

**AFRL-IF-RS-TR-2006-117**  
**In-House Final Technical Report**  
**March 2006**



# **JOINT BATTLESPACE INFOSPHERE (JBI)**

## **CONTENT BASED ROUTING**

*APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.*

**AIR FORCE RESEARCH LABORATORY**  
**INFORMATION DIRECTORATE**  
**ROME RESEARCH SITE**  
**ROME, NEW YORK**

## **STINFO FINAL REPORT**

This report has been reviewed by the Air Force Research Laboratory, Information Directorate, Public Affairs Office (IFOIPA) and is releasable to the National Technical Information Service (NTIS). At NTIS it will be releasable to the general public, including foreign nations.

AFRL-IF-RS-TR-2006-117 has been reviewed and is approved for publication

APPROVED:       /s/

HELEN M. RICO  
Chief, Distributed Information Systems Branch  
Information Grid Division

FOR THE DIRECTOR:       /s/

WARREN H. DEBANY, JR., Technical Advisor  
Information Grid Division  
Information Directorate

|                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |                                                                     |                                                                    |                                                                                        |                               |
|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------|--------------------------------------------------------------------|----------------------------------------------------------------------------------------|-------------------------------|
| <b>REPORT DOCUMENTATION PAGE</b>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |                                                                     |                                                                    | Form Approved<br>OMB No. 074-0188                                                      |                               |
| Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing this collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188), Washington, DC 20503                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |                                                                     |                                                                    |                                                                                        |                               |
| <b>1. AGENCY USE ONLY (Leave blank)</b>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |                                                                     | <b>2. REPORT DATE</b><br>MARCH 2006                                | <b>3. REPORT TYPE AND DATES COVERED</b><br>In-House Final, Feb 04 – Dec 05             |                               |
| <b>4. TITLE AND SUBTITLE</b><br>JOINT BATTLESPACE INFOSPHERE (JBI) CONTENT BASED ROUTING                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |                                                                     |                                                                    | <b>5. FUNDING NUMBERS</b><br>C - N/A<br>PE - 62702F<br>PR - 4519<br>TA - CB<br>WU - RP |                               |
| <b>6. AUTHOR(S)</b><br>Mark D. Saeger, Capt., USAF                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |                                                                     |                                                                    |                                                                                        |                               |
| <b>7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)</b><br>Air Force Research Laboratory/IFGA<br>525 Brooks Road<br>Rome New York 13441-4505                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |                                                                     |                                                                    | <b>8. PERFORMING ORGANIZATION REPORT NUMBER</b><br><br>N/A                             |                               |
| <b>9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)</b><br>Air Force Research Laboratory/IFGA<br>525 Brooks Road<br>Rome New York 13441-4505                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |                                                                     |                                                                    | <b>10. SPONSORING / MONITORING AGENCY REPORT NUMBER</b><br><br>AFRL-IF-RS-TR-2006-117  |                               |
| <b>11. SUPPLEMENTARY NOTES</b><br><br>AFRL Project Engineer: Mark D. Saeger, Capt., USAF/IFGA/(315) 330-7059/ Mark.Saeger@rl.af.mil                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |                                                                     |                                                                    |                                                                                        |                               |
| <b>12a. DISTRIBUTION / AVAILABILITY STATEMENT</b><br>APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |                                                                     |                                                                    |                                                                                        | <b>12b. DISTRIBUTION CODE</b> |
| <b>13. ABSTRACT (Maximum 200 Words)</b><br>The purpose of this effort was to integrate a hardware based eXtensible Markup Language (XML) processor into the JBI architecture. Integration was to be invisible to the user and work seamlessly with the current client software. Changes necessary to support plug-in hardware solutions should not require client code changes but rely solely on JBI core framework changes. At low loading levels, the two implementations should behave similarly. The hardware assisted JBI implementation should accelerate JBI operations under heavy loading of publication/subscribe requests with complicated XPATH parsing requirements. It was deemed that XPATH processing and XML parsing were the two areas most likely to suffer under extreme loads and promising areas to leverage commercial technology. A software-based solution must ingest the whole JBI object before beginning processing. Object sizes and number of concurrent objects processed will directly affect system responsiveness. A hardware-based solution can process objects at line speed (e.g. 1Gb/s) as packets flow through the network. Large/complex objects will potentially slow processing down, but in general, as bits flow on the wire the internal workings of the XML router is making decisions and taking action on the data. Lastly, the hardware-based solution is scalable and can provide more processing power as required. |                                                                     |                                                                    |                                                                                        |                               |
| <b>14. SUBJECT TERMS</b><br>Joint Battlespace Infosphere, JBI, Content Based Routing, XML Routing, Publish and Subscribe Architecture                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |                                                                     |                                                                    | <b>15. NUMBER OF PAGES</b><br>76                                                       |                               |
|                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |                                                                     |                                                                    | <b>16. PRICE CODE</b>                                                                  |                               |
| <b>17. SECURITY CLASSIFICATION OF REPORT</b><br><br>UNCLASSIFIED                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         | <b>18. SECURITY CLASSIFICATION OF THIS PAGE</b><br><br>UNCLASSIFIED | <b>19. SECURITY CLASSIFICATION OF ABSTRACT</b><br><br>UNCLASSIFIED | <b>20. LIMITATION OF ABSTRACT</b><br><br>UL                                            |                               |

