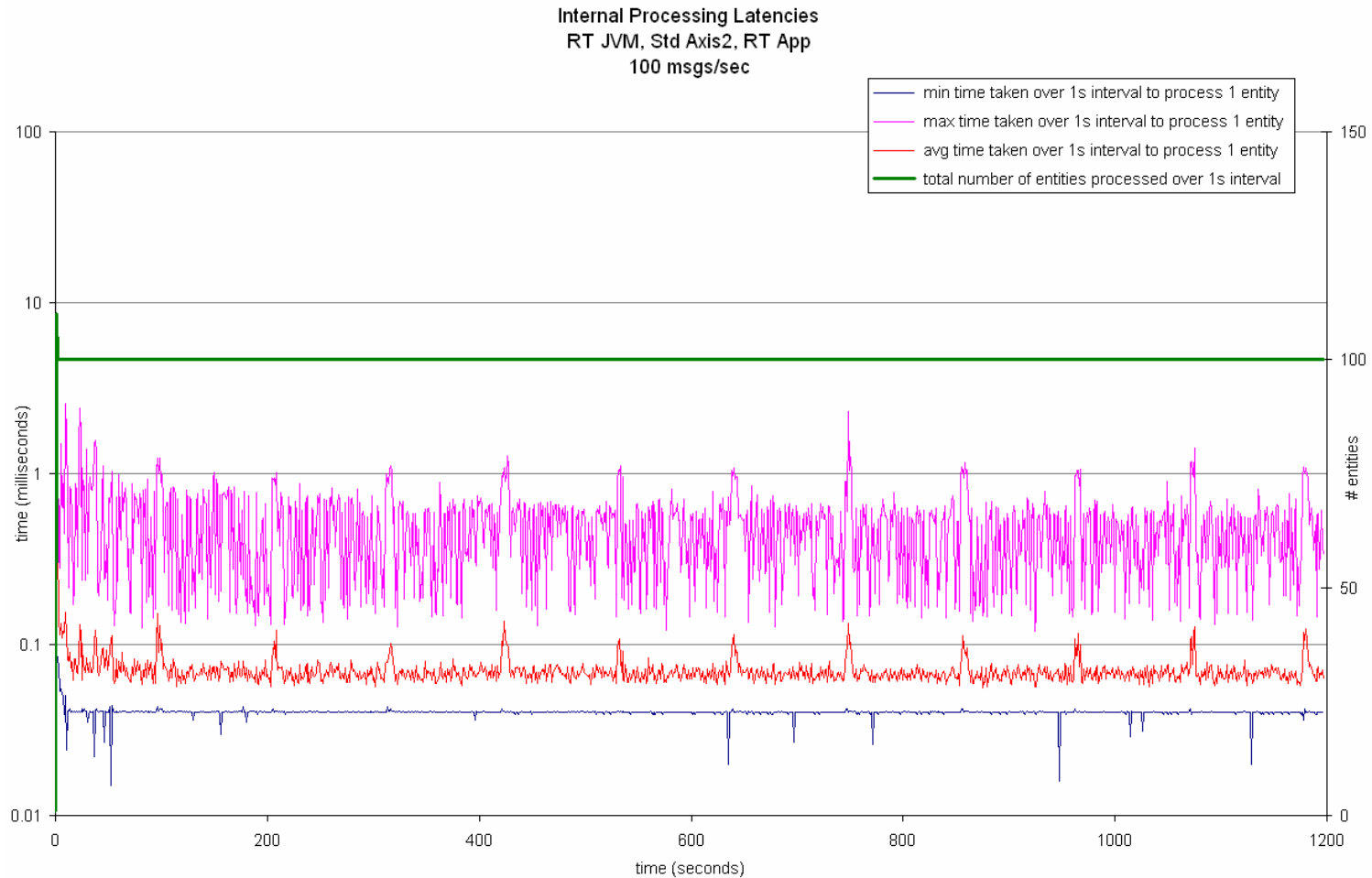


Internal Processing Latencies
RT JVM, Std Axis2, Std App
100 msgs/sec



- Majority of maxes above 1ms, but below 5ms
- Worse performance and determinism when compared to Profile 1 and Profile 2. Better than Profile 5
- Adding a RT JVM to standard middleware and app can boost performance, but does not guarantee determinism

