PRISMTECH
Productivity Tools & Middleware

Real-time & Embedded Systems Workshop July 2007

Building Successful Real-time Distributed Systems in Java

Andrew Foster
Product Manager
PrismTech Corporation

The Case for Java in Enterprise Real-Time Systems

- Changes in scale complexity and scope of Real-Time systems are forcing the Real-time community to re-think the way in which it develops software
- Elements of Real-Time programming are now finding their way into applications that had never previously required Real-Time programming expertise
- It is increasingly difficult to view Real-Time developers and business application developers as members of two separate communities
- Ways have to be found to build bridges between the two communities to facilitate the smooth flow of information from the smallest embedded device to the largest enterprise information systems and vice versa
- These trends make it necessary for the Real-Time community to adopt Real-Time platforms that provide higher level abstractions and advanced tools for functional and temporal programming

Meeting Diverse Operational Requirements

Command & Control

Transport Management

Air Traffic Control

- Typically complex & heterogeneous, requiring support for :
  - Different hardware platforms
  - Software written in different programming languages
  - Highly distributed networked environment
- Need to guarantee efficient, predictable, scalable QoS (latency, jitter, throughput, fault tolerance & security) from end-to-end
- Need to be dynamically reconfigurable in order to support varying workloads over the operational lifecycle of the system
- Systems also need to be affordable in order to reduce initial system acquisition costs & recurring upgrade & evolution costs

Benefits of RT CORBA for Mission Critical Systems

- CORBA is well established as a QoS-enabled middleware technology for integrating diverse systems
  - Provides standard multi-language multi-platform support – including support for different communication protocols & hardware characteristics
  - Supports location transparency – clients can continue to access servers regardless of their location on a network
  - Scalable, portable, high performance
  - Interoperable between different implementations
  - CORBA Messaging - providing support for a range of QoS policies
  - Comprehensive supporting services – Naming, Trading, Notification, CCM, Fault tolerance & Security + many more
  - Real-time CORBA addresses the issues of end-to-end predictable QoS across CORBA systems & provides a solution in terms of priority control, synchronization & resource control
  - With the emergence of the Real-Time Specification for Java (RTSJ), developed through the Java Community Process by the Real-Time Expert Group, RT CORBA now available for the Java platform

So Why Can't Java Be used in Real-Time Systems ?

- Use of Java in Real-Time systems is limited by its inability to predictably control the temporal execution of applications due to:
  - Unpredictable latencies introduced by automatic memory management (garbage collectors)
  - Inadequate scheduling control
  - Unpredictable synchronization delays
  - Very coarse timer support
  - No asynchronous event processing
  - No "safe" asynchronous transfer of control

The Real-Time Specification for Java

- The Real-Time Specification for Java (RTSJ), developed through the Java Community Process by the Real-Time Expert Group
  - Objective to enable the real-time embedded software developer to use the Java programming language where predictable/hard real-time behaviour is a must
- Specified as a set of extensions to the Java Technology Model
  - For example, the RTSJ shall not prevent existing properly written non real-time Java programs from executing on implementations of the RTSJ

Key Extensions to the Java Technology Model

- The RTSJ supports the following important new features which make Java suitable for use in “Hard” Real-Time systems:
  - New Real-Time Threads, Scheduling, and Synchronization
  - New Memory Management Schemes
  - Asynchronous Events Handling & Asynchronous Transfer of Control
  - Time & Timers
  - Direct Access to Physical Memory

Resulted in OpenFusion™ RTOrb Java

- OpenFusion RTOrb Java Edition is PrismTech's Real-Time CORBA compliant ORB for the Java platform - **launched in Jan 2006**
- OpenFusion RTOrb Java Edition is the first COTS Java ORB that can be used in **"hard"** Real-Time systems
- Made possible through the emergence of RTJVM implementations based on the Real-Time Specification for Java (RTSJ)
  - e.g., Sun's Java Real-Time System v1.0 (formally known as the Mackinac JVM) & IBM's WebSphere Real-time v1.0 JVM
- OpenFusion RTOrb Java Edition can provide a unified solution for different needs & uses, supporting both **"hard & soft"** Real-Time systems & **"non"** Real-Time Enterprise applications in a single ORB
- RTOrb Java was developed in close collaboration with Raytheon

Continuous Evolution Since Launch

- Since initial release of the product PrismTech has continued to evolve and improve the product based on both our own and customer's experiences of building distributed Real-time systems in Java

- Experience has shown us that there are a number of key concerns that must be addressed in order to develop successful Real-time CORBA systems in Java, these include:
  - improving the IDL to Java language mapping in order to minimize garbage collection
  - supporting multiple Real-time modes of ORB operations (non Real-time, soft Real-time and hard Real-time) and threads of execution from the same application process
  - techniques for maximizing re-use of legacy code in a Real-time Java application
  - leveraging Real-time garbage collectors to improve system predictability and improve ease-of-use
  - what are the tradeoffs that can be made between RTSJ features, real-time predictability and ease-of-use

Improving the IDL to Java Language Mapping

- Current IDL to Java language mapping was never designed for use in Real-time systems
- For generated stub and skeleton code there are a number of places where constructed parameter types (Arrays, Sequences, Structs, Unions etc.) are automatically allocated
- If would be much better if the user had more control over the management of these allocations
- By using Factories or different memory management strategies such as caching to control parameter allocations can have number of benefits in Java systems with temporal requirements

OpenFusion™ RTOrb – Object Caching

- OpenFusion RTOrb's IDL compiler has been extended so that an object cache Factory can be automatically generated in the stubs and skeletons from the IDL for each constructed parameter type
- Cache is configurable and extensible
- Initially this was targeted for use in Java applications that don't use NHRTs and either Scoped or Immortal memory. Types of application that the cache is used for include:
  - Java applications that use standard Java threads and allocate in heap
  - Or Java applications that use Real-time threads but still allocate in heap
- Benefit is that this can be used as a strategy to reduce the amount of garbage generated and hence the number of garbage collection events, improving overall system predictability
- Current implementation limited to server side skeletons targeting Structs and Union parameter types
- Same memory management techniques can be used to generate mappings that can be used in "hard" Real-time systems (using NHRT, Scoped and Immortal memory) – in this case the Factory can be used to make allocations in Scoped (the typical case) or Immortal memory

OpenFusion™ RTOrb – Object Caching Example

Number of Threads - 3	JDK1.5 / Serial GC	RTS 2 / Serial GC
Caching On	9	13
Caching Off	11	15

Number of Threads - 10	JDK1.5 / Serial GC	RTS 2 / Serial GC
Caching On	32	43
Caching Off	38	51

Number of Threads - 20	JDK1.5 / Serial GC	RTS 2 / Serial GC
Caching On	64	85
Caching Off	76	102

Supporting Multiple Modes of Operation

- Initially RTOrb was designed to be used as either a general purpose Enterprise Java ORB or a RTSJ (NHRT, Scoped & Immortal memory) compliant “hard” Real-Time CORBA ORB
- Individual CORBA applications could support either non Real-time threads of execution or “hard” Real-time threads of execution but not both
- Typically only a very small % of the total threads of execution in a system need to support “hard” Real-time QoS.
- However much larger % of threads have some sort of “soft” Real-time QoS requirements

OpenFusion™ RTOrb - Multiple Modes of Operation

- Additional “soft” Real-time mode of operation added to the ORB
  - Uses Real-time Java threads (no NHRTs)
  - Still allocates in heap memory (no Scoped or Immortal memory use)
  - Used in conjunction with JVM’s Real-time garbage collector to maximise performance (minimize jitter, maximize throughput)
- Soft Real-time mode make it much easier to create Java applications with very predictable behaviour without having to always use the advanced RTSJ features
  - Helps preserves one of key benefits of the Java programming language, that of “ease of use”
- Subsequently we have made it possible to run multiple different ORB instances on the same JVM
  - A user can now start either a “non” Real-time ORB or a “soft” Real-time ORB or a “hard” real-time ORB, or all three types of ORB concurrently within the same application process

