PRISMTECH

Distributed & Wireless Software Infrastructure

# Experiences in Developing a Real-Time Java Object Request Broker

John Russell, PrismTech

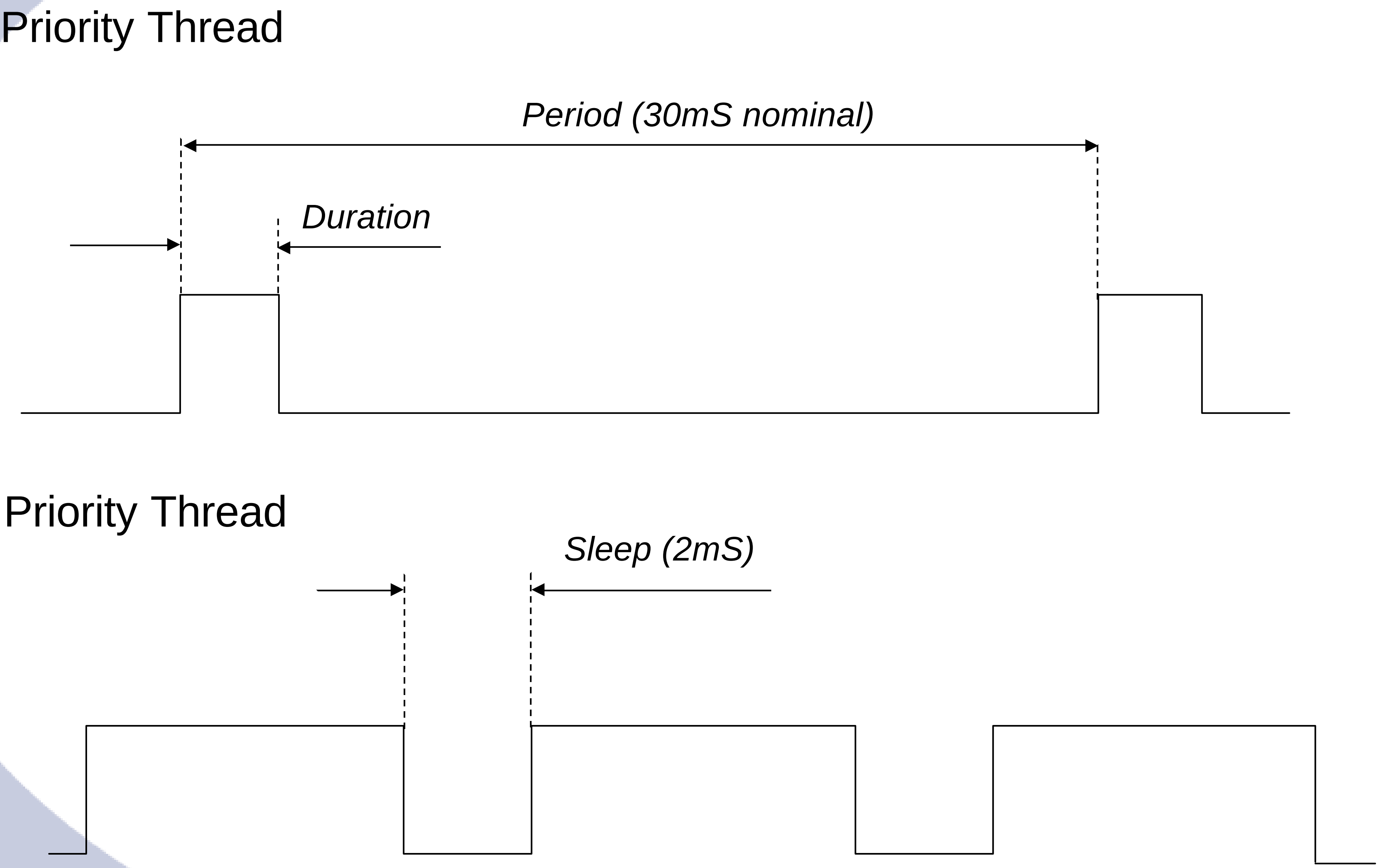
# Goals

- Demonstrate practicality of real-time CORBA in a Java environment
  - Mackinac, Solaris 9
- Acceptance criterion
  - Measured jitter <1mS in test environment

# Approach

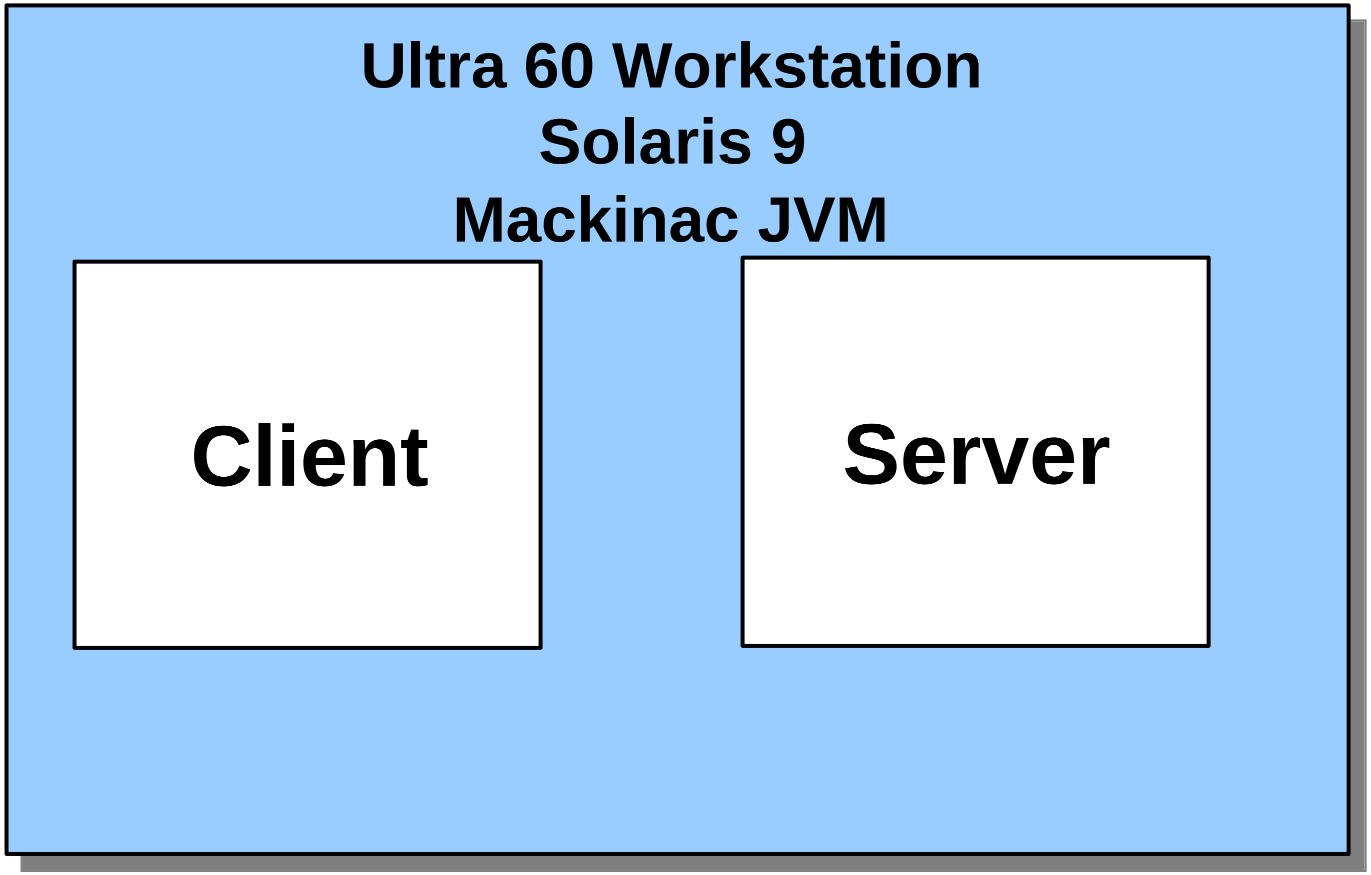
- An experimental approach with three phases:
  - Obtain benchmark figures from a C socket to socket example
  - Repeat socket example in Java
    - Isolate contributors to jitter
  - Repeat tests with OpenFusion RT for Java, PrismTech's real time Java ORB
    - Develop tests using RT threads, NHRT Threads, scoped and immortal memory