Profile 2: Internal Processing Latencies

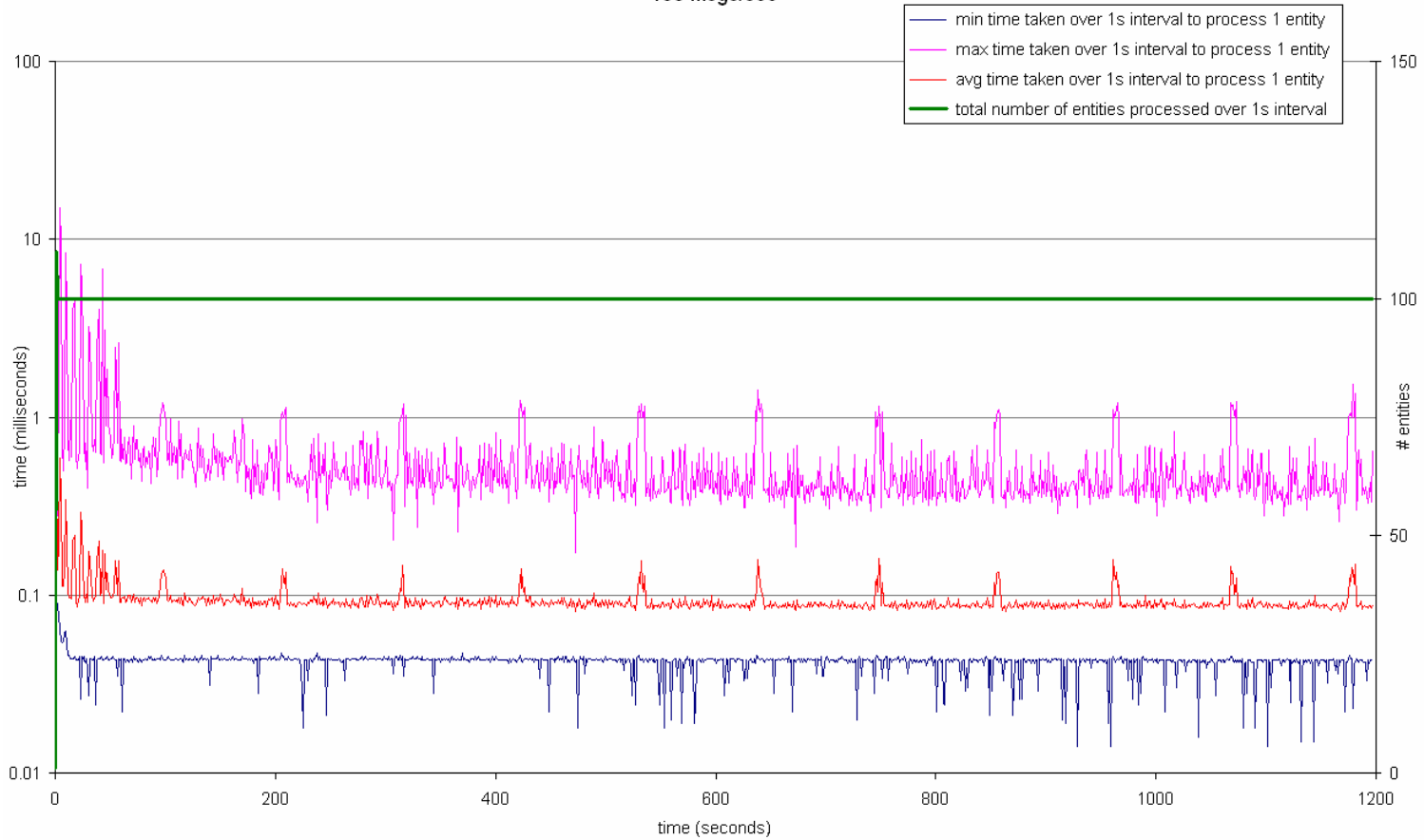


- Majority of maxes bounded between 200-700 μ s. A few outliers slightly over 1ms
- One max around 2ms
- Low and tightly bounded averages around 70 μ s. Averages better than Profile 1

Profile 1: Internal Processing Latencies



Internal Processing Latencies
 RT JVM, RT Axis2, RT App
 100 msgs/sec



- Max processing latencies below 1.5ms. Tightly bounded between 300-500µs
- Low and tightly bounded averages around 80µs



Summary of Internal Processing Latencies

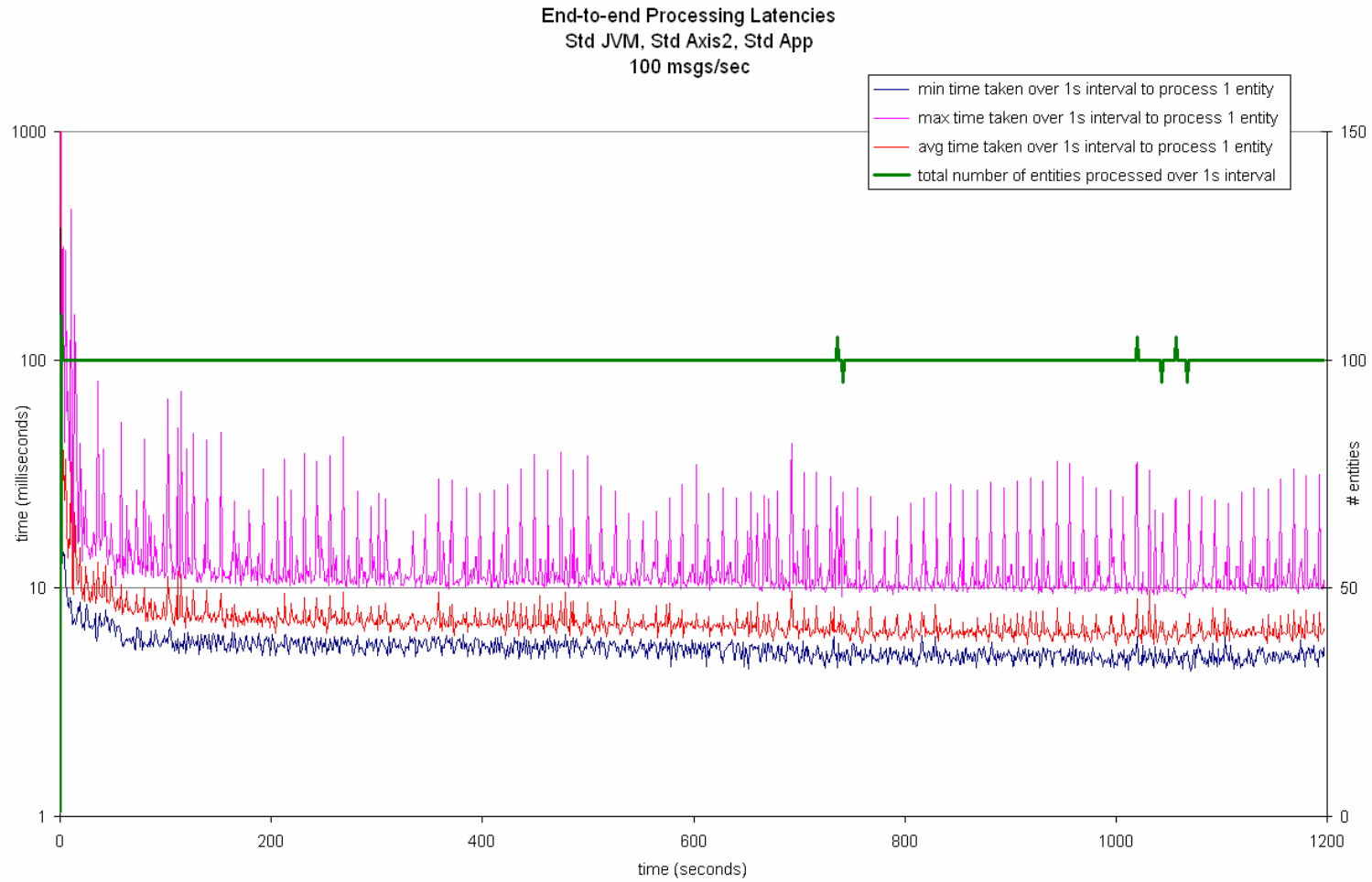
- In the first minute all profiles have higher outliers. Most likely due to JIT, class loading and/or Axis2 creating more threads for processing incoming messages
- Profile 1 has best performance and determinism while Profile 5 has the worst

	JVM	Axis2	App	Determ. Rank	Perf. Rank	Notes
Profile 1	RT	RT	RT	1	1	<ul style="list-style-type: none"> • Maxes bounded at 1.5ms • Majority of maxes sub 1ms (between 300-500µs) • Most averages around 80µs
Profile 2	RT	Std	RT	2	2	<ul style="list-style-type: none"> • A few max outliers around 1ms. One outlier around 2ms • Majority of maxes sub 1ms (between 200-700µs) • Most averages around 70µs
Profile 3	RT	RT	Std	4	4	<ul style="list-style-type: none"> • A majority of maxes between 1.5-2ms. A few between 3-7ms • Most averages around 700-800µs • Presumed std processing thread is starved by RT Axis2 threads
Profile 4	RT	Std	Std	3	3	<ul style="list-style-type: none"> • A majority of maxes between 1-2ms. A few around 3ms • Most averages around 300-400µs
Profile 5	Std	Std	Std	5	5	<ul style="list-style-type: none"> • A number of max outliers over 10ms • Majority of maxes between 900µs and 1100µs • Averages around 600-800µs