# Table of Contents

|       |                                             |    |
|-------|---------------------------------------------|----|
| 1     | Summary .....                               | 1  |
| 2     | Introduction.....                           | 1  |
| 3     | Terminology.....                            | 2  |
| 4     | Setup .....                                 | 3  |
| 4.1   | Database.....                               | 3  |
| 4.2   | Java SDK .....                              | 3  |
| 4.3   | Network.....                                | 3  |
| 4.4   | System Hardware Configuration.....          | 4  |
| 4.5   | Configuration File.....                     | 4  |
| 4.5.1 | Sample configuration file.....              | 5  |
| 4.5.2 | server.....                                 | 5  |
| 4.5.3 | port .....                                  | 5  |
| 4.5.4 | adminport .....                             | 6  |
| 4.5.5 | hardwareoverride .....                      | 6  |
| 4.5.6 | mode.....                                   | 6  |
| 5     | Implementation Details.....                 | 6  |
| 5.1   | Software Changes .....                      | 7  |
| 5.1.1 | Base64.java .....                           | 7  |
| 5.1.2 | Hardware.java .....                         | 8  |
| 5.1.3 | Listener.java.....                          | 9  |
| 5.1.4 | SubscriberSequence.java .....               | 9  |
| 5.1.5 | PublisherSequence.java .....                | 10 |
| 5.2   | Message Representation: .....               | 10 |
| 5.2.1 | Subscribe Request.....                      | 11 |
| 5.2.2 | Publish Request.....                        | 11 |
| 5.3   | Schemas .....                               | 12 |
| 6     | Methodology .....                           | 14 |
| 7     | Results and Discussion .....                | 16 |
| 7.1   | Fast/Slow Subscription Case.....            | 17 |
| 7.1.1 | Software JBI.....                           | 19 |
| 7.1.2 | Hardware JBI .....                          | 20 |
| 7.1.3 | Analysis.....                               | 20 |
| 7.2   | Steady Increase in Subscribers Case.....    | 24 |
| 7.2.1 | Software JBI.....                           | 25 |
| 7.2.2 | Hardware JBI .....                          | 26 |
| 7.2.3 | Analysis.....                               | 26 |
| 7.3   | Payload Increase Case.....                  | 27 |
| 7.3.1 | Software JBI.....                           | 29 |
| 7.3.2 | Hardware JBI .....                          | 30 |
| 7.3.3 | Analysis.....                               | 32 |
| 7.4   | Steady Increase/Payload Increase Case ..... | 34 |