# Test Application



# Test Scenario 1: Hardware

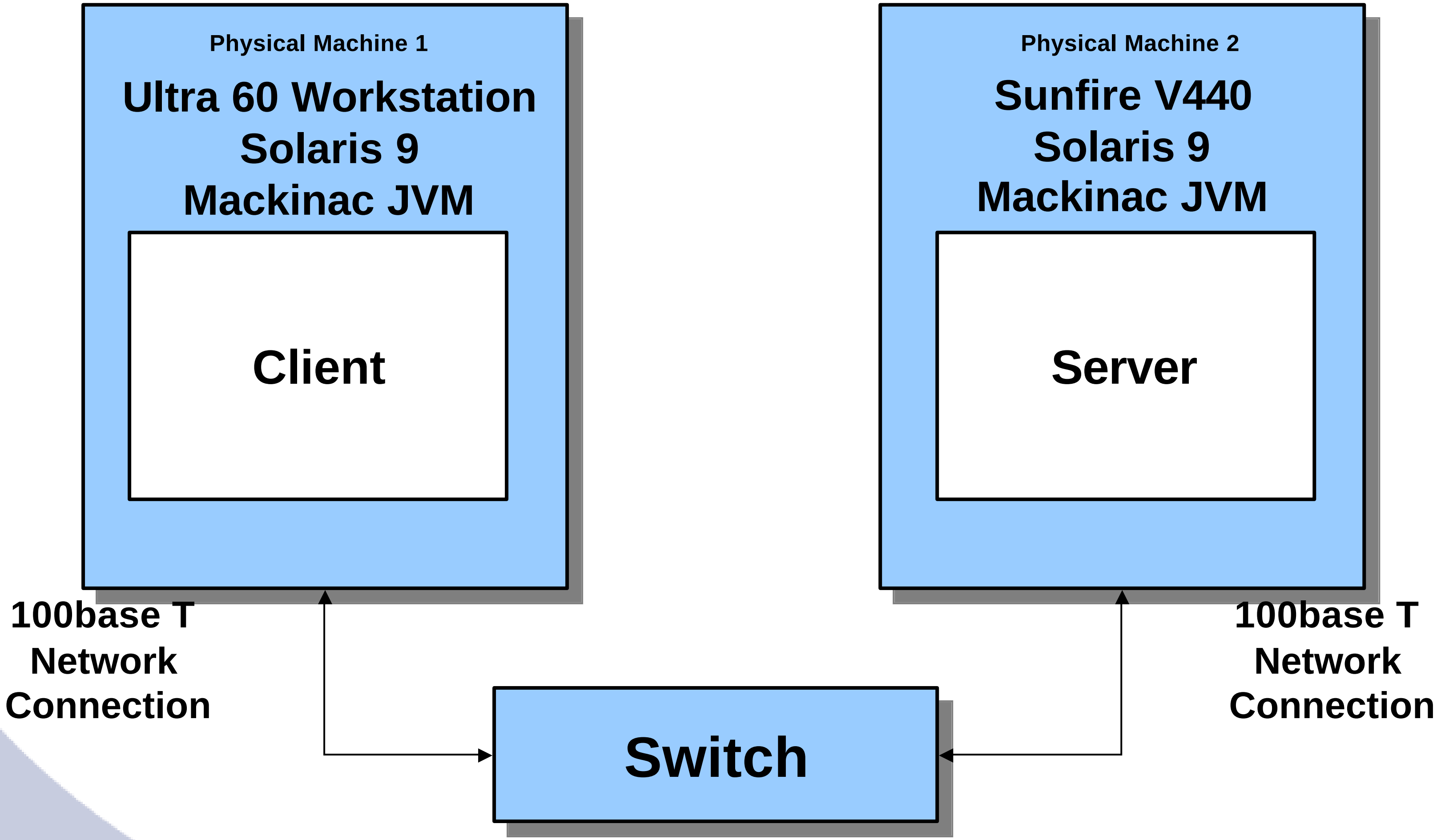
## Co-located client and server



Ultra 60 Workstation
Solaris 9
Mackinac JVM

# Test Scenario 2: Hardware

## Networked client and server



Physical Machine 1

Ultra 60 Workstation
Solaris 9
Mackinac JVM

Client

Physical Machine 2

Sunfire V440
Solaris 9
Mackinac JVM

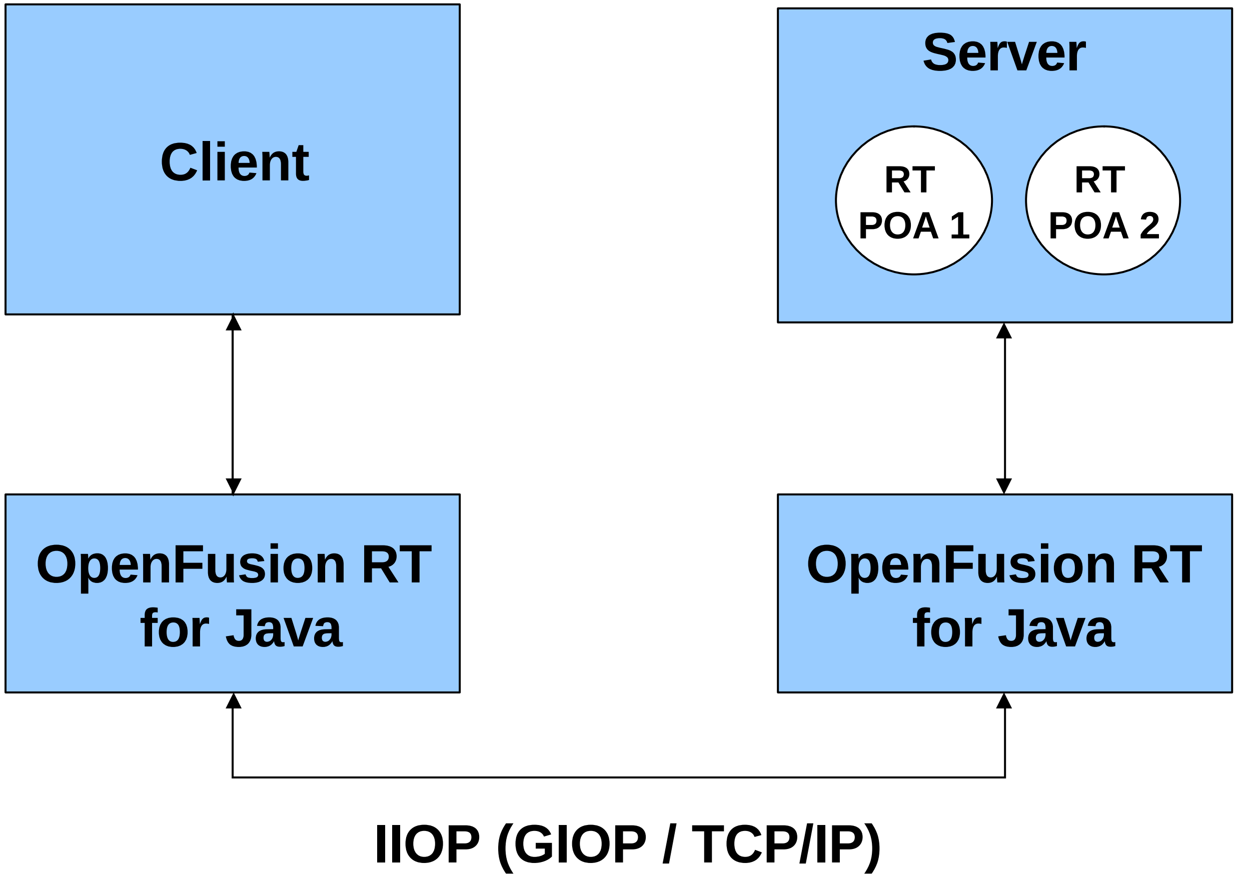
Server

100base T
Network
Connection

Switch

100base T
Network
Connection

# Test Scenario - Software



# Test Client

Low Priority Task

- Send Large Object (200 ints)
- Wait for Server response
- Sleep for 2mS
- Repeat until stopped