RT = real-time

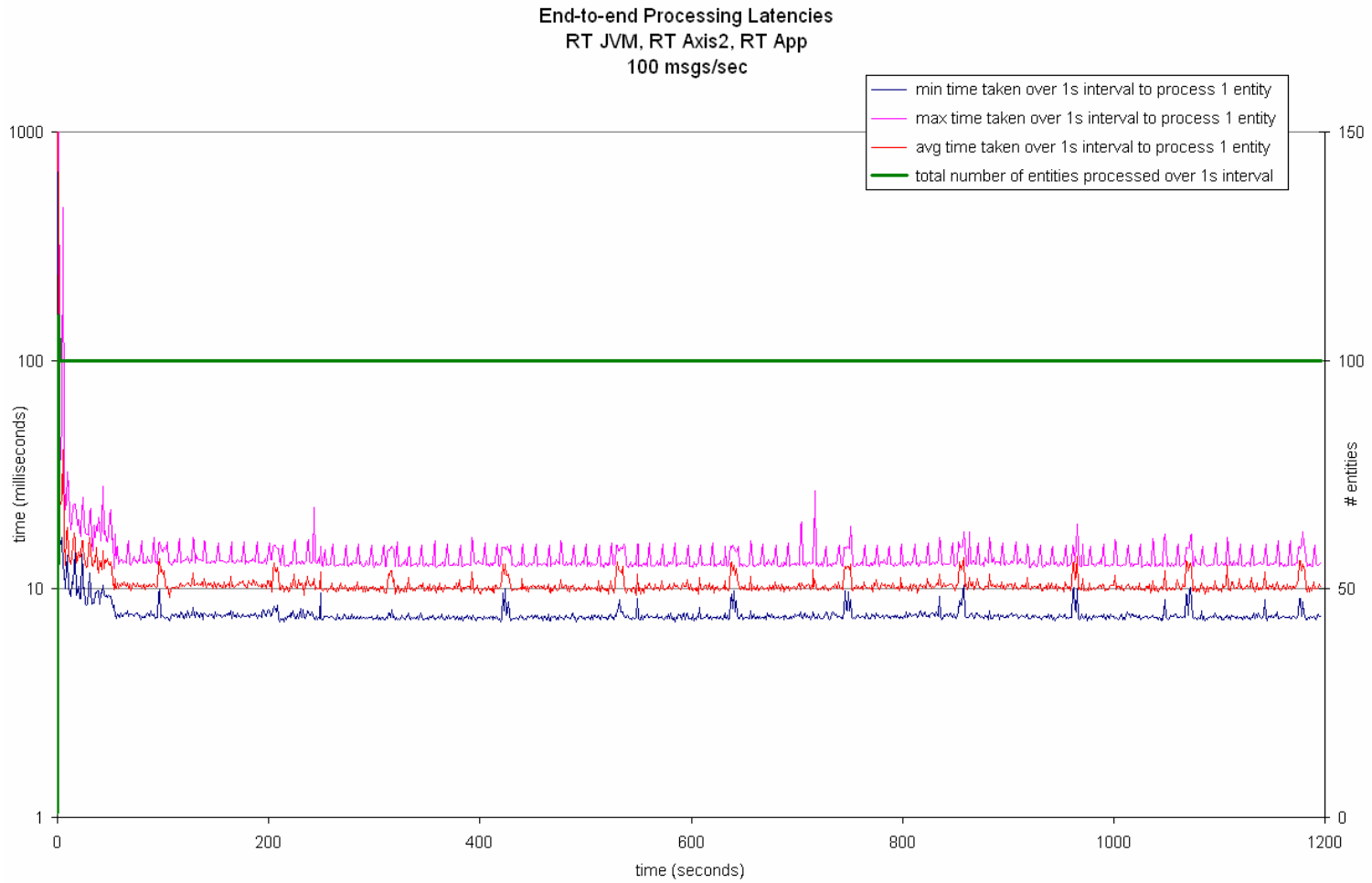
Std = standard

Profile 5: End-to-end Processing Latencies



- Many maxes around 10.5ms. Consistent number of outliers between 25ms and 73ms
- Six instances of jitter with the message processing
- Averages between 6.5-8.5ms

Profile 1: End-to-end Processing Latencies



- Many maxes around 13-16ms
- One outlier at 26ms and 3 near 20ms
- Averages around 10ms