Maximizing Legacy Code Re-use

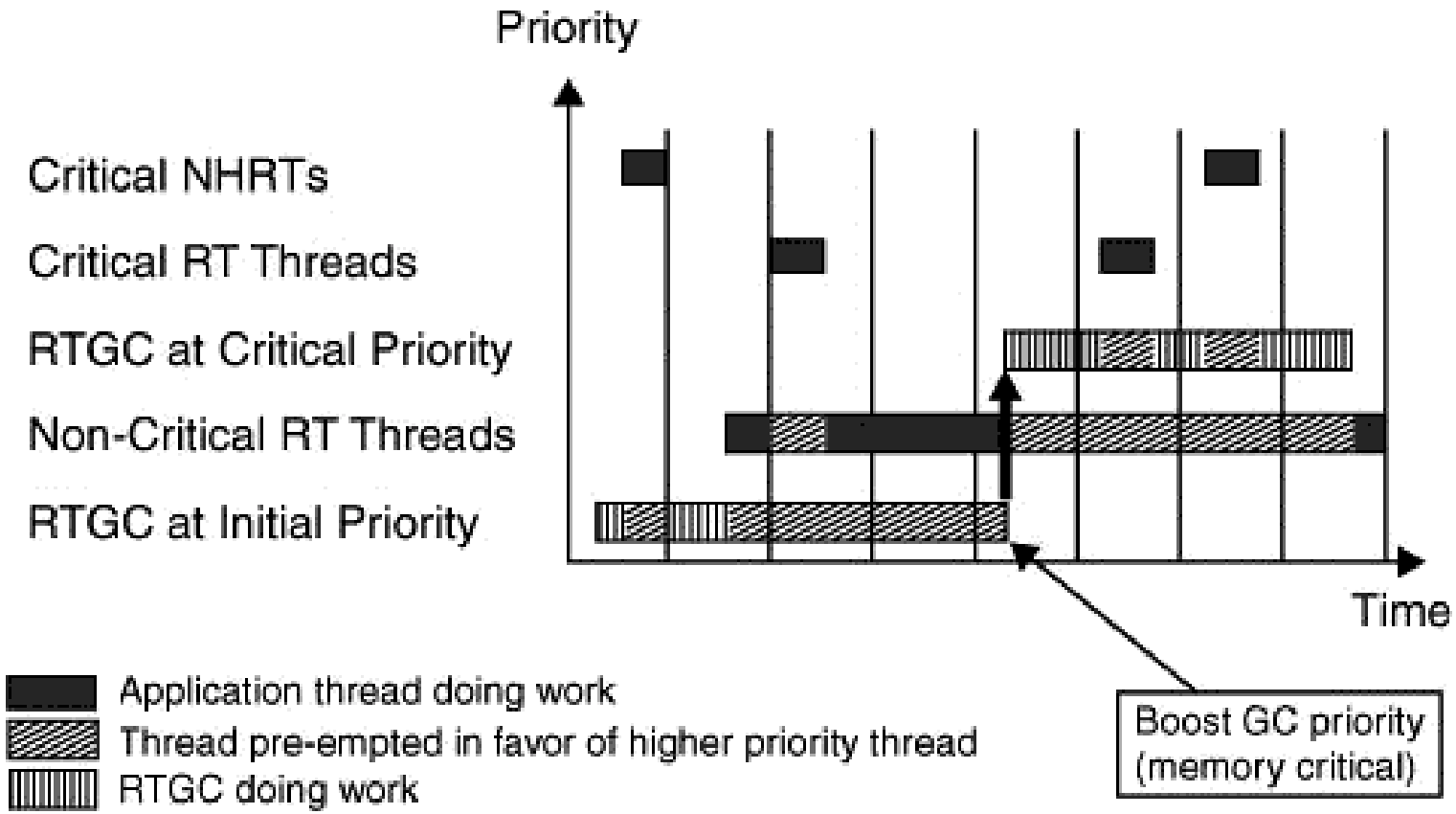
- Typically legacy applications written in Java have not been written with RTSJ in mind
- Highly unlikely that any legacy Java libraries can be used safely with NHRTs even within a scope
- Suggested strategy for inclusion of legacy code within a Real-time CORBA application written in Java:
  - Hard Real-time CORBA threads of execution will have to be written very carefully from scratch
  - Legacy CORBA code can be ran from Real-time Java threads for threads of execution that require “soft” Real-time QoS and in conjunction with Real-time garbage collector
    - If legacy code spawns any Java threads then this code will have to be re-written to use Real-time Java threads otherwise the predictability of the systems will be lost
- By supporting multiple modes of ORB operation from within the same CORBA application process allows a legacy application to have RT capabilities incrementally added without necessarily having to re-write the application from scratch

Leveraging Real-time Garbage Collection

- Automatic garbage collection is one of the main contributors to jitter in a Java system
- A Real-time garbage collector is required to recycle memory without disrupting the temporal determinism of time critical tasks
- RT garbage collection not currently standardized
- Different Real-time JVMs provide different Real-time garbage collection strategies, including:
  - Work based Real-time collectors execute when allocations are performed, regardless of the priority of the threads that are executing
  - Time based Real-time collectors execute at specified times and at top priority
  - The latest Real-time collectors analyse the criticality of an applications threads based on their priority, they are fully concurrent and can be pre-empted at any time

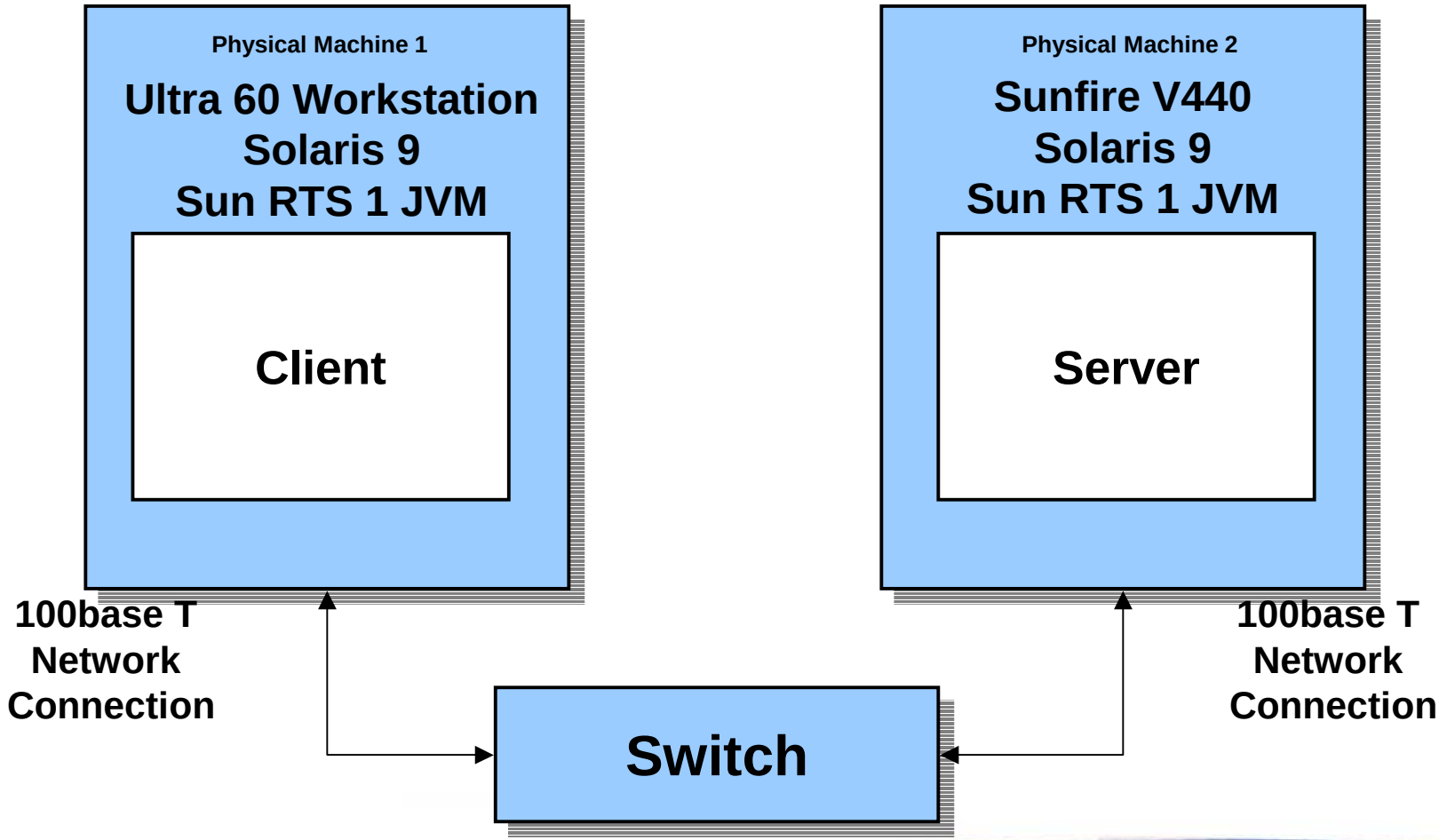
- OpenFusion RTOrb current supports two Real-time JVMs each with their own but different RT garbage collectors. These are:
  - Sun's RT Java System 2.0 (formally Mackinac)
  - IBM's Web Sphere Real-time v1.0 (formally J9)
- Web Sphere Real-time v1.0 – time based RT garbage collection that takes place at set intervals over a period of time
- RT Java System 2.0 – RT garbage collection algorithm analyses the criticality of an application's threads