|        |                                                |    |
|--------|------------------------------------------------|----|
| 7.4.1  | Software JBI.....                              | 35 |
| 7.4.2  | Hardware JBI .....                             | 38 |
| 7.4.3  | Analysis.....                                  | 39 |
| 7.5    | Heavy Load Case .....                          | 41 |
| 7.5.1  | Software JBI.....                              | 42 |
| 7.5.2  | Hardware JBI .....                             | 42 |
| 7.5.3  | Analysis.....                                  | 43 |
| 7.6    | Saturation Case .....                          | 47 |
| 7.6.1  | Software JBI.....                              | 47 |
| 7.6.2  | Hardware JBI .....                             | 48 |
| 7.6.3  | Analysis.....                                  | 49 |
| 8      | Follow-on Work.....                            | 53 |
| 9      | Conclusion .....                               | 54 |
| 10     | Appendix A – Raw Data .....                    | 57 |
| 10.1   | Experiment 1.....                              | 57 |
| 10.1.1 | Hardware JBI – Object Size 2KB .....           | 57 |
| 10.1.2 | Software JBI – Object Size 2KB .....           | 57 |
| 10.2   | Experiment 2.....                              | 58 |
| 10.2.1 | Hardware JBI – Object Size 2KB .....           | 58 |
| 10.2.2 | Software JBI – Object Size 2KB .....           | 59 |
| 10.3   | Experiment 3.....                              | 60 |
| 10.3.1 | Hardware JBI – 25KB and 50KB Object Size.....  | 60 |
| 10.3.2 | Software JBI – 25KB and 50KB Object Size ..... | 61 |
| 10.4   | Experiment 4.....                              | 62 |
| 10.4.1 | Hardware JBI – Object Size 2KB .....           | 62 |
| 10.4.2 | Software JBI – Object Size 2KB .....           | 63 |
| 10.5   | Experiment 5.....                              | 64 |
| 10.5.1 | Hardware JBI – Object Size 2KB .....           | 64 |
| 10.5.2 | Software JBI – Object Size 2KB .....           | 65 |
| 10.6   | Experiment 6.....                              | 66 |
| 10.6.1 | Hardware JBI – Object Size 2KB .....           | 66 |
| 10.6.2 | Software JBI – Object Size 2KB .....           | 67 |
| 11     | List of Acronyms .....                         | 68 |

## List of Figures

|                                                                                    |    |
|------------------------------------------------------------------------------------|----|
| Figure 1 - Lab Network Configuration .....                                         | 4  |
| Figure 2 - Sample Client XPE Configuration File.....                               | 5  |
| Figure 3 - Base64 Encoding.....                                                    | 8  |
| Figure 4 - Sub/Pub Sequence.....                                                   | 9  |
| Figure 5 - Message Object Representation .....                                     | 10 |
| Figure 6 - Logical XPATH Expression Example .....                                  | 11 |
| Figure 7 - Schema and Metadata Types.....                                          | 12 |
| Figure 8 - jbi.rl.af.mil.ato Object Fields .....                                   | 12 |
| Figure 9 - jbi.rl.af.mil.basic Object Fields.....                                  | 13 |
| Figure 10 - jbi.rl.af.mil.xmlxpath Object Fields .....                             | 13 |
| Figure 11 - Randomization Values for Fields.....                                   | 16 |
| Figure 12 - Randomized Key Words for Fields.....                                   | 16 |
| Figure 13 - Slow/Fast Subscriber Network Setup.....                                | 17 |
| Figure 14 - (a) Slow Subscriber Data, (b) Fast Subscriber Data, (c) Combined ..... | 19 |
| Figure 15 - Performance Difference of "Slow" Subscriber .....                      | 21 |
| Figure 16 - Fast/Slow Subscriber Trend .....                                       | 22 |
| Figure 17 - Delivered Objects/Second.....                                          | 23 |
| Figure 18 - Steady Subscriber Increase Network Setup .....                         | 24 |
| Figure 19 - Steady Subscriber Increase Data.....                                   | 25 |
| Figure 20 - Steady Increase Trend.....                                             | 26 |
| Figure 21 - Delivered Objects/Second.....                                          | 27 |
| Figure 22 - Payload Increase Network Setup .....                                   | 27 |
| Figure 23 - Payload Increase Data, (a) Software, (b) Hardware.....                 | 28 |
| Figure 24 - Payload Increase Trend.....                                            | 29 |
| Figure 25 - Percent Increase over Base Object Size .....                           | 30 |
| Figure 26 - Payload Increase Trend.....                                            | 31 |
| Figure 27 - Percent Increase over Base Object Size .....                           | 32 |
| Figure 28 - Objects Delivered/Second.....                                          | 33 |
| Figure 29 - Steady Increase/Payload Increase Network Setup .....                   | 34 |
| Figure 30 - (a) Software Data, (b) Hardware Data, (c) Combined .....               | 35 |
| Figure 31 - Software Object Size Percentage Impact .....                           | 37 |
| Figure 32 - Hardware Object Size Percentage Impact.....                            | 39 |
| Figure 33 - Objects Delivered/Second.....                                          | 40 |
| Figure 34 - Heavy Load Network Setup .....                                         | 41 |
| Figure 35 - Heavy Load Data.....                                                   | 42 |
| Figure 36 - Heavy Load Trend.....                                                  | 44 |
| Figure 37 - Objects Delivered/Second.....                                          | 45 |
| Figure 38 - Subscriber Count Comparison .....                                      | 46 |
| Figure 39 - Saturation Network Setup .....                                         | 47 |
| Figure 40 - Saturation Data.....                                                   | 48 |
| Figure 41 - Aggregate Hardware/Software Fast/Slow Subscriber.....                  | 49 |
| Figure 42 - Software Fast/Slow Object Processing .....                             | 50 |