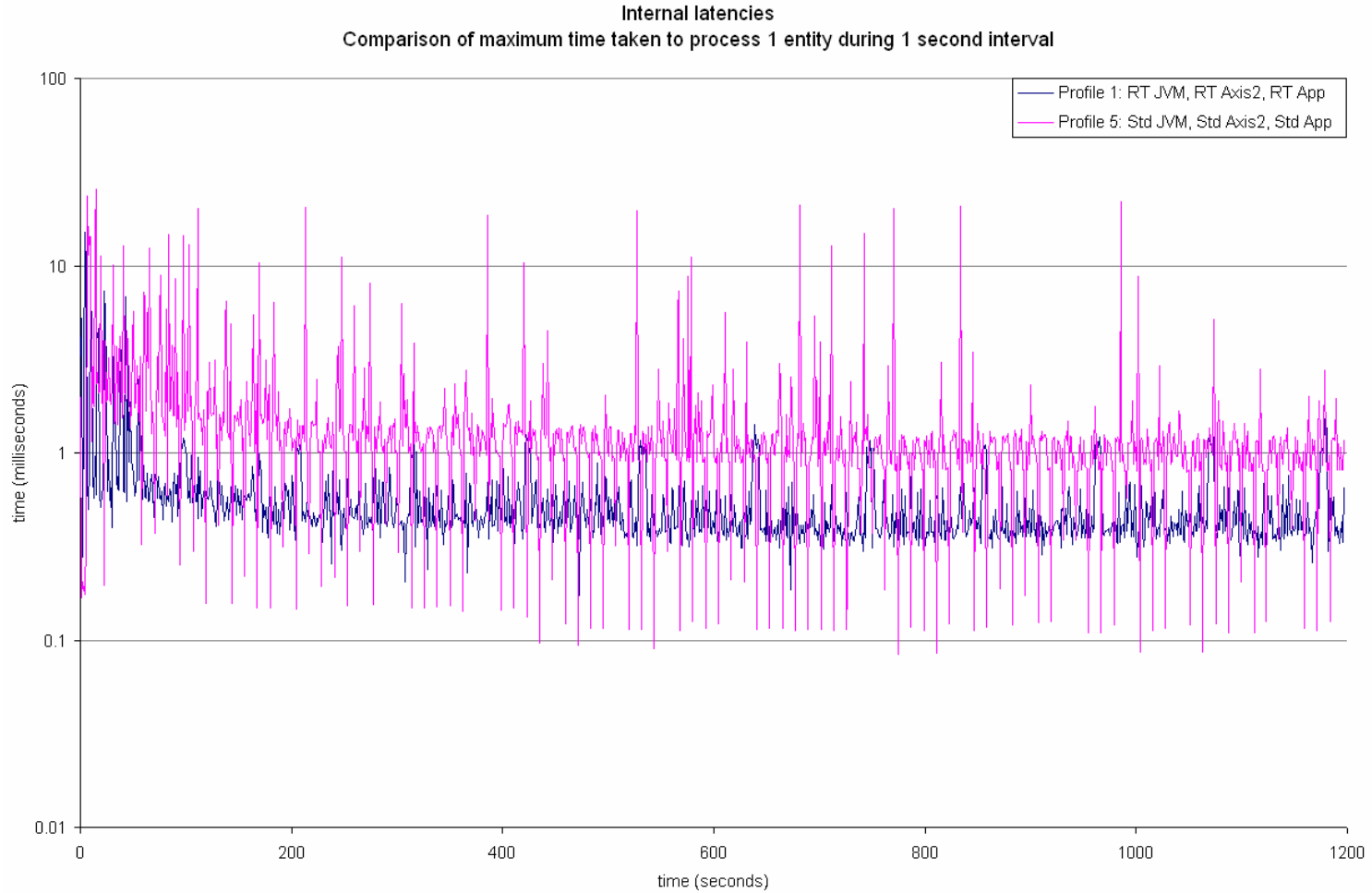


Summary of End-to-end Processing Latencies

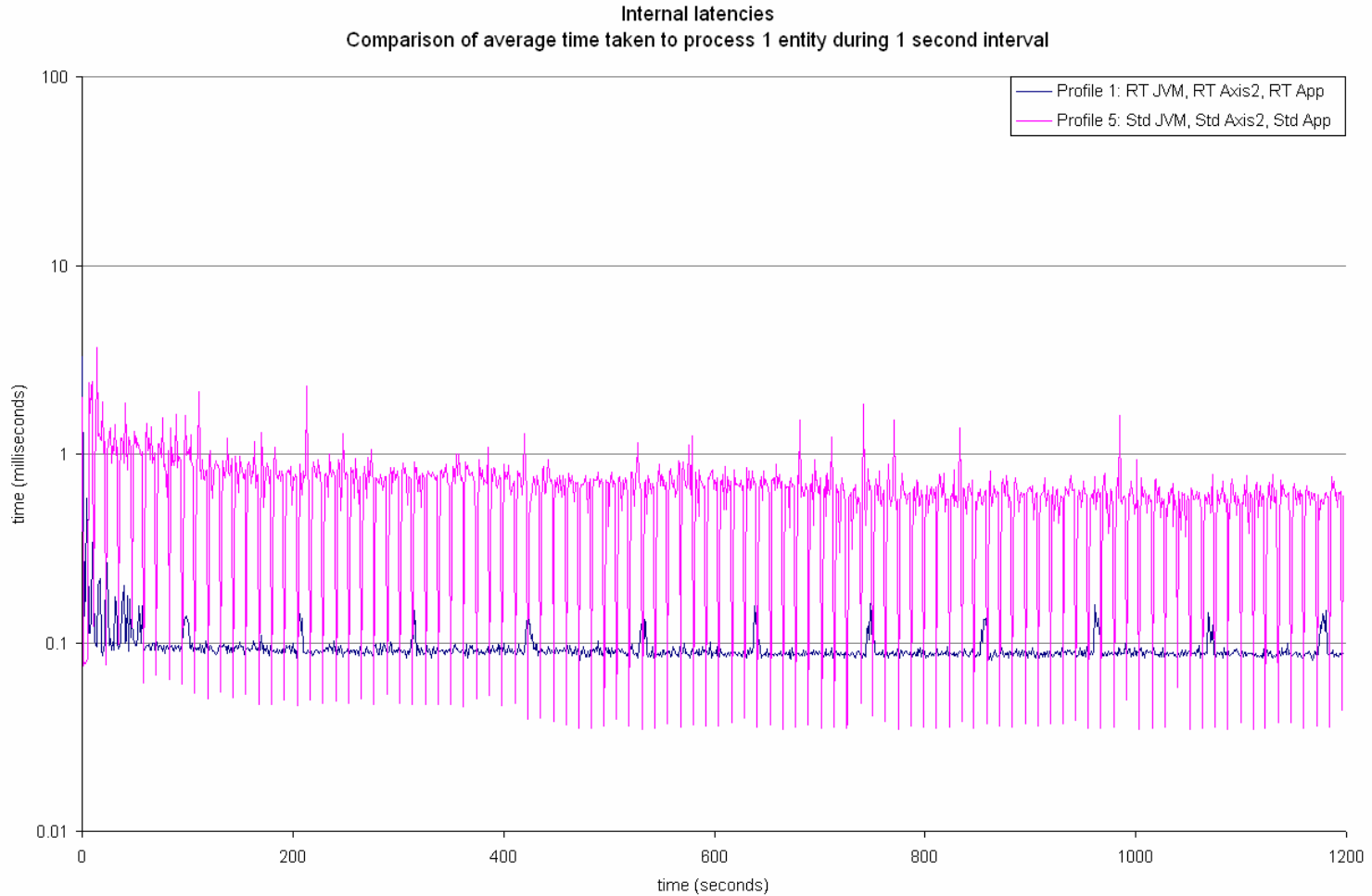
- All the profiles that used the real-time JVM have similar performance and determinism to that of Profile 1
- Profile 1 showed a greater level of determinism than that of Profile 5
 - Tightly bound maxes for Profile 1
 - Profile 5 had many sizeable outliers as well as jitter in its message processing
- Profile 5 had slightly better performance over Profile 1
 - Profile 5 had a lower processing average