Leveraging Real-time Garbage Collection – RTS 2.0



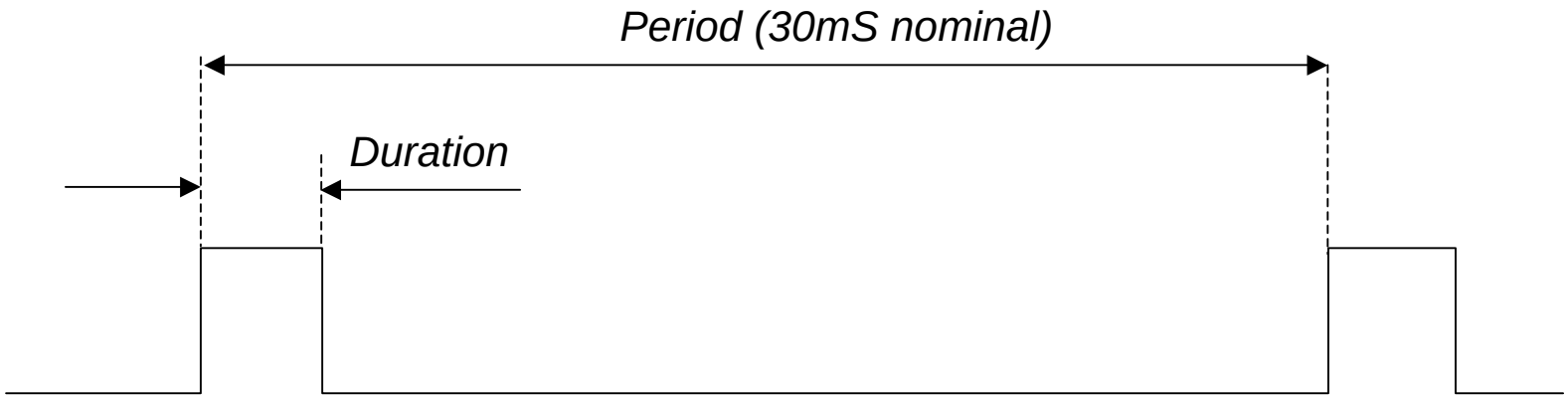
Test Configuration (1)