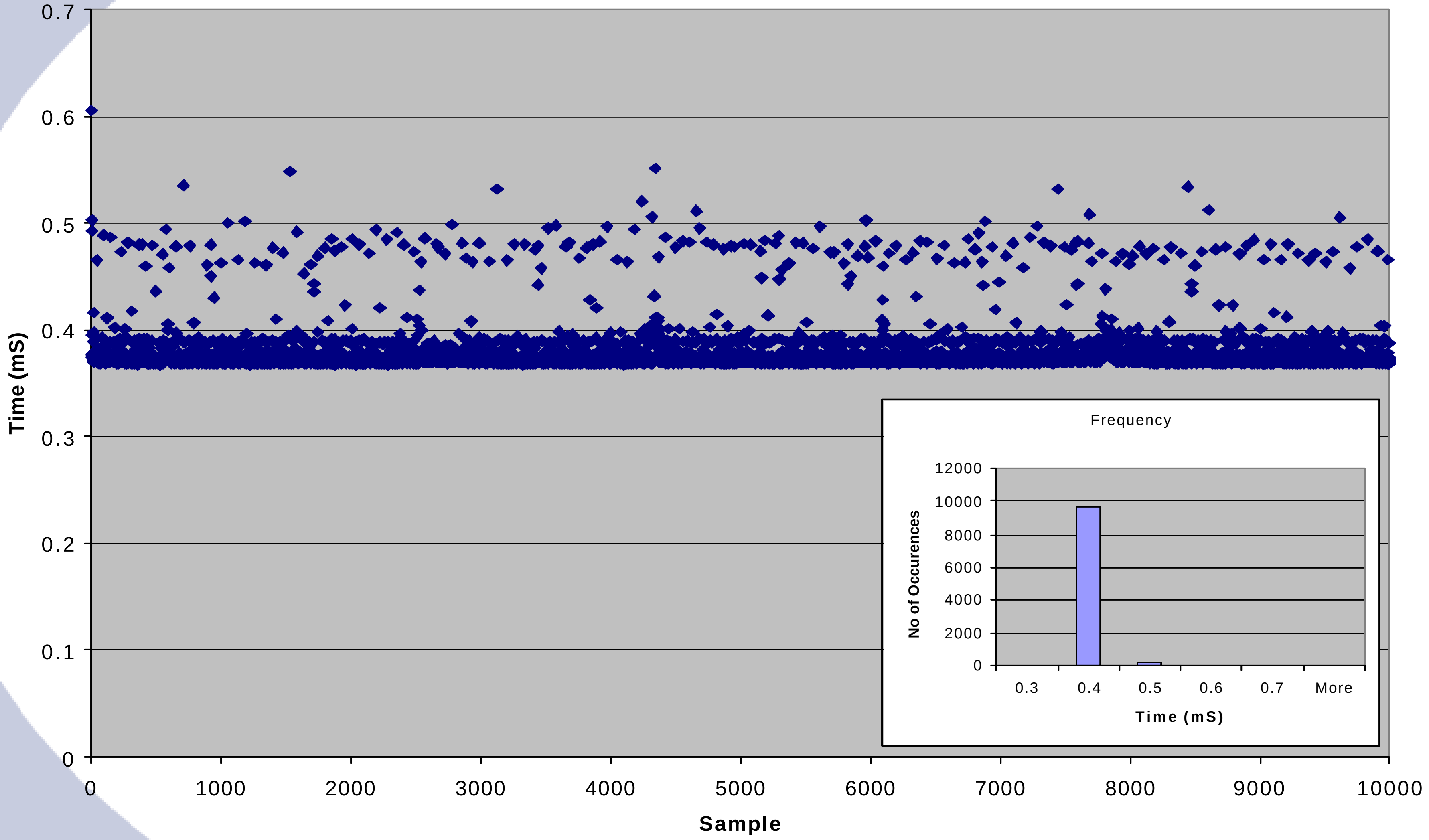
High Priority Task

- Send Small Object (10 ints)
- Wait for Server response
- Repeat every 30mS for 32000 cycles

Record start and end time of each cycle using real time clock

# RESULTS

# C Client Server, 1 High Priority Thread

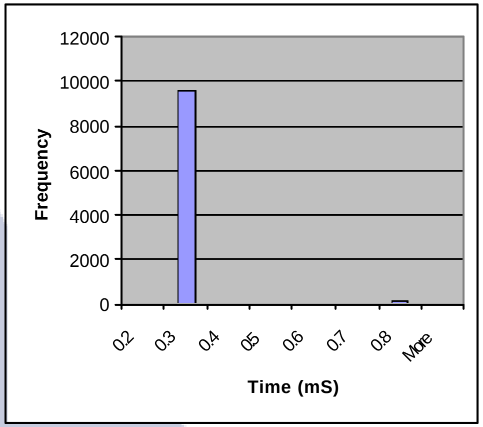


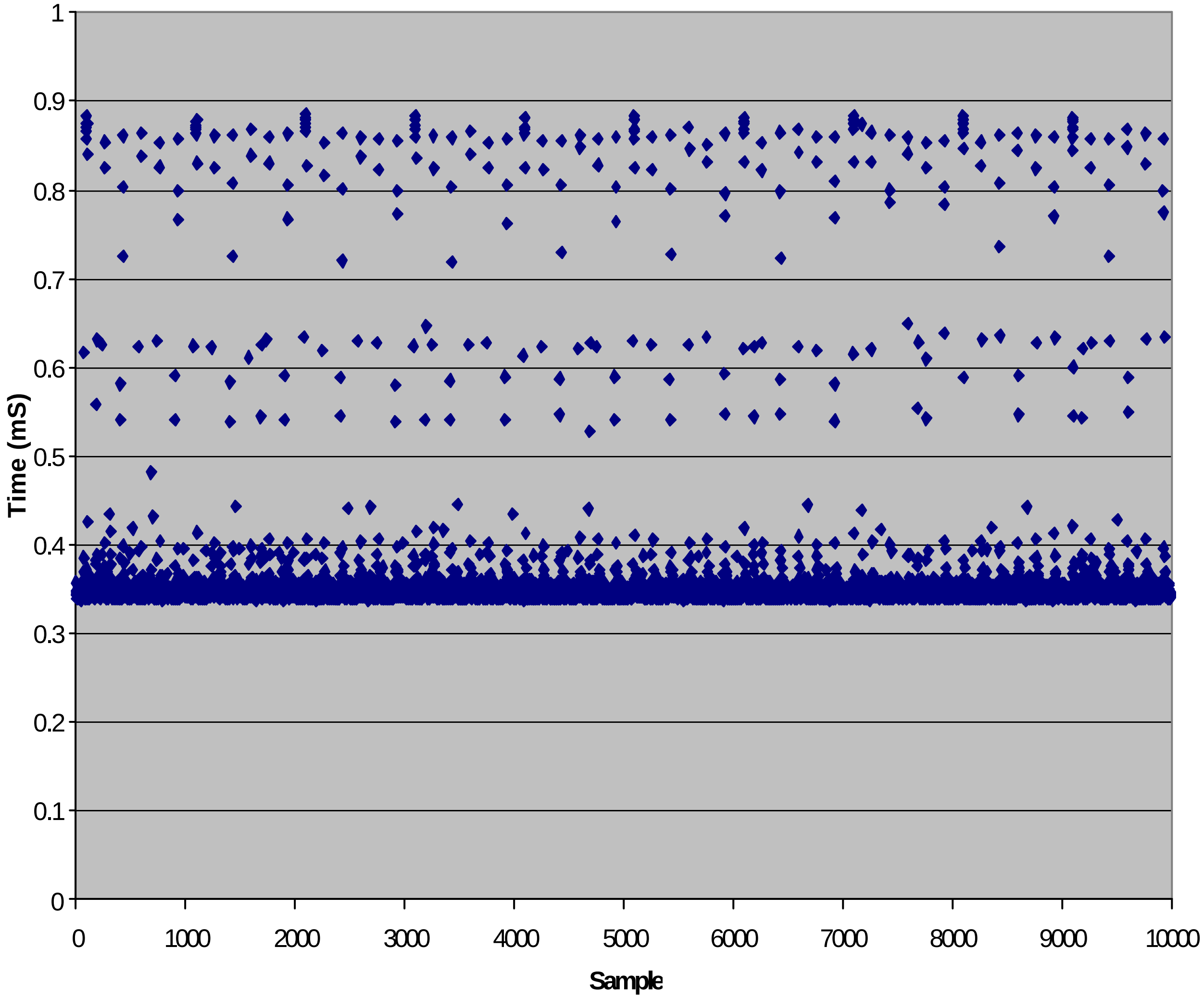
# Co-located Java Sockets

Last 10K results of a 20K run

GC logged over run – only minor events recorded

All results within 0.6mS





# Networked Java Sockets

Last 10000 samples from a 50000 sample run.