Results Comparison

Profile 1 & Profile 5: Max Internal Processing Latency Comparison

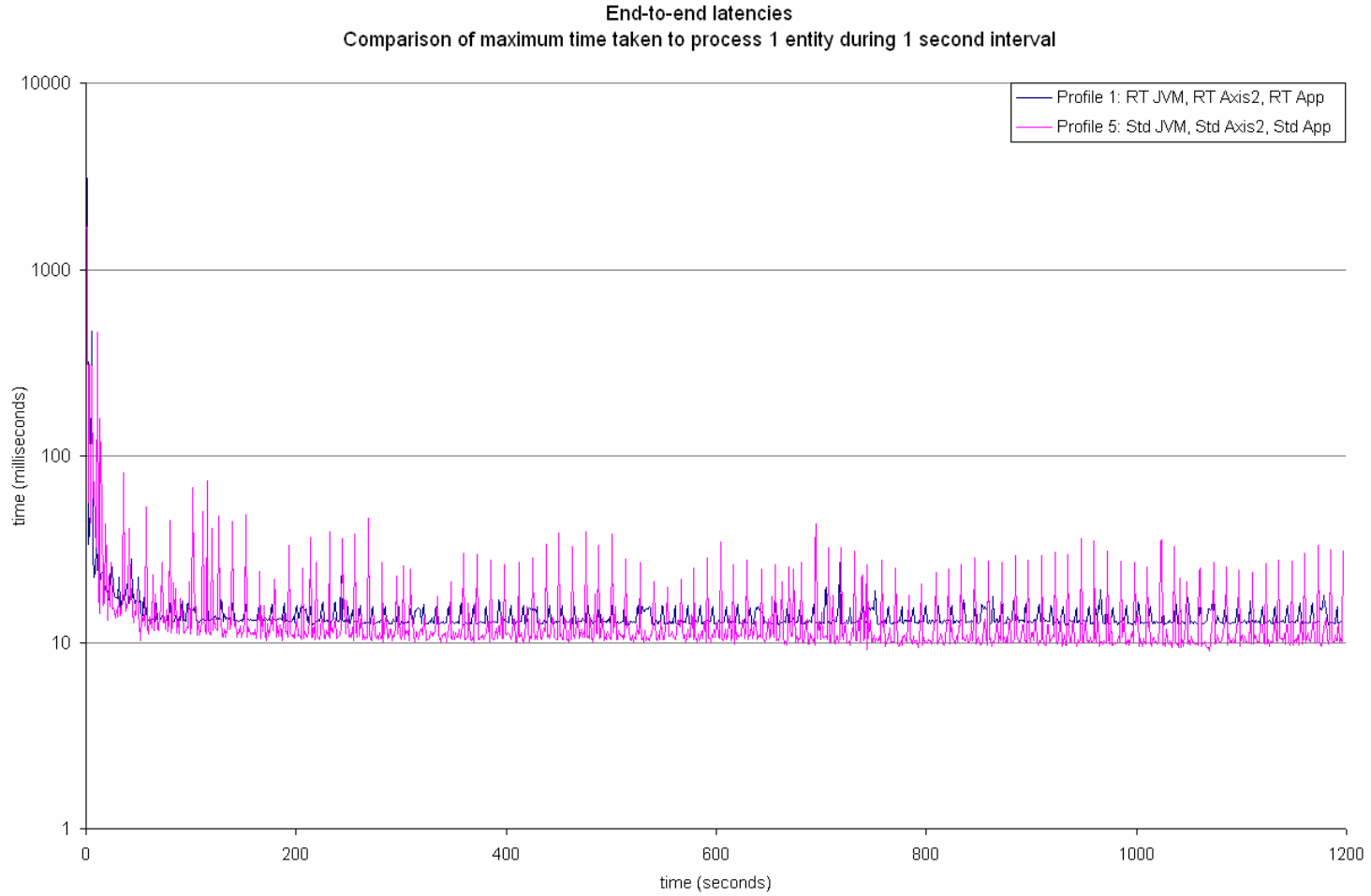


Profile 1 & Profile 5: Average Internal Processing Latency Comparison



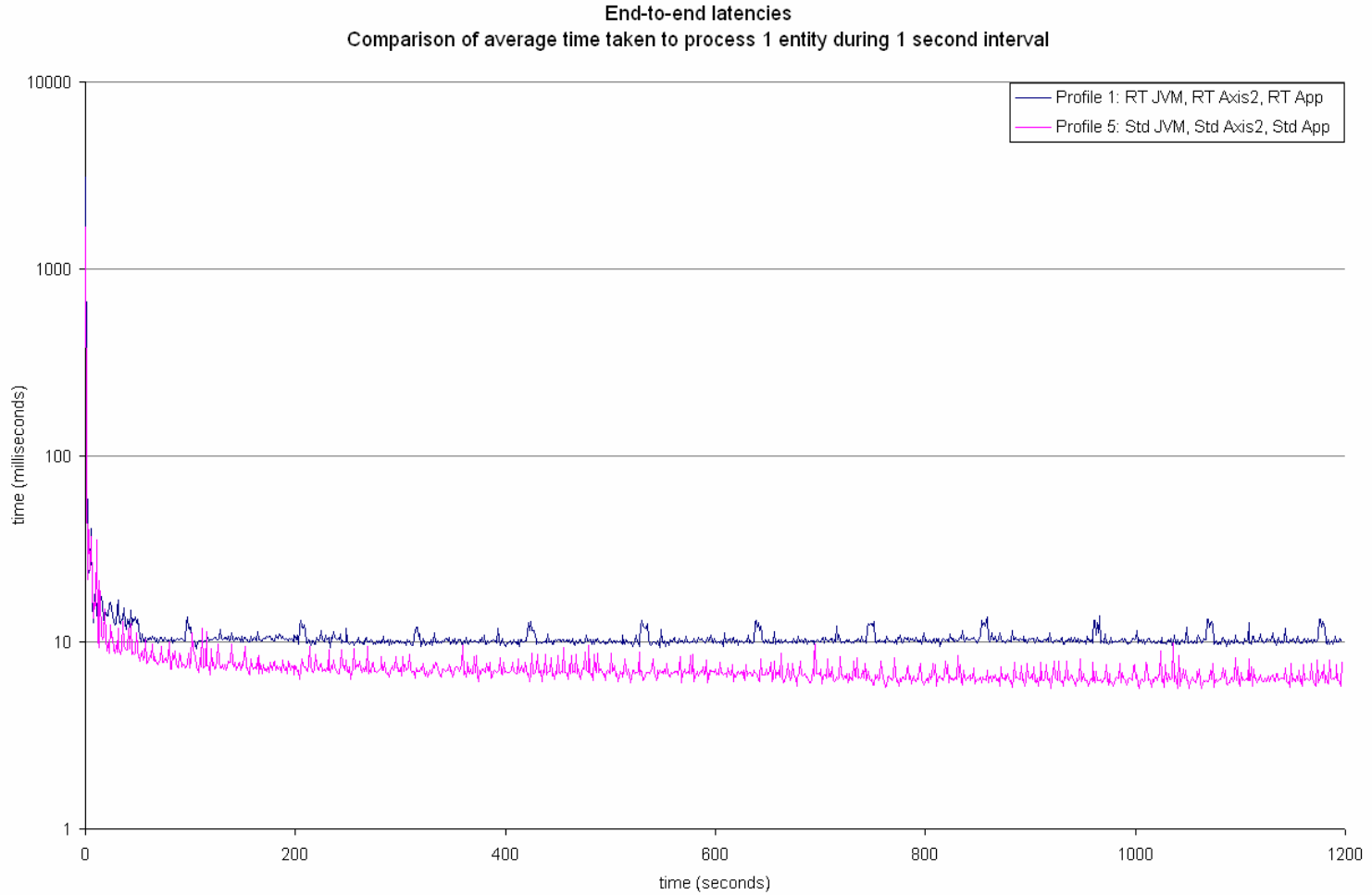


Profile 1 & Profile 5: Max End-to-end Processing Latency Comparison



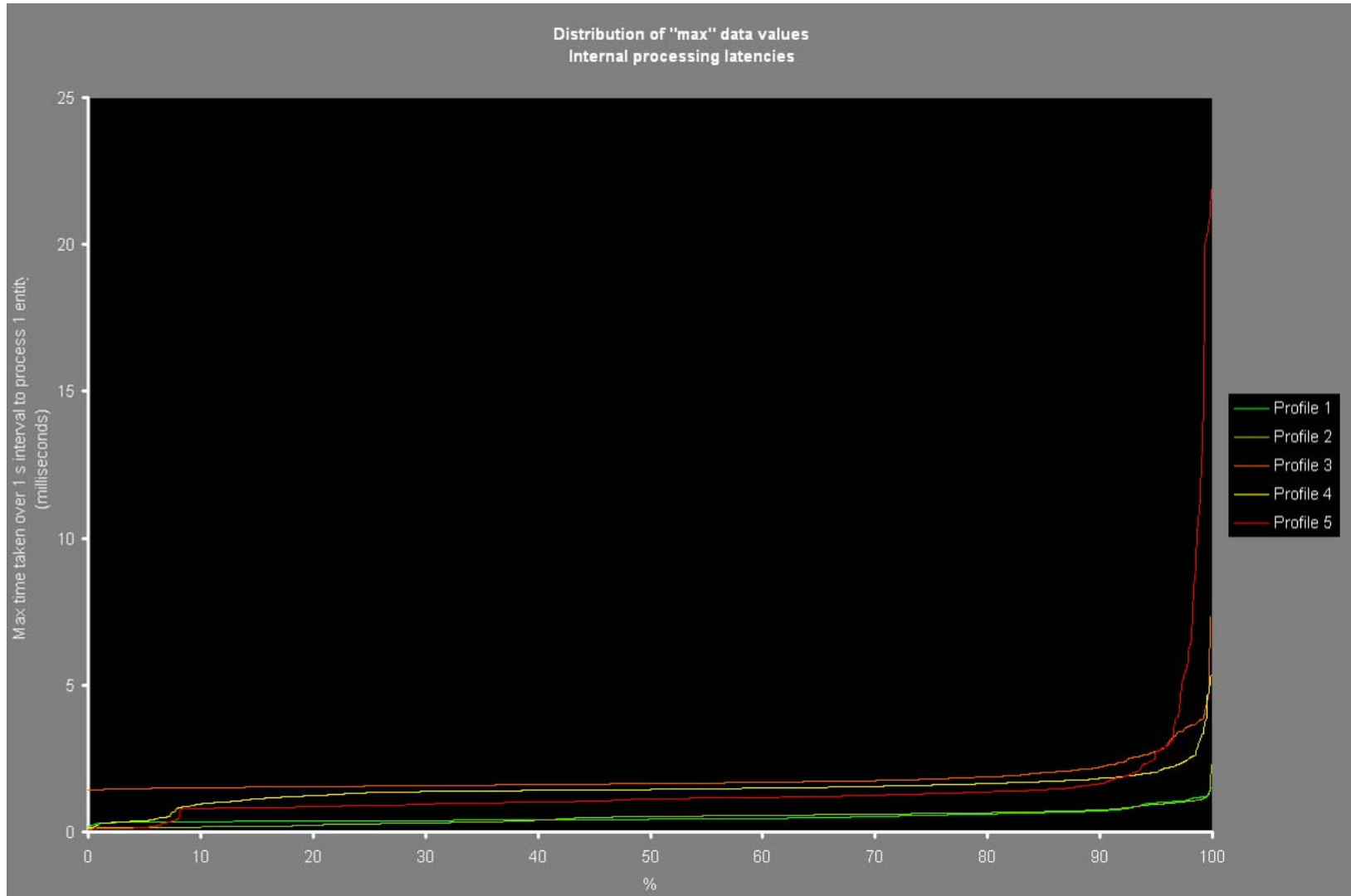


Profile 1 & Profile 5: Average End-to-end Processing Latency Comparison



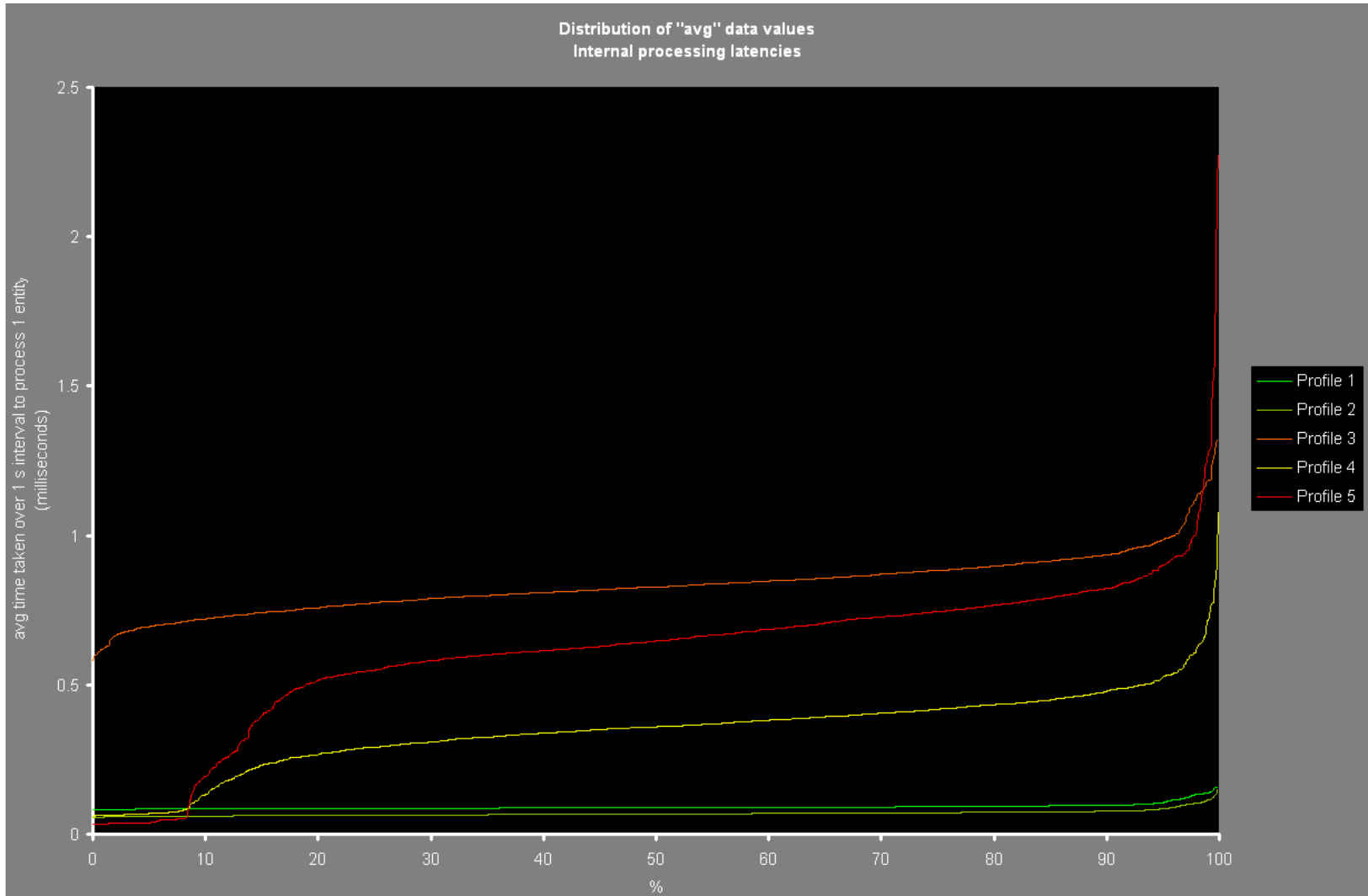
Analysis

Distribution of Maximum Internal Processing Latencies