Networked client and server

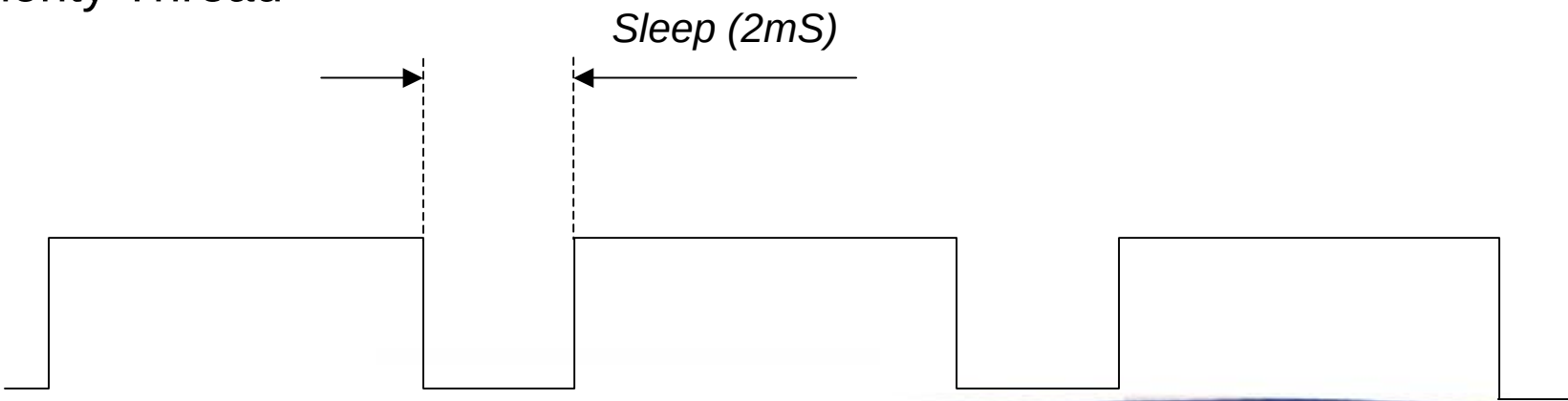


Test Application

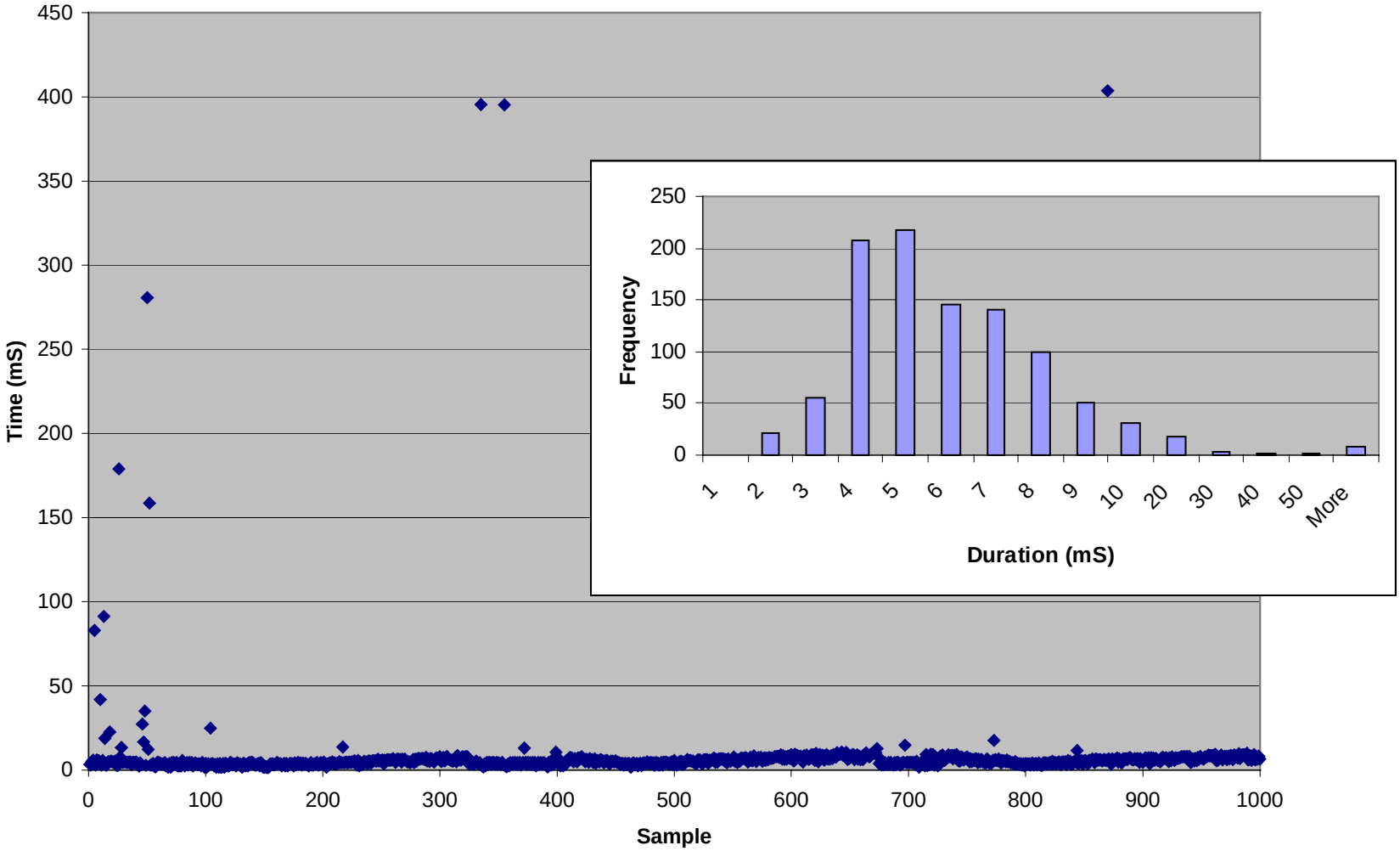
High Priority Thread



Low Priority Thread

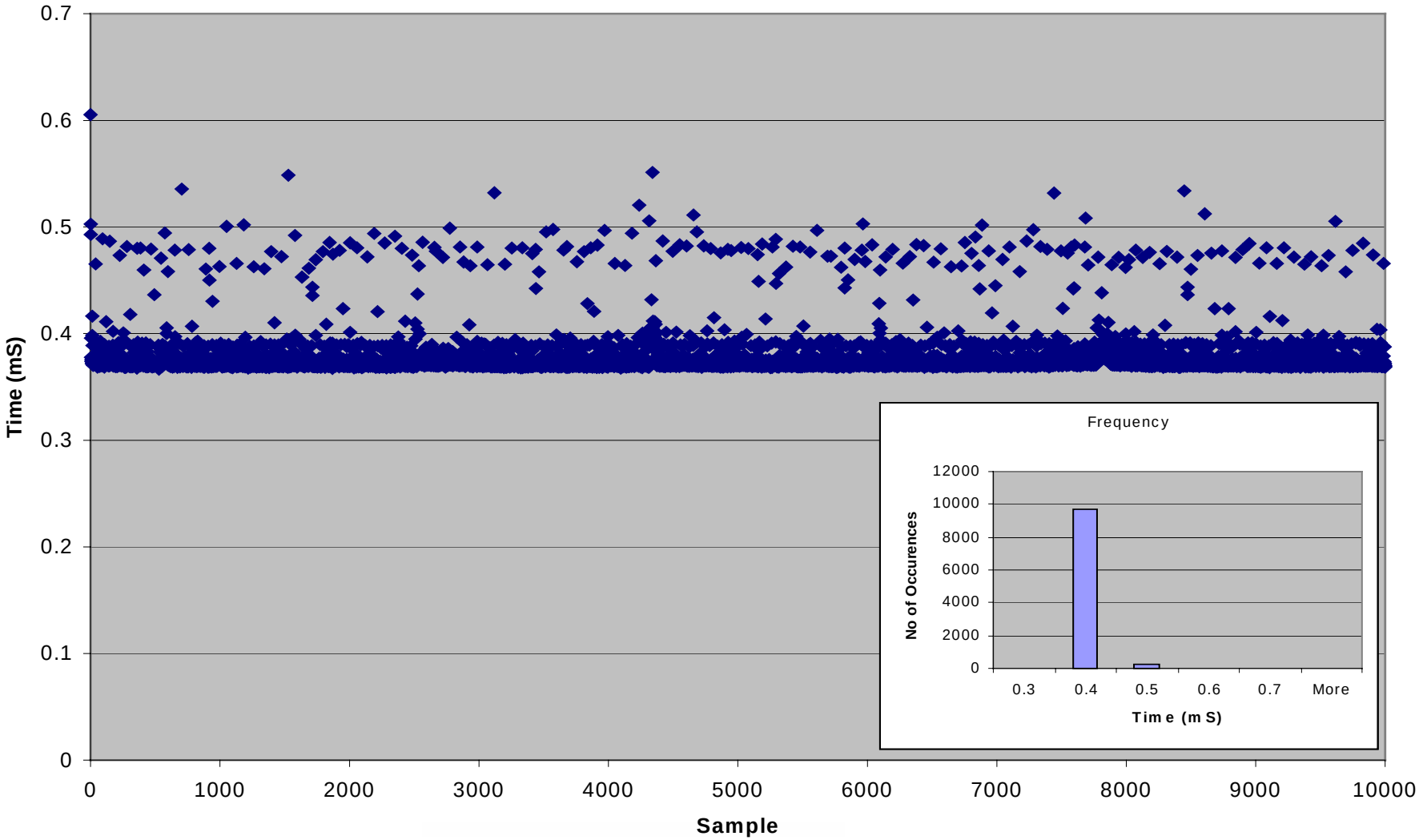


Non-Real-Time ORB and JVM



PRISMTECH
Productivity Tools & Middleware

C Client Server, 1 High Priority Thread