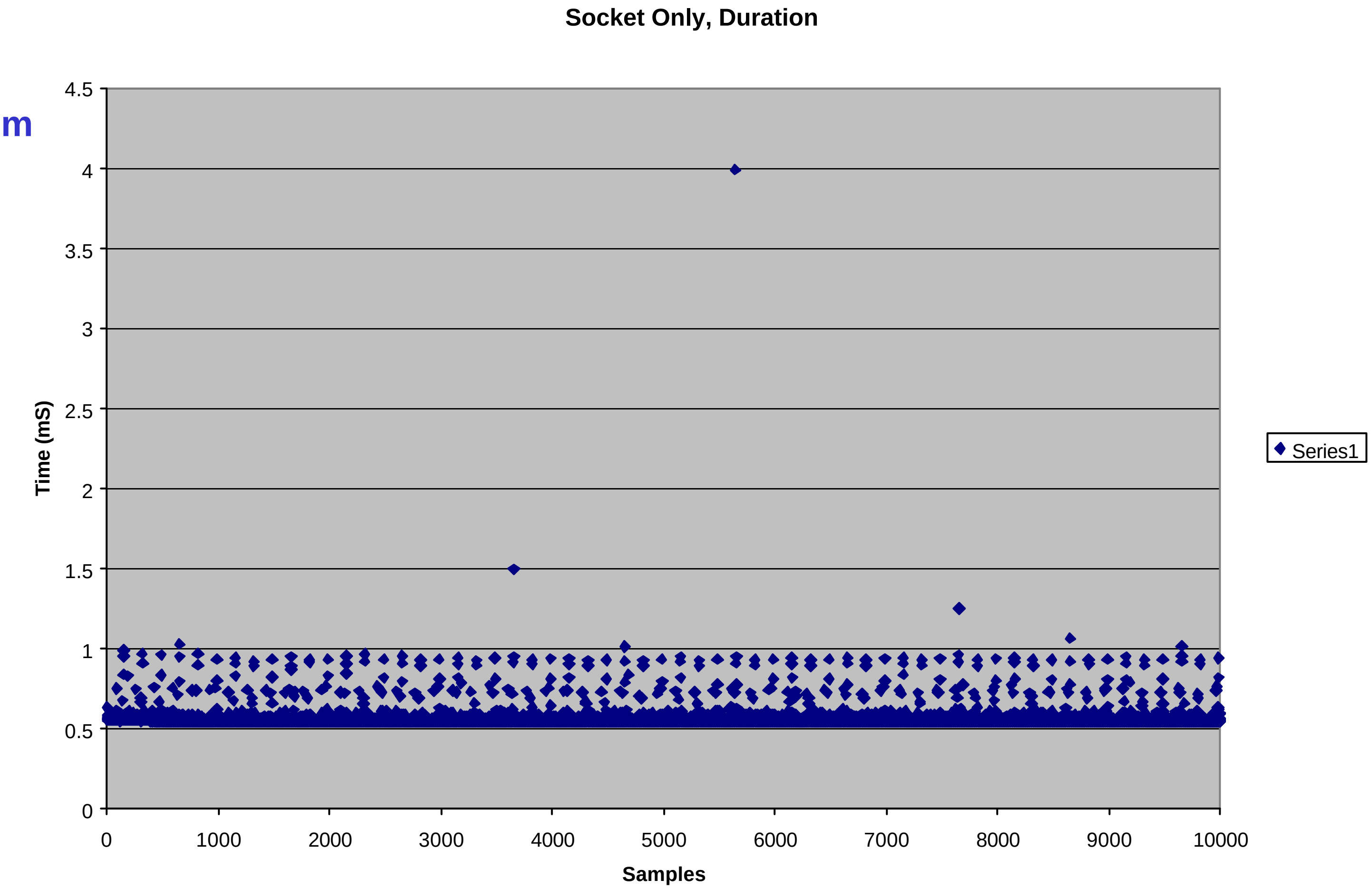
Heap size 256MB

No Garbage Collection recorded during entire run

95% of results within 0.1mS

Only 0.03% of results outside 0.5mS

Indicates what can be achieved



# Networked Sockets with Garbage Collection

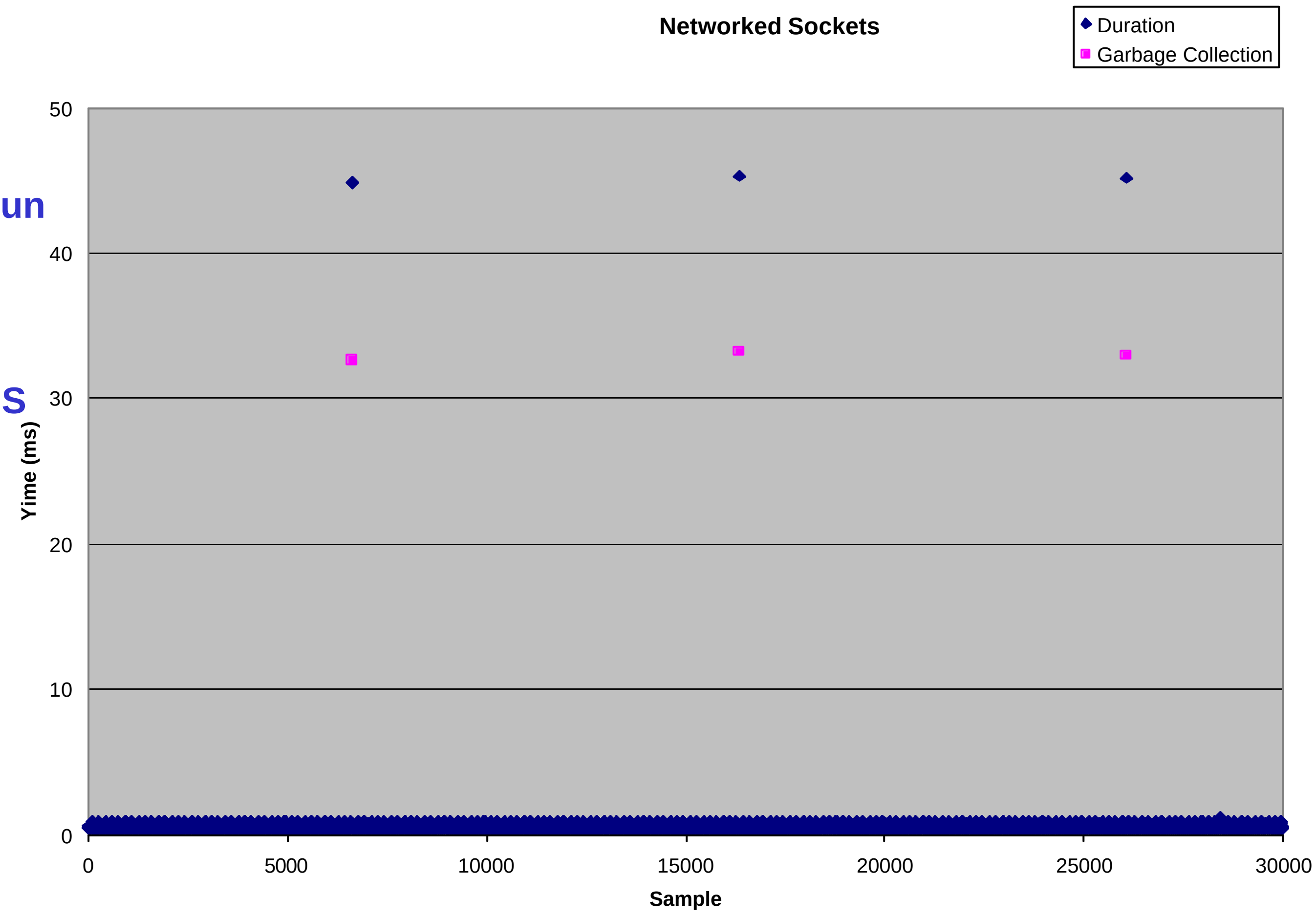
Last 30K samples of a 40K run