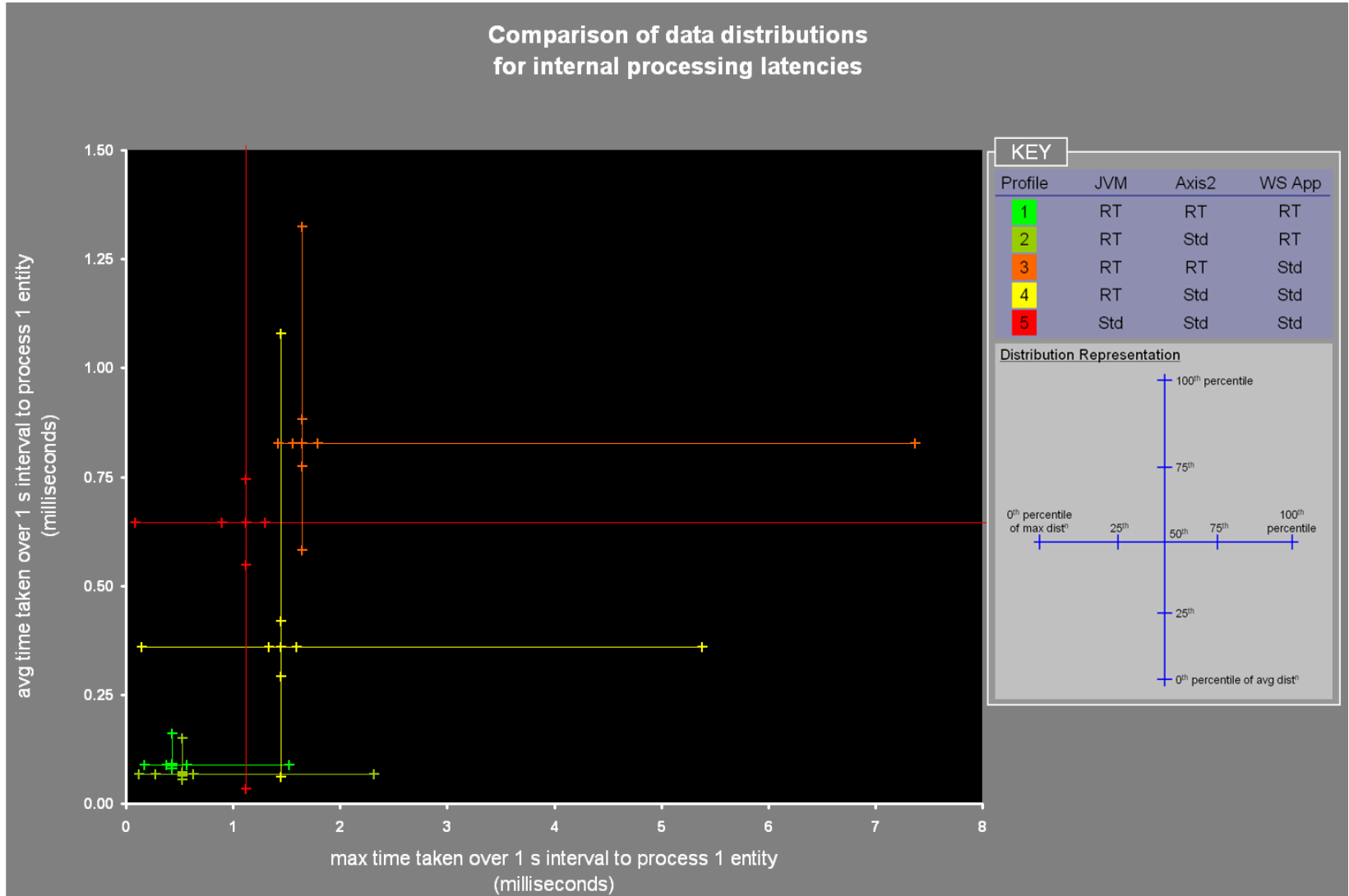
* The first 60 seconds of data from Profiles 1,2, & 4, and the first 100 seconds from Profiles 3 & 5 were omitted from the distribution analysis

Distribution of Average Internal Processing Latencies



* The first 60 seconds of data from Profiles 1,2, & 4, and the first 100 seconds from Profiles 3 & 5 were omitted from the distribution analysis

Comparison of Maximum and Average Latency Distributions