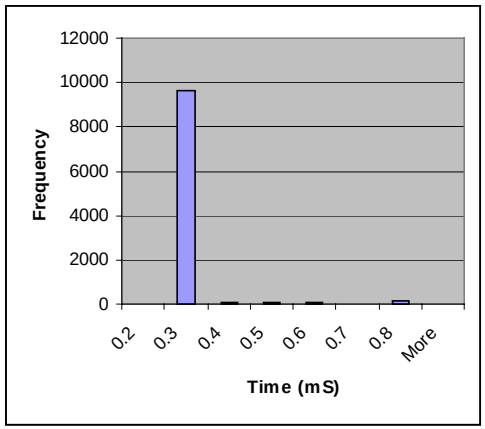


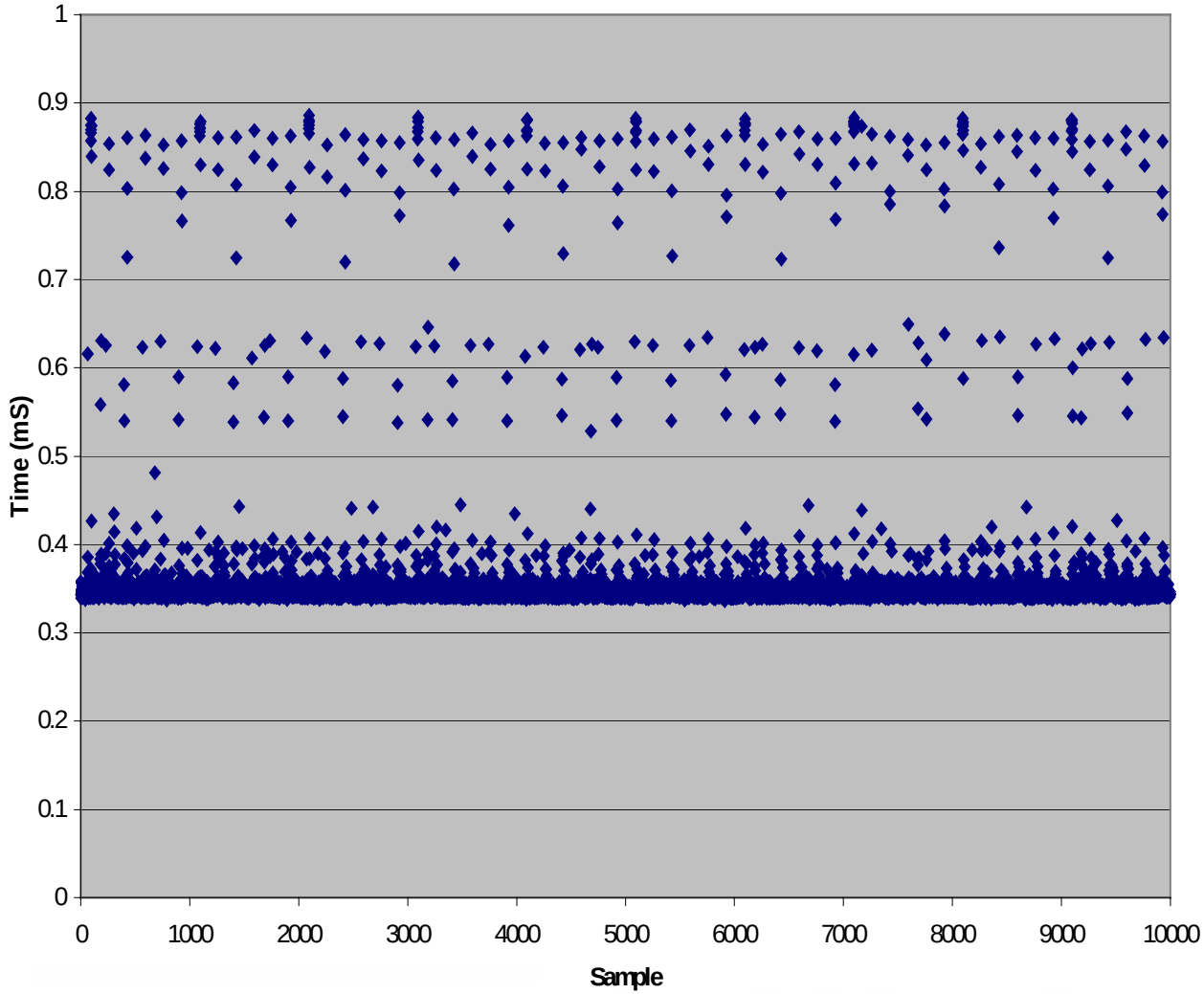
Co-located Java Sockets

Last 10K results of a 20K run

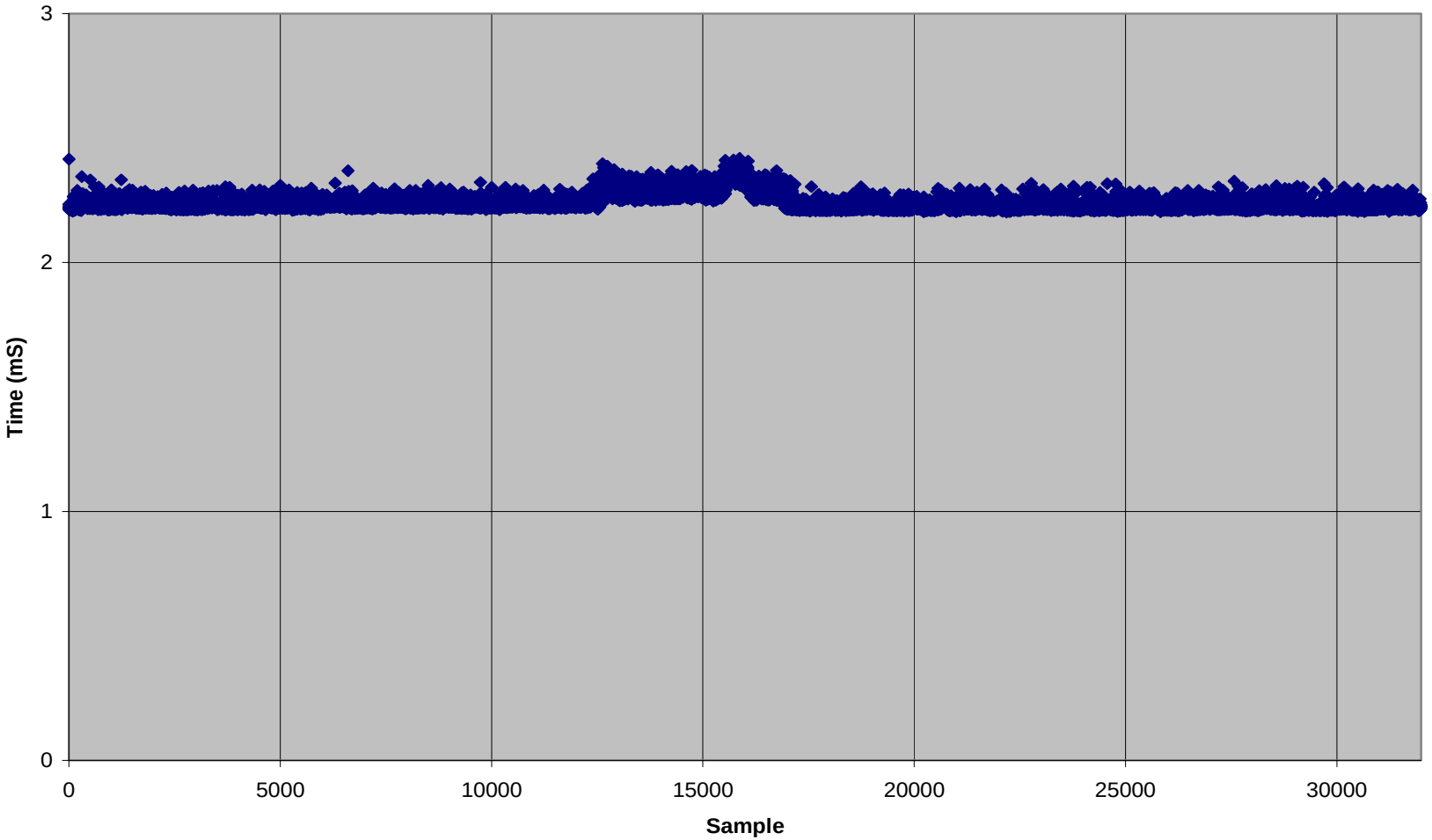
GC logged over run – only minor events recorded

All results within 0.6mS



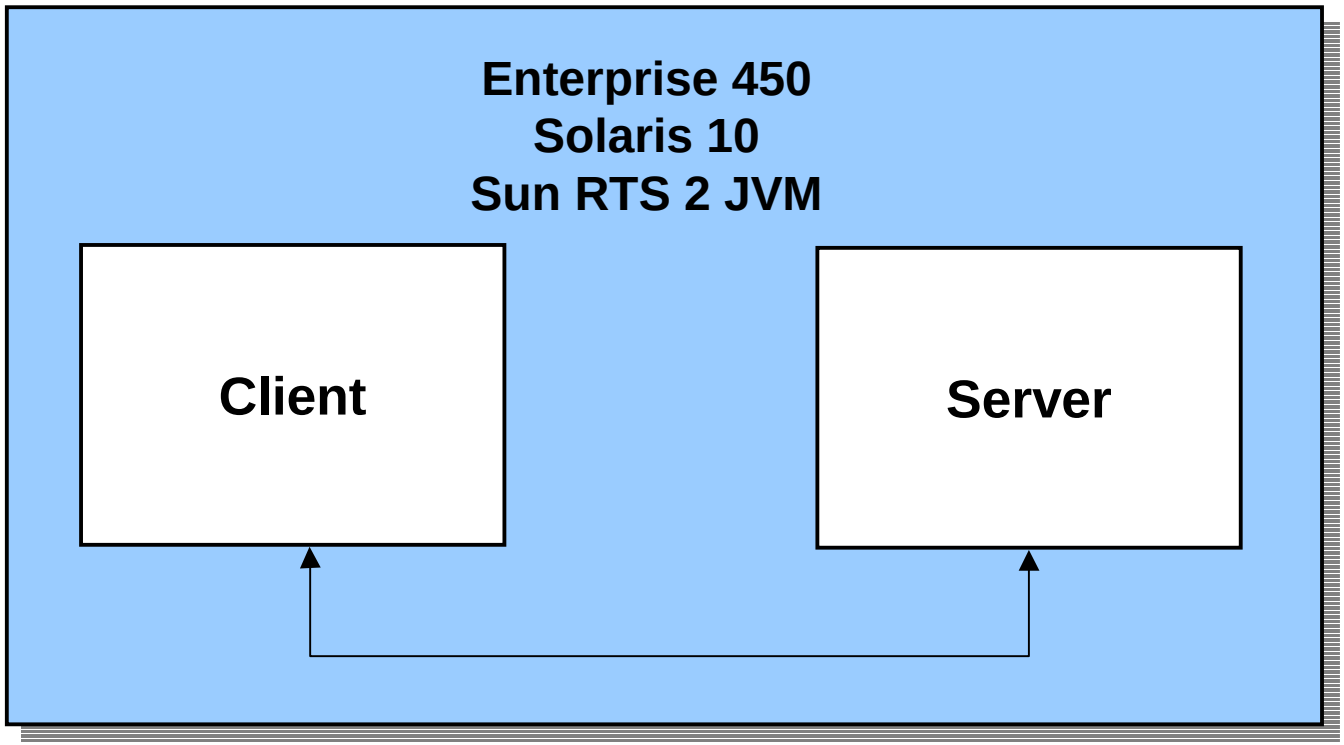


RTOrb Java Provides Excellent Predictable QoS (Jitter < 1mS)