|                                                        |    |
|--------------------------------------------------------|----|
| Figure 43 - Hardware Fast/Slow Object Processing ..... | 51 |
| Figure 44 - Objects Delivered/Second.....              | 52 |
| Figure 45 - Saturation Trend.....                      | 53 |

## List of Tables

|                                                                                    |    |
|------------------------------------------------------------------------------------|----|
| Table 1- System Configuration .....                                                | 4  |
| Table 2 - Sync/Async Comparison .....                                              | 6  |
| Table 3 - Impact of "Slow" Subscriber .....                                        | 19 |
| Table 4 - Percent Increase due to Subscriber Load Increase .....                   | 36 |
| Table 5 - Percent Increase due to Payload Size Increase .....                      | 36 |
| Table 6 - Percent Increase due to Subscriber Load Increase .....                   | 38 |
| Table 7 - Percent Increase due to Payload Size Increase .....                      | 38 |
| Table 8 - Ratio of Published Objects to Delivered Objects .....                    | 43 |
| Table 9 - Percent Increase in Object and Processing Speed from Base of 4x250 ..... | 48 |

# 1 Summary

The integration of a hardware based eXtensible Markup Language (XML) processor for accelerating JBI performance was invisible to the user and worked seamlessly with no changes to JBI client software and only minor changes to the JBI provided core framework. Results gathered reflect the simplest change to the JBI framework operation in that the final step of the publish/subscribe operation was done differently using hardware – this was to make comparisons between both solutions as simple and fair as possible.

The experiment outcomes were not surprising. The software JBI is facing a number of hurdles not present with the hardware JBI, most notably that the software JBI is not running on a dedicated, XML aware, network router. At low loading levels, the two implementations behaved similarly as was expected. Only at higher loading levels did the differences in implementation become apparent – most notably the time required to handle the offered subscription load. Payload size appeared to have a greater impact on both solutions than increased subscriber load. The hardware JBI presented the counter-intuitive result of becoming more efficient under greater loading. This result was due to the ability of the hardware to rapidly disseminate a processed publication request at minimal cost, so under higher loading where a single published object will fulfill a larger number of subscriptions, the hardware paid an upfront penalty then could cheaply replicate and send it multiple times. The software JBI had to perform a software duplication of each packet then traverse the TCP/IP stack to distribute the fulfillment to the subscribing nodes.

The data distribution abilities of both systems were captured by the objects delivered per second metric. The hardware had a much wider range of values than the software and generally performed better with increased subscriber loading – leveraging its inherent ability to rapidly route published data to multiple subscribers. In the case of software, the range of behavior was much more restricted and never exceeded the hardware speed for a given experiment. Across all loading levels and payload sizes, hardware ranged from 14.2obj/s to 627.2obj/s and software ranged from 16.4obj/s to 118.4obj/s. The lower bounds were similar, but the upper bounds show an over 5x increase in objects per second processed for hardware.

The hardware JBI system was faster than the software JBI for all experiments performed. The range of improvement was dependent on the configuration of the experiment, but improvements ranged from 340% to 750% faster at the x2000 loading levels. Therefore, the JBI of the future will have to have XML hardware aware routers at its foundation in order to provide the best performance. Tangible performance and scalability benefits available from hardware will far exceed the larger upfront cost of a dedicated hardware router.