* The first 60 seconds of data from Profiles 1,2, & 4, and the first 100 seconds from Profiles 3 & 5 were omitted from the distribution analysis

** 100th percentile values for Profile 5 extend beyond the plotted region; they are 2.3ms (avg) and 21.9ms (max)

Conclusions

Conclusion

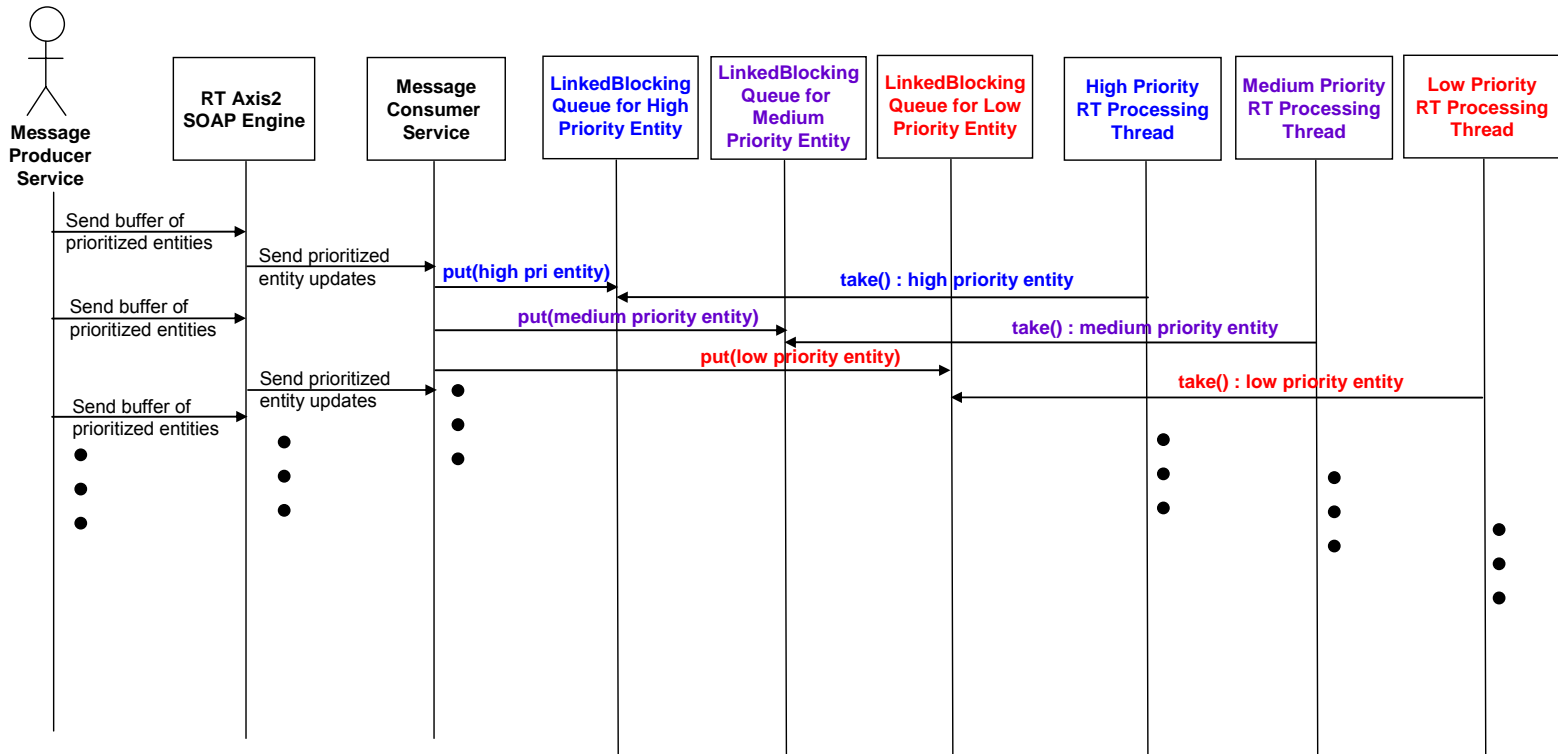
- A real-time Service Oriented Architecture (SOA) application can be created using Web Service components synthesized with real-time technologies, resulting in deterministic performance
- Initial indications are that this technical direction has merit