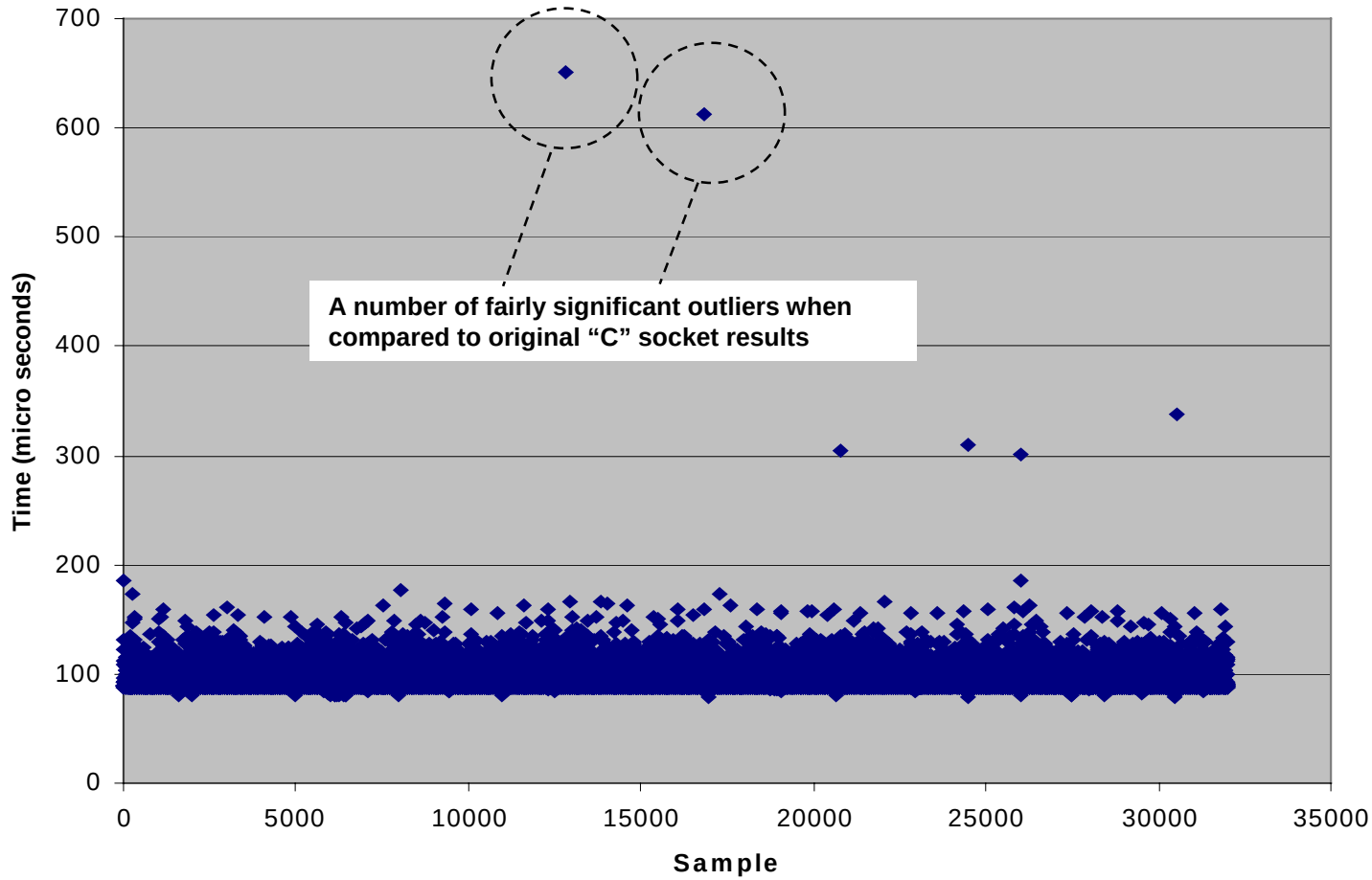


Results showing the roundtrip CORBA call invocation times for a high priority thread with one or more low priority threads running concurrently