# 2 Introduction

The purpose of this effort was to integrate a hardware based eXtensible Markup Language (XML) processor into the JBI architecture. Integration was to be invisible to the user and work

seamlessly with the current client software. Changes necessary to support plug-in hardware solutions should not require any changes to client code but rely totally on changes to the JBI provided core framework.

In simple cases, at low loading levels, the two implementations should behave similarly. The hardware assisted JBI implementation should accelerate JBI operations under heavy loading of publication/subscribe requests with complicated XPATH parsing requirements. Investigating how to offload CPU intensive tasks to dedicated hardware is forward thinking toward a major problem that the JBI will face – how to handle mountains of information in an efficient and timely manner. It was deemed that XPATH processing and XML parsing were the two areas most likely to suffer under extreme loads and promising areas to leverage commercial technology.

A software-based solution must ingest the whole JBI object before beginning processing. Object sizes and number of concurrent objects processed will directly affect how responsive a software-based system will behave. A hardware-based solution on the other hand, can process objects at line speed (1Gb/s, 100Mb/s, 10Mb/s) as the packets flow through the network. Large/complex objects will potentially slow processing down, but in general, as bits flow on the wire the internal workings of the XML router is making decisions and taking action on the data. Lastly, the hardware-based solution is scalable and can be cascaded to provide more processing power as required.

### 3 Terminology

References to *software JBI* refer to the publication/subscription via the traditional JBI software implementation. All JBI actions are handled by the software based JBI platform services and software based common application programmer interface (CAPI). Hardware handles all XPATH processing and publication/subscription processing. Server application and network loads will directly affect the responsiveness of a software-based system

The hardware XML processor is a commercial product developed by Sarvega and has the nomenclature XPE-2000. Sarvega worked closely with AFRL/IF to ensure that the publish/subscribe paradigm implemented internally to their product behaved similarly to the *software JBI*. References to *hardware JBI* refer to publishing/subscribing using a modified JBI CAPI that recognizes when hardware is available and uses hardware to perform XPATH processing and hardware subscription/publication handling. In this case, publication/subscription messages route directly through the XPE-2000 not to the JBI framework. The off-loading of these CPU and time intensive tasks should free the core framework for other important operations. Note that the changes to the CAPI only replaced the critical function calls to publish/subscribe, all other CAPI operations occur as before.

In both the software and hardware case, the core JBI framework internally handles the query and archive operations.

## **4 Setup**

Testbed setup followed the included JBI documents. All systems were configured as JBI clients so had the client install loaded. The JBI server system had the database and JBI framework configured/setup according to the JBI installation documents. Database scripts were executed to setup the database correctly. The Java SDK was installed on all systems to facilitate compiling/installing of updated CAPI files on client systems. Lastly, a network share was setup to distribute updated source files to all target systems.

### **4.1 Database**

Database setup in the lab environment was according to the JBI installation guides. Although the JBI supported both Oracle and MySQL, the experiment was run with only Oracle 9.2.0.1. Database setup scripts were run directly from the included JBI source CD.

### **4.2 Java SDK**

The Java Software Development Kit (SDK) was installed on each system. The version chosen was 1.4.2 and the SDK was used to compile updated CAPI files and client load testing code. Standard Ant build scripts were executed to build the capi-jms library file.

### **4.3 Network**

Network setup was based on client workstation availability/capability and desk space. As such, the network consisted of seven computers as shown in Figure 1, six of which had 1Gbps Ethernet cards, one having only a 100Mbps Ethernet card. The computer (Arnold) with the 100Mbps Ethernet card was also the slowest system – this was intentional so that a “slow” subscriber or “slow” publisher could be introduced into the JBI test environment. The “slow” computer would then throttle both solutions as each solution had to wait on the “slow” system before moving on to the next published object.