Conclusion (cont.)

- Using real-time technologies (i.e., RTOS, RT JVM, and RTGC) and injecting real-time mechanisms (i.e., RealtimeThreads, Priority Scheduler, etc.) within an open-source web service engine, such as Apache Axis2, and the Web Services application which rests upon the engine **can improve both the performance and determinism** within the internal processing of the application. Such an approach can aid in realizing real-time SOAs with Web Services
- Since out-of-the-box settings were used, **even better performance and determinism likely for Profile 1 with tuning** of RTOS, RT JVM, RTGC, and Axis2
- Internal processing latencies had significantly better determinism and performance for Profile 1 (real-time profile) when compared to Profile 5 (standard profile)
 - For this application, maximum internal processing latencies were bounded below 1.5ms for Profile 1
- Although end-to-end latencies were not the focus of this study due to the processing “noise” caused by the HTTP transport and XML message sizes, Profile 1 again showed better determinism than Profile 5. However, Profile 5 had slightly better end-to-end performance

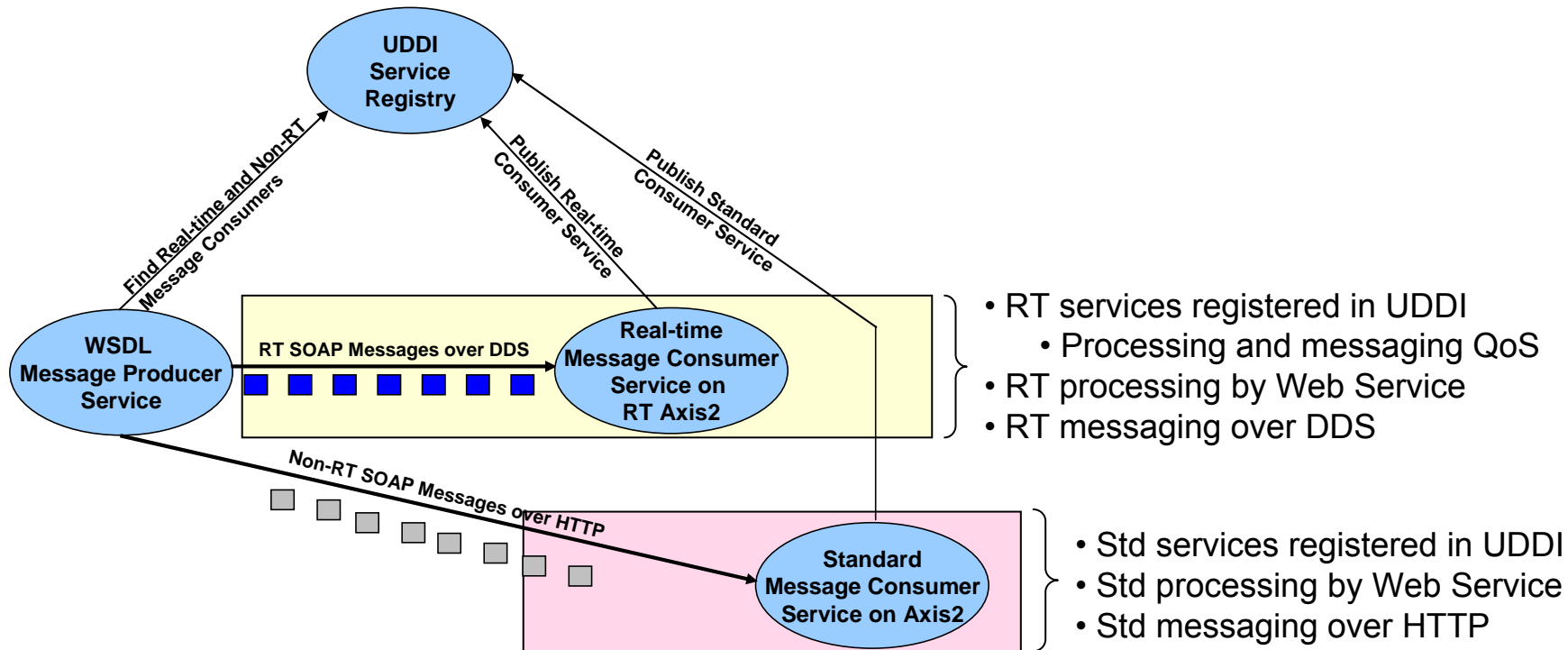
Next Steps

- Increase thread contention to stress internal processing
 - Send prioritized messages to different queues
 - Have different prioritized RealtimeThreads processing different prioritized messages
 - This would demonstrate the priority scheduler's ability to manage RealtimeThreads with competing priorities.



Next Steps (cont.)

- Reconfigurable Real-time Web Services
 - Services publish the level of real-time fidelity their services provide
 - Service requestor can query the registry for RT services and required QoS
 - RT transport and messaging
 - Service that handles prioritized messages
 - RT guarantees for message processing by Web Service engine and application
 - Service requestor can choose service to meet its real-time requirements



Summary

- Real-time Java technologies can be successfully integrated with Web Services technologies to increase the determinism and performance of a real-time SOA application
- Minor “real-time changes” to the middleware and application layers can be made to realize these benefits
- Additional research in this vein as well as at the transport, messaging, and middleware layers will be necessary to further increase the level of determinism of real-time SOAs instantiated with Web Services

Questions?