Client and server running on different CPUs on the same server

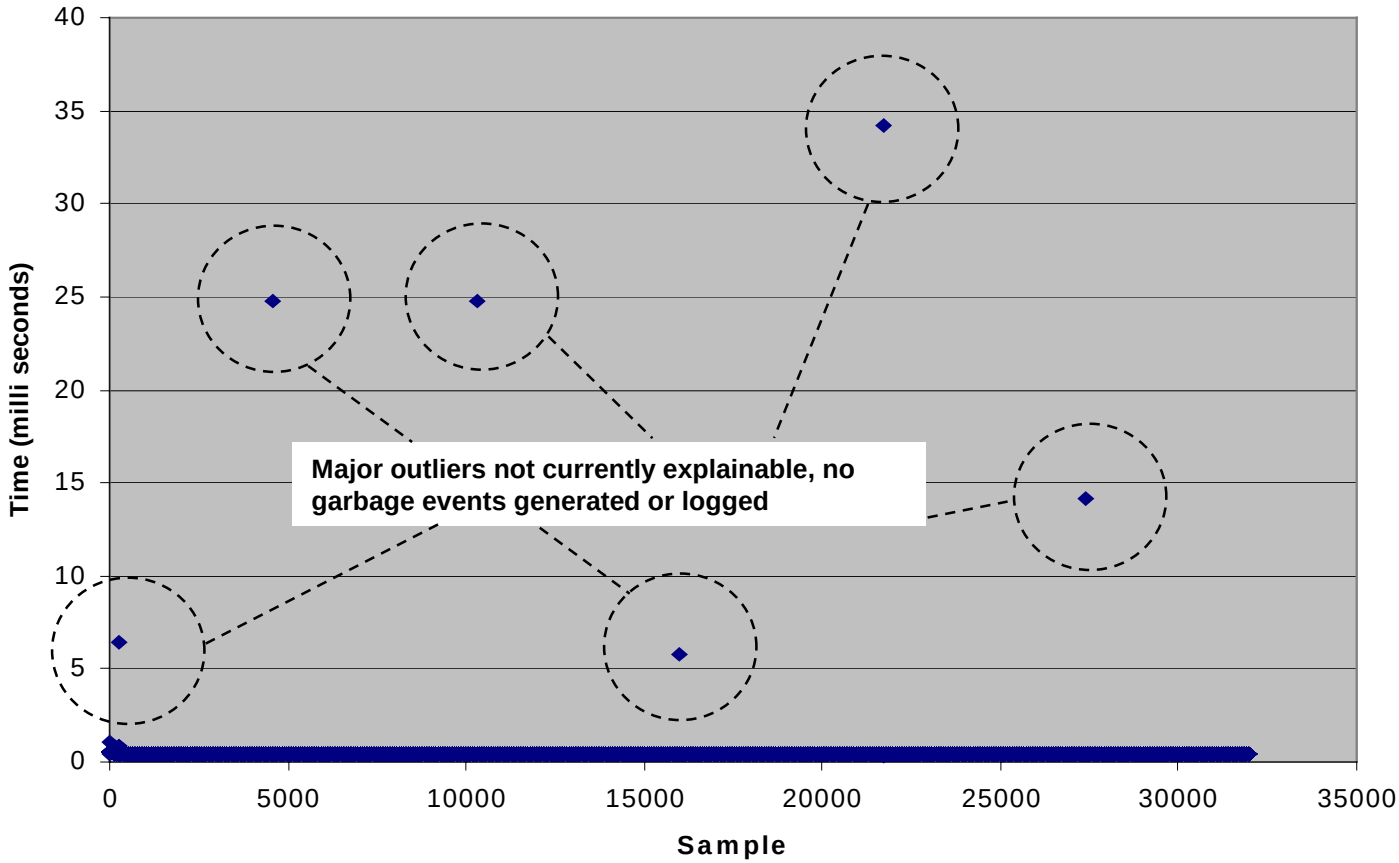


C Client Server, 1 High, 1 Low Priority Thread



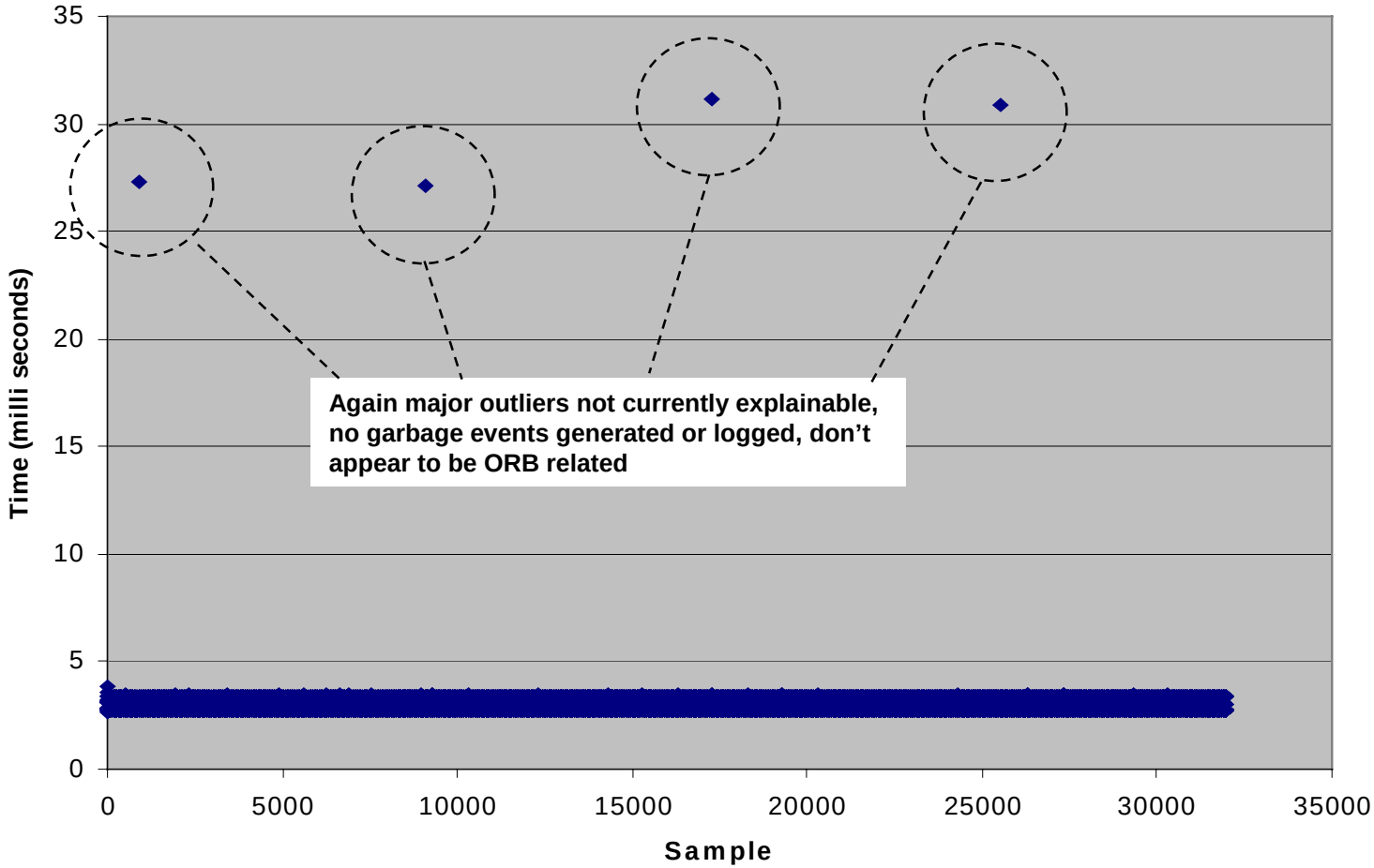
Results showing the roundtrip C socket call invocation times for a high priority thread with no other threads running concurrently. Results show a number of random outliers, worst case jitter value .7 mS

Java Sockets 1 High, 1 Low Priority NHRT Threads



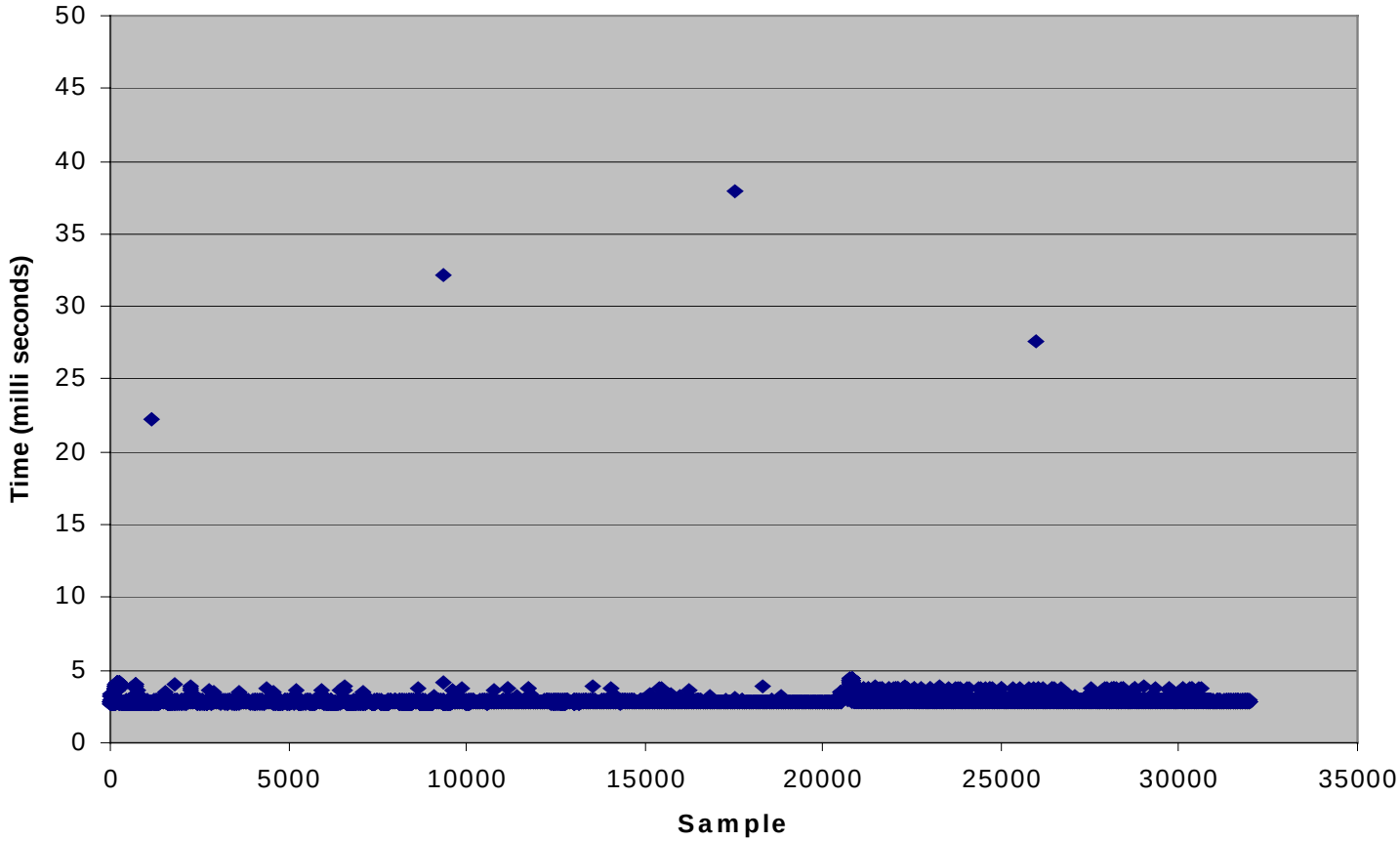
Results showing the roundtrip Java socket call invocation times for a high priority NHRT with one low priority NHRT running concurrently. Results show a number of significant outliers with a fairly consistent interval between, worst case jitter value 35 mS