Heap size 8MB

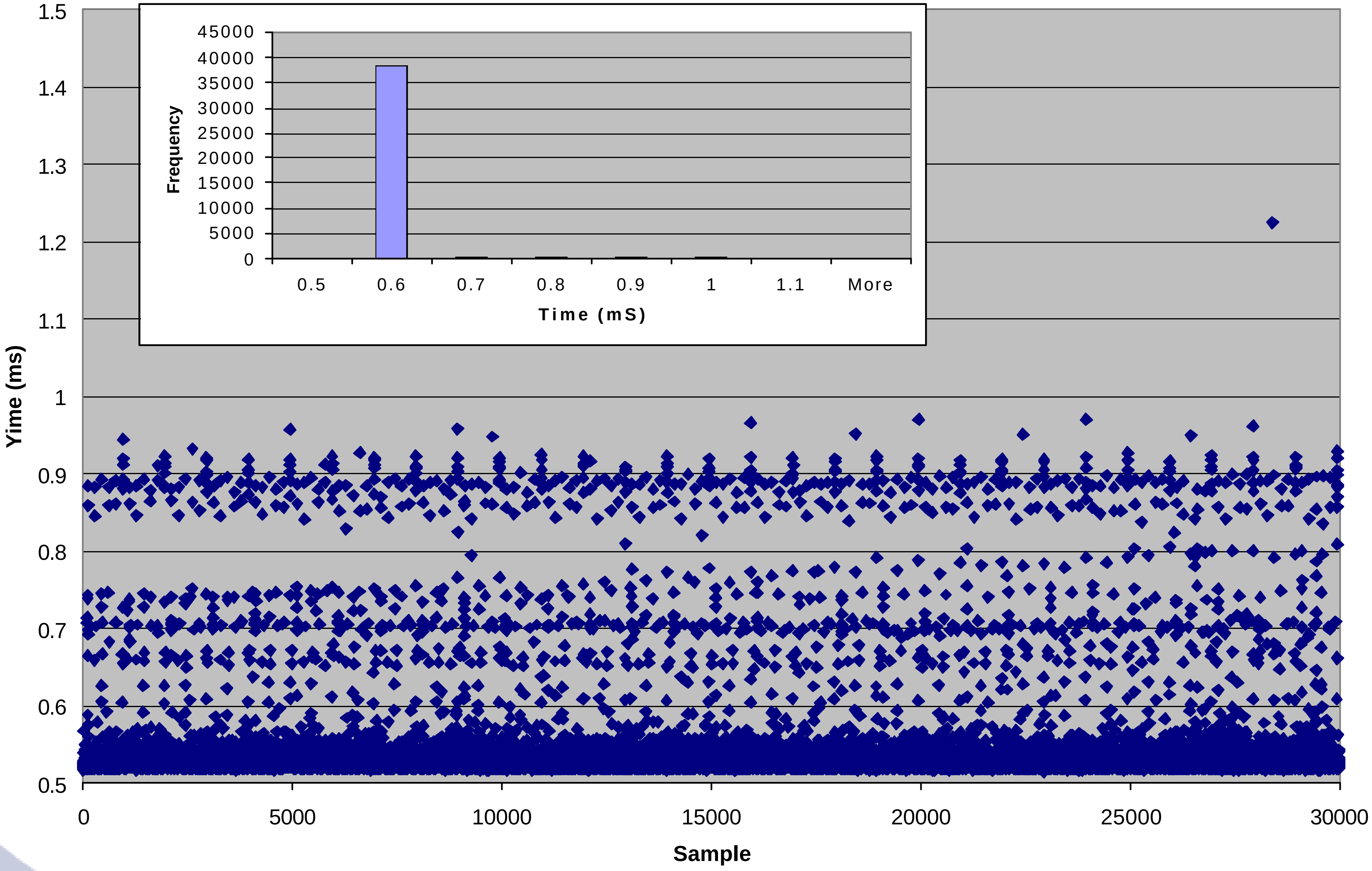
GC Logging enabled

96% of samples within 0.1mS

Only 0.01% outside 0.5mS



# Networked sockets - Expanded Scale

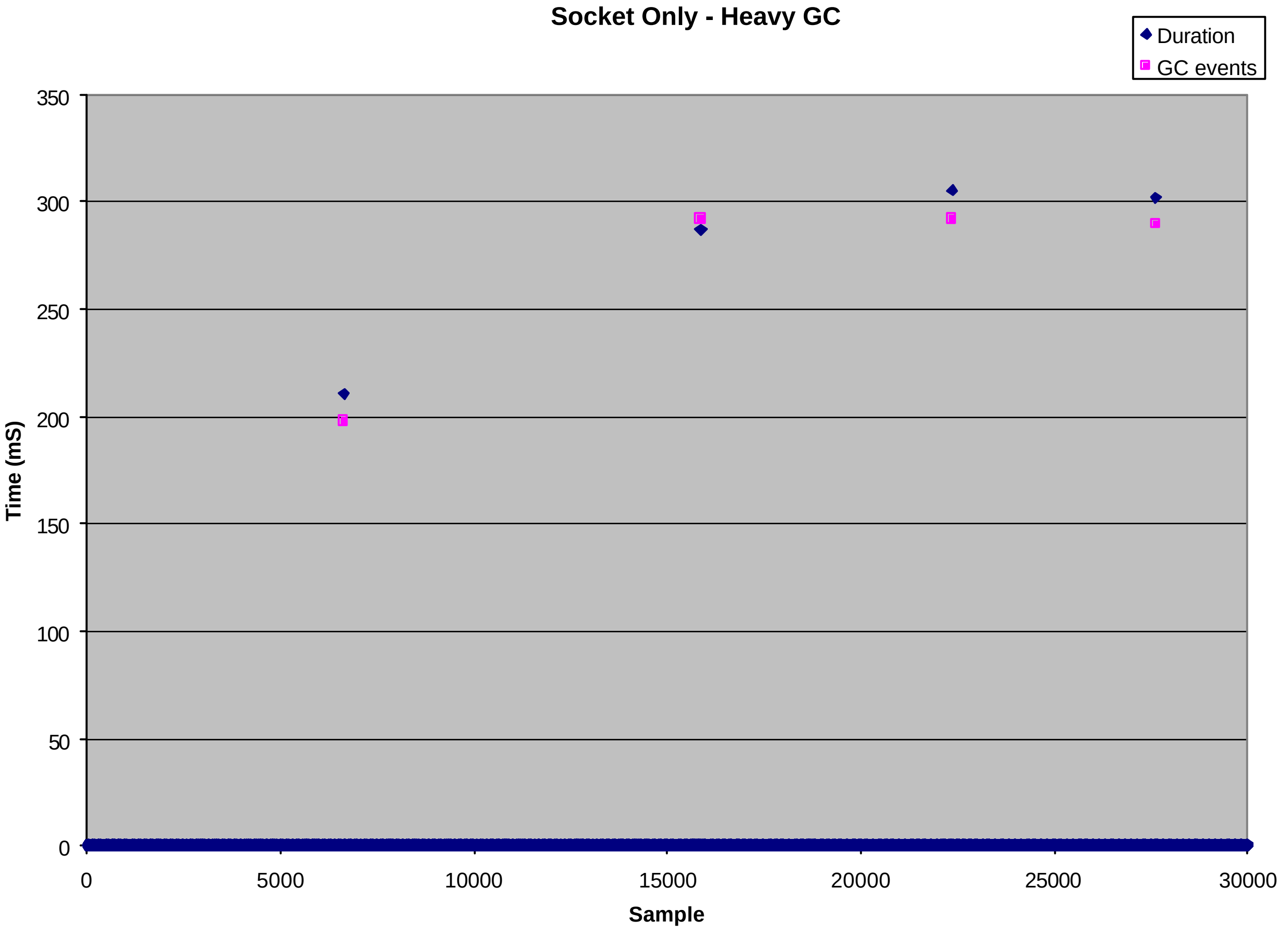


# Sockets and Heavy GC

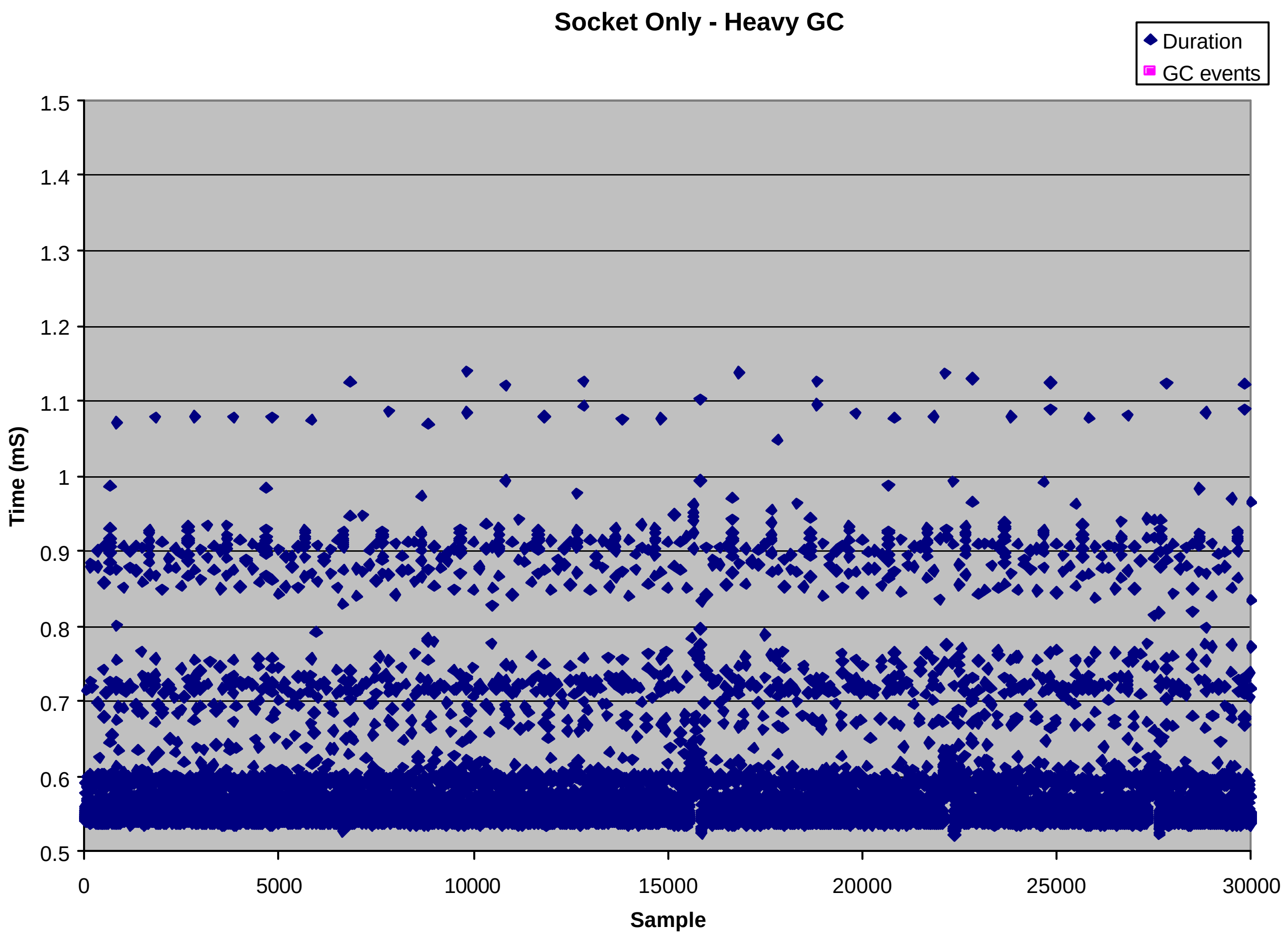
Last 30K samples of 40K run

Heap size 4MB

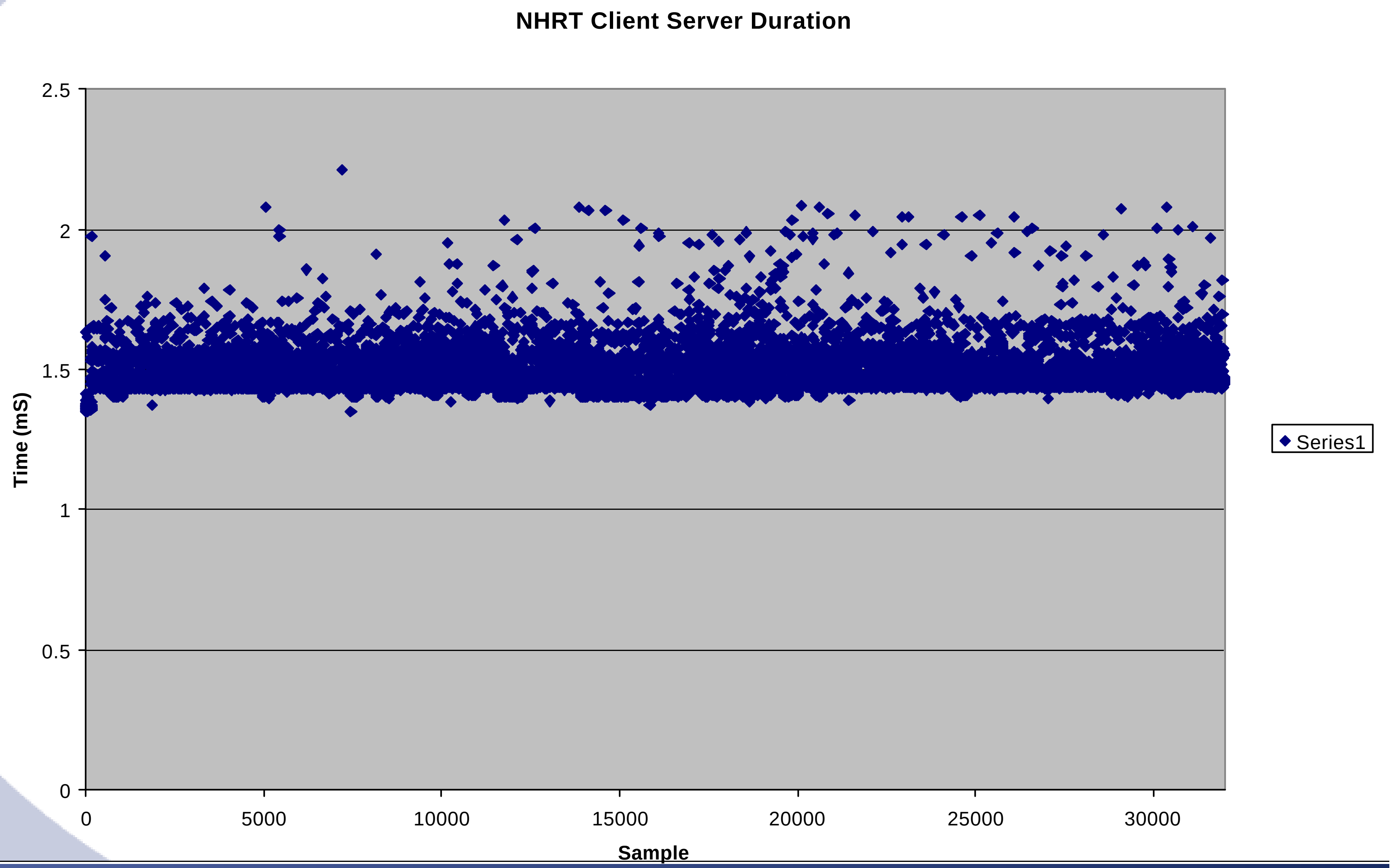
Dramatic increase in GC effects



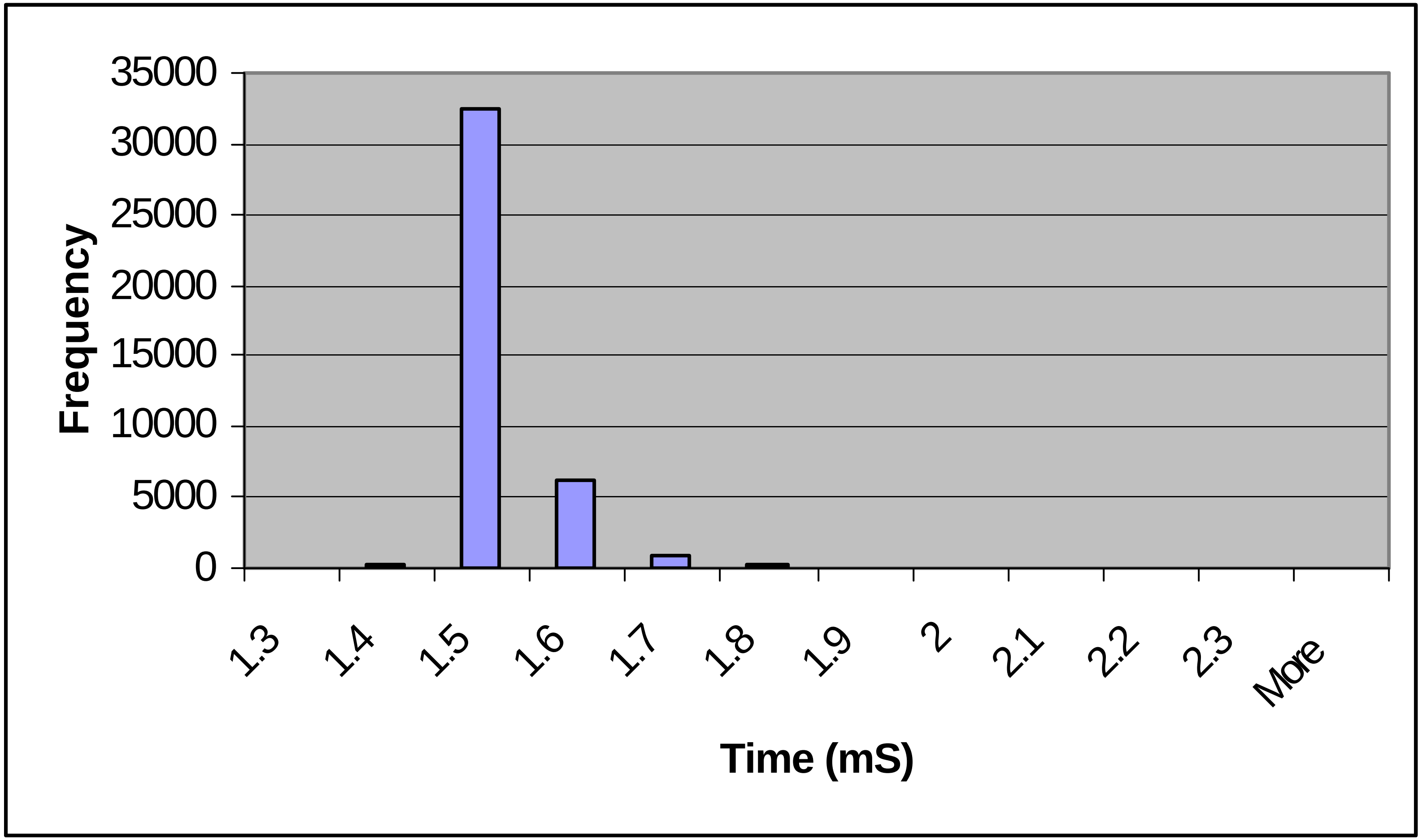
# Socket only, Heavy GC



# NHRT Client - Duration



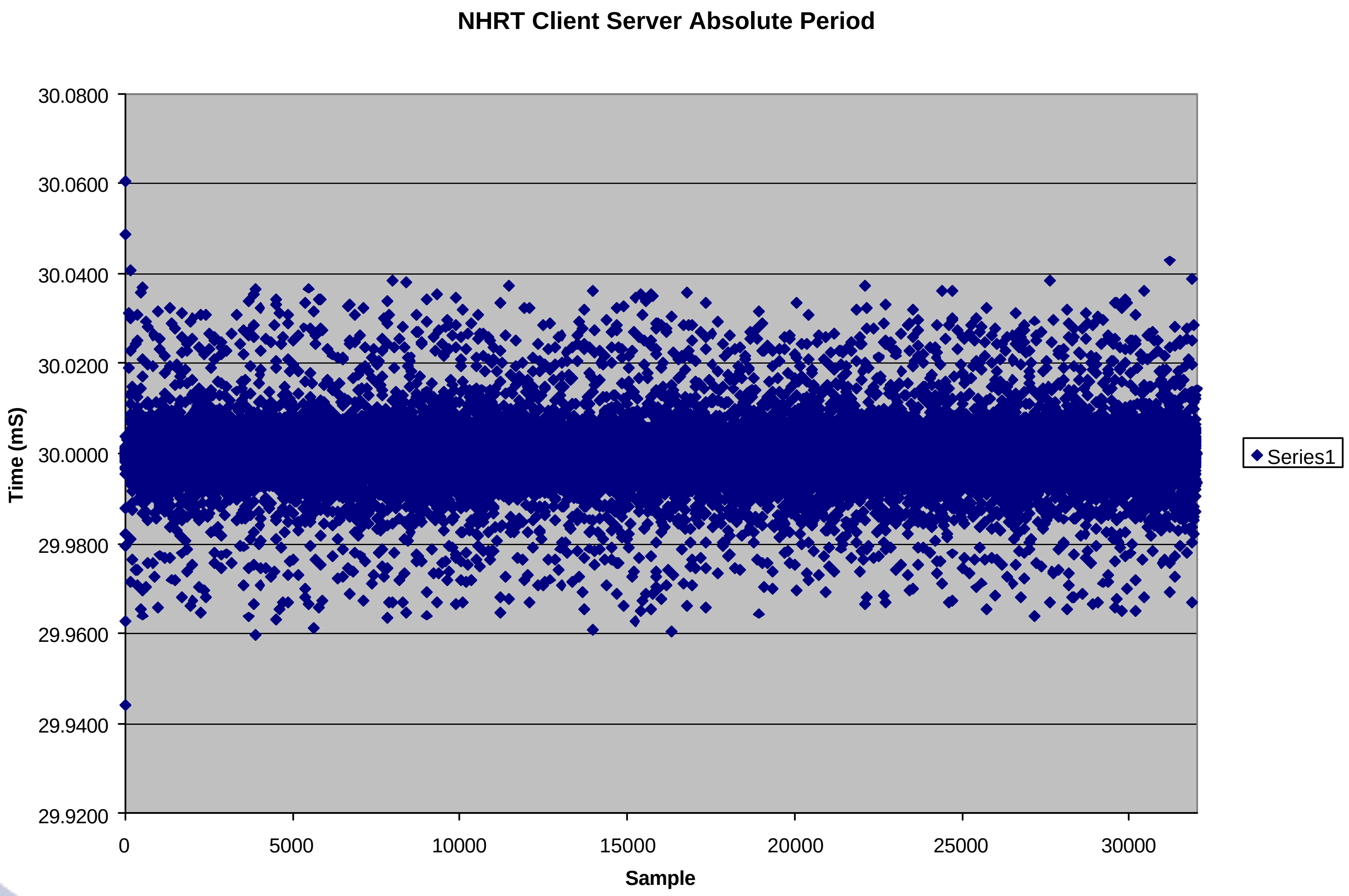
# NHRT Client - Statistics



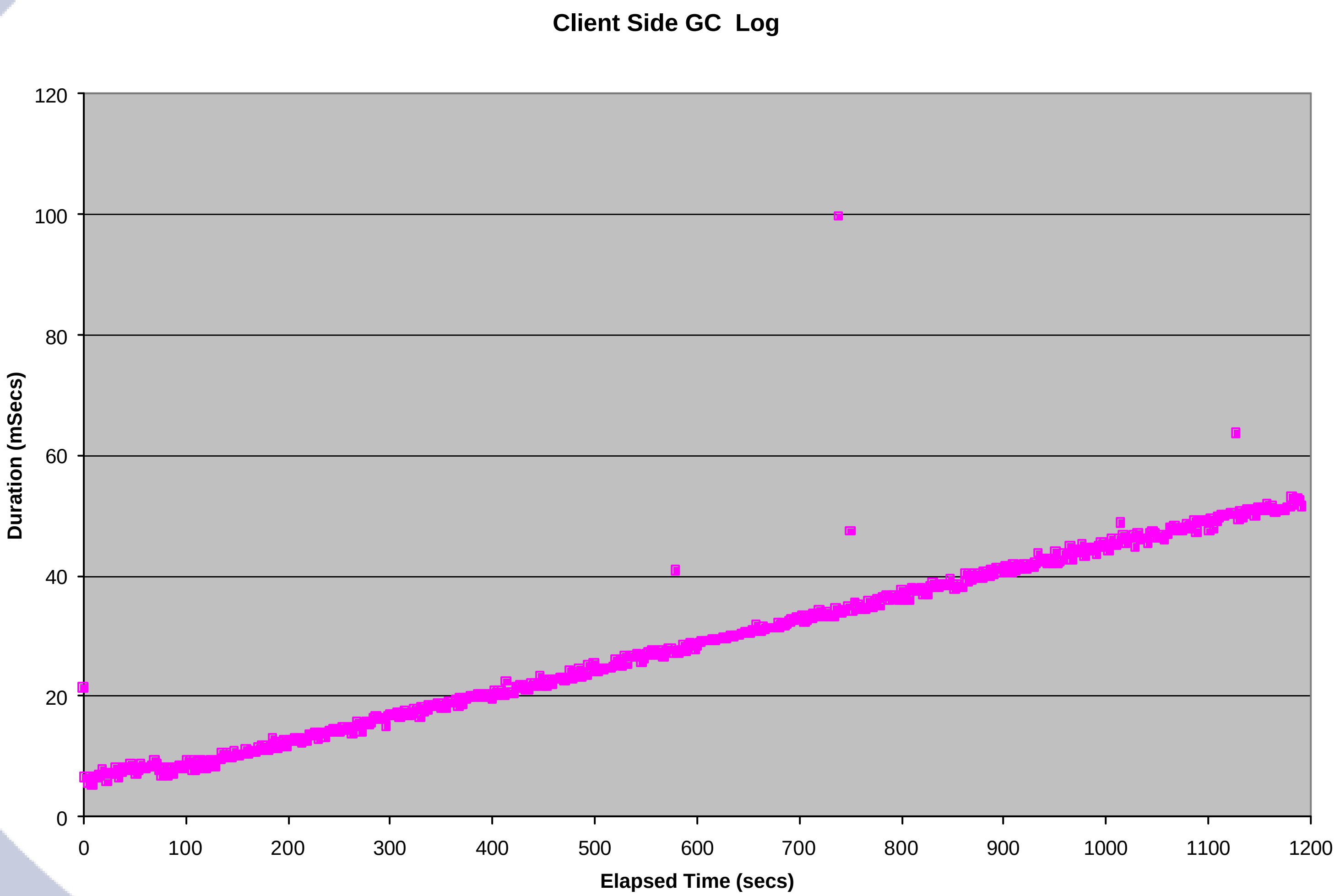
# NHRT Client - Statistics

| *Time* | *Frequency* | *%* |
|---|---|---|
| 1.3 | 0 | 0.00% |
| 1.4 | 169 | 0.42% |
| 1.5 | 32569 | 81.42% |
| 1.6 | 6066 | 15.17% |
| 1.7 | 887 | 2.22% |
| 1.8 | 167 | 0.42% |
| 1.9 | 59 | 0.15% |
| 2 | 49 | 0.12% |
| 2.1 | 31 | 0.08% |
| 2.2 | 2 | 0.01% |
| 2.3 | 1 | 0.00% |
| More | 0 | 0 |
| Total | 40000 | |