RTOrb 1 High, 1 Low Priority NHRT Threads



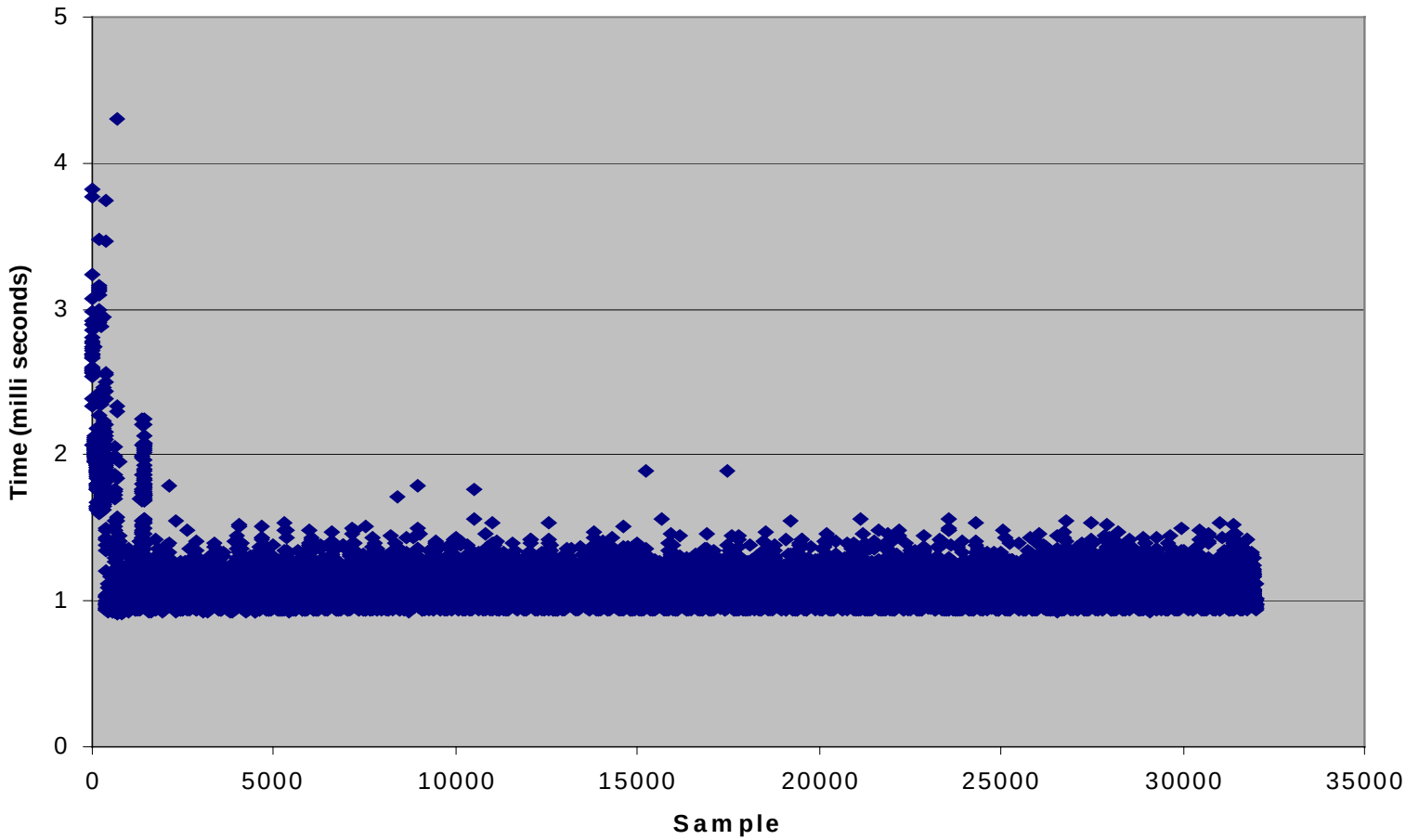
Results showing the roundtrip ORB call invocation times for a high priority NHRT with one low priority NHRT t running concurrently. Results show a number of significant outliers with a consistent interval between as in Java socket test, worst case jitter value 32 mS

RTOrb 1 High, 1 Low Priority NHRT Threads + 1 Garbage Creating Thread



Results showing the roundtrip ORB call invocation times for a high priority NHRT with one low priority NHRT and a garbage creating thread running concurrently. Results show a number of significant outliers with a consistent interval between as in previous tests. There is no correlation between major outliers and any garbage events.

RTOrb 1 High, 1 Low Priority RT Threads + 1 Garbage Creating Thread

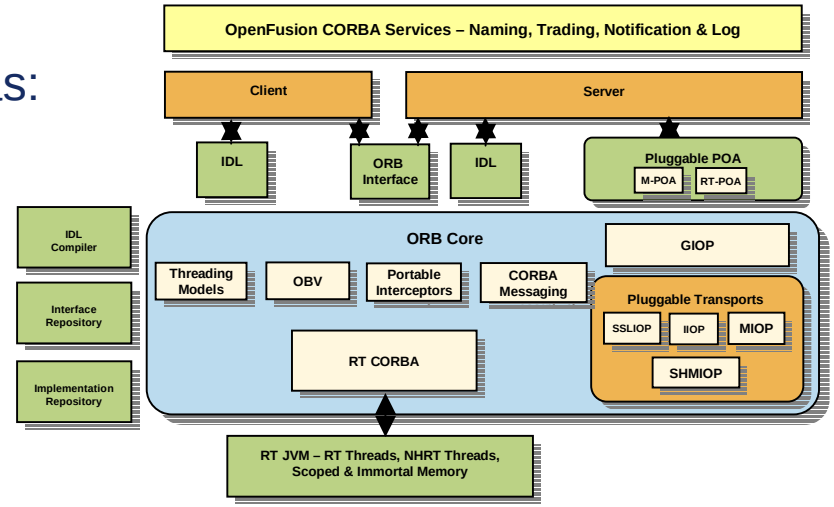


Results showing the roundtrip ORB call invocation times for a high priority RT Java thread with one low priority RT Java thread and a garbage creating thread running concurrently. Results show a number outliers within the first few hundred samples, however major outliers not present as in previous tests.

Feature Summary & Platform Availability

Key Features:

- CORBA 3.0 ORB - including features such as:
  - IDL to Java compiler
  - GIOP 1.3
  - Full POA
  - Dynamic ANY
  - Portable Interceptors
  - CORBA Messaging (AMI, QoS)
  - Value Types
- Support for OMG's Real-Time CORBA v1.2 specification - including features such as:
  - RT ORB, RT POA, RT Priority & Mappings, RT Current, RT Priority Model – Client propagated & Server declared, Priority Transforms, Mutex , Thread Pools, PrivateConnection Policy, Invocation Timeout



- **Extensible Transport Framework**: multi transport plug-in support for transports such as TCP/IP, UDP, Shared Memory, Unix Domain Sockets etc.
- **Common Object Services**: OpenFusion Naming, Trading, & Notification Services

Availability:

- JVM - Sun RT Java System v1.0, 2.0 (shortly), Web Sphere Real-time v1.0, JDK 1.4, 1.5
- Operating System Support - Solaris 10, Linux, LynxOS (planned)

OpenFusion™ RTOrb Java – Key Benefits

Benefits:

- **An ORB specifically designed to support distributed Real-time programming in a Java environment**
- For the first time enables Real-time system developers to leverage the benefits of the Java programming model – ease of use, write once run anywhere, portability
- Provides a single ORB solution for systems with a range of different Real-time QoS requirements (non Real-time, soft Real-time & hard Real-time)
- Developers only have to learn how to use one ORB
- Highly configurable – "mixed mode" allows a single application process to support non real-time, soft real-time & hard real-time threads of execution
- Can transparently improve the responsiveness of a wide range of Java applications, including legacy applications with minimal code changes & without having to be a real-time expert
- Single ORB solution minimizes ORB interoperability issues
- Open & configurable architecture
- High performance – excellent latency & throughput characteristics, as good or better than other Java ORBs
- Highly portable, scalable, flexible, & reliable
- Standards compliant (CORBA 3, RT-CORBA 1.2, & RTSJ 1.0)
- Guaranteed interoperability with PrismTech's other CORBA products including TAO & JacORB

Major Users:

- **Defense**: Raytheon (DDG 1000), Brazilian Navy

Summary

- PrismTech continuing to improve the RTOrb Java implementation as both Real-time JVM technology and our knowledge of the problem space evolves
- By using memory management techniques such as Factories or object caching the Java language mappings can be improved to make it more suitable for use in Real-time CORBA systems
- In order to maximize re-use of legacy code it is important that the ORB can also support a mode of operation that allows a combination of Real-time Java threads and heap allocation, used in combination with a Real-time garbage collector
- Supporting multiple Real-time modes of ORB operations (non Real-time, soft Real-time and hard Real-time) and threads of execution from the same application process gives the application developer the maximum flexibility
- With RT JVM technology evolving steadily, leveraging automatic garbage collection can improve system predictability while maintaining the “ease-of-use” associated with the Java programming language

Contact Information?

For additional information please visit:

- **www.prismtech.com**

or contact Andrew Foster

- **awf@prismtech.com**