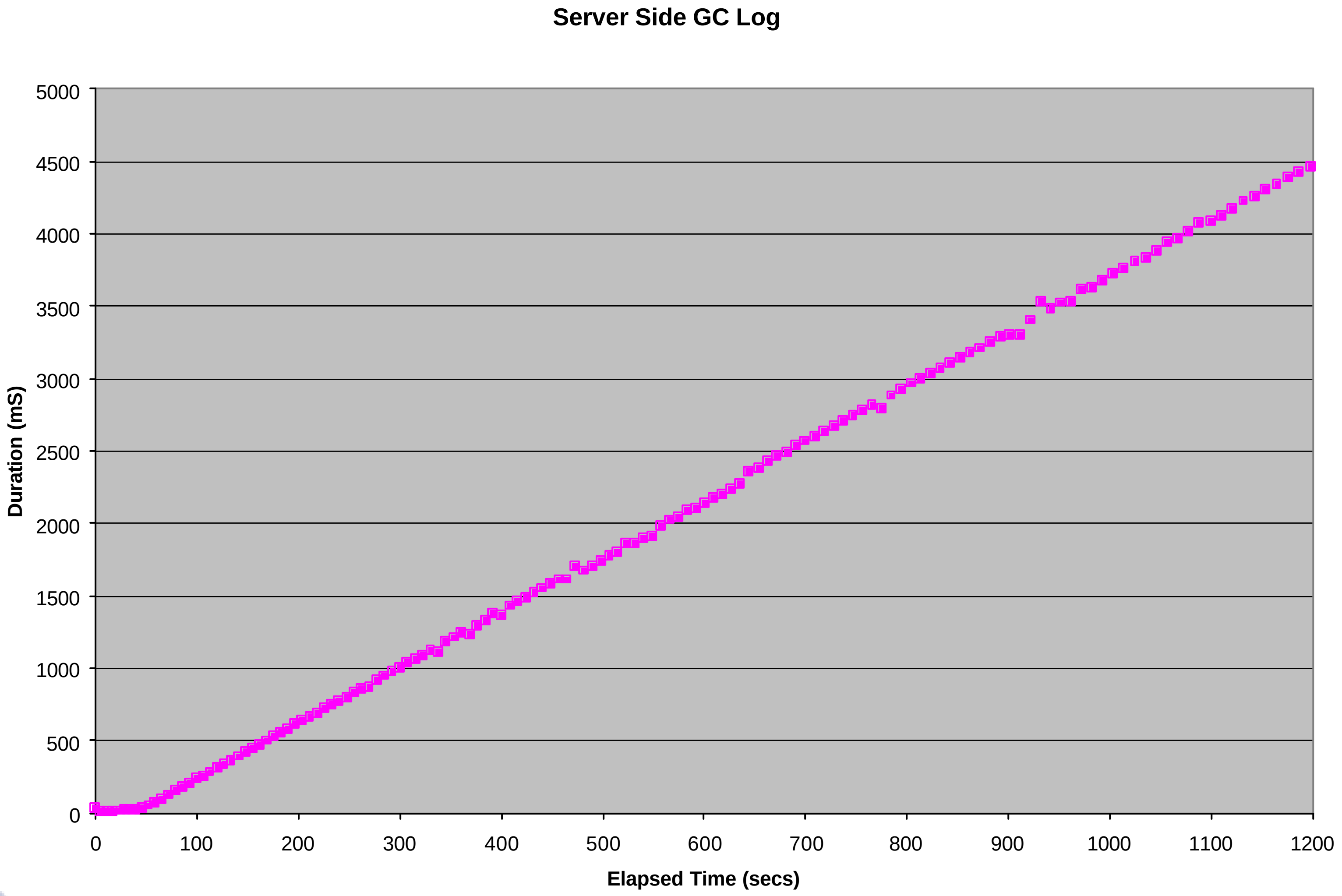
# NHRT Client – Absolute Period



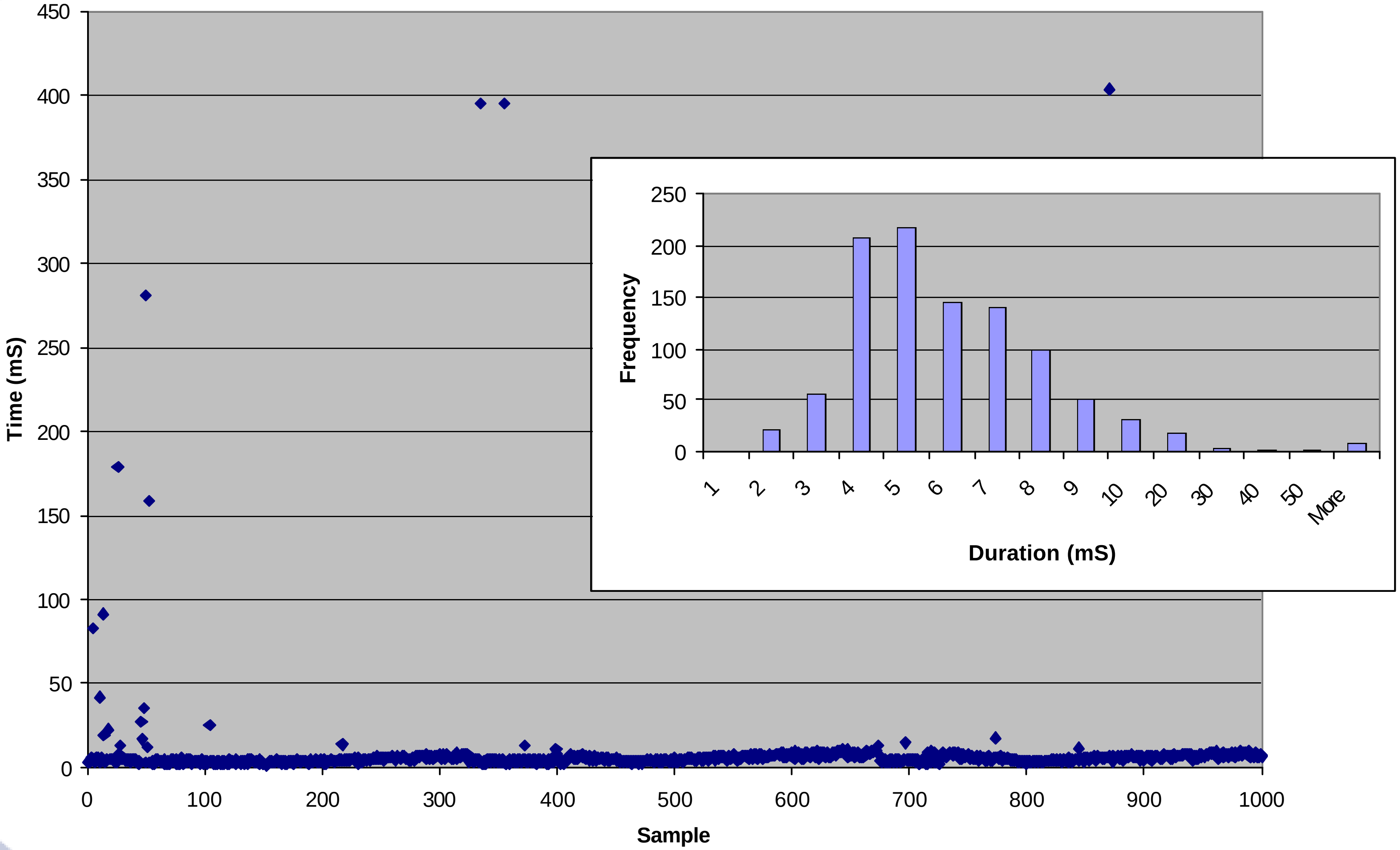
# Client GC Logs



# Server Side GC Log



# Non-Real-Time ORB and JVM



# Summary and Conclusions

- Real-time CORBA in a Java environment is a realisable goal
- The impact of GC can be eliminated by using NHRT threads and immortal memory
- Real-time performance is influenced by a number of factors:
  - ORB architecture
  - Underlying network transport can affect jitter significantly, TCP/IP may not be appropriate
  - JIT compiler must be disabled!
  - Pre-compilation should be considered
  - Operating system must provide preemptive scheduling
- Writing real-time Java/CORBA applications will not be trivial – strict design patterns must be followed
  - Existing application code will require migration

# For more Information…

- Please visit the PrismTech website

**www.prismtech.com**

- Email

**info@prismtech.com**