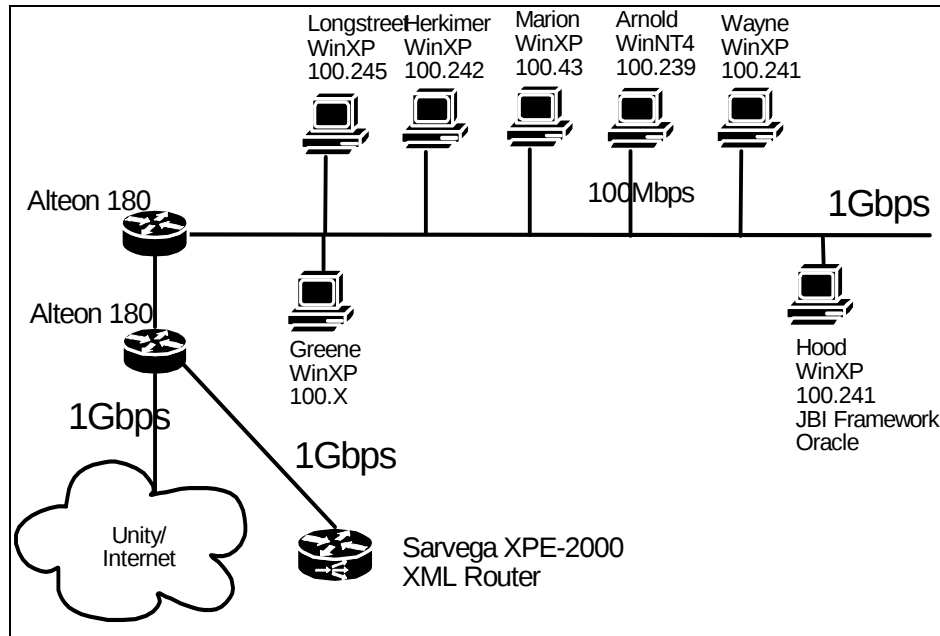


Figure 1 - Lab Network Configuration

#### 4.4 System Hardware Configuration

The hardware configuration of each system is listed in Table 1.

| System     | Processor   | Speed   | Memory | Ethernet | Operating System   |
|------------|-------------|---------|--------|----------|--------------------|
| Longstreet | Pentium IV  | 2.53Ghz | 512MB  | 1GBps    | Windows XP SP1     |
| Herkimer   | Pentium IV  | 2.53Ghz | 512MB  | 1GBps    | Windows XP SP1     |
| Marion     | Pentium IV  | 2.53Ghz | 512MB  | 1GBps    | Windows XP SP1     |
| Arnold (*) | Pentium III | 500Mhz  | 256MB  | 100Mbps  | Windows NT 4.0 SP6 |
| Hood (**)  | Pentium IV  | 2.0 Ghz | 512MB  | 1GBps    | Windows XP SP1     |
| Wayne      | Pentium IV  | 2.0 Ghz | 512MB  | 1GBps    | Windows XP SP1     |
| Greene     | Pentium IV  | 2.0 Ghz | 512MB  | 1GBps    | Windows XP SP1     |

(\*) Arnold was used to introduce a “slow” system in to the experiment  
(\*\*) Hood is the JBI Framework Server and Oracle Database Server

Table 1- System Configuration

#### 4.5 Configuration File

A configuration file is required on all clients that will be using a hardware-based implementation. The configuration file consists of four required parameters and one optional parameter. The required parameters are “*server*”, “*port*”, “*adminport*”, and “*mode*” where *server* is the IP address of the Sarvega hardware, *port* is for sending publication requests, *adminport* is for sending subscription requests, and *mode* determines if an acknowledgement is necessary. The optional parameter *hardwareoverride* if specified to “true” will cause the CAPI to behave as

if hardware is not present. To increase readability of the file, comments and blank space may be included. A hash symbol (“#”) precedes comment lines and blank lines are not processed.

#### 4.5.1 Sample configuration file

In order for the client to locate the Sarvega XPE a configuration file is located on each client. This configuration file identifies the IP address/ports of the XPE hardware and some additional configuration options. Figure 2 presents a sample configuration file for the client.

```
#
#Config file for the sarvega xpe
#
#

#ip address of the server
server=155.244.100.4

#port that is used for sending publication request
port=55555

#port that is used for sending subscription message
adminport=80

#whether to ignore hardware even if hardware is present
hardwareoverride=false

#set the sync mode for transactions (publish)
mode = async
```

**Figure 2 - Sample Client XPE Configuration File**

#### 4.5.2 server

Set *server* to point to the Sarvega XPE’s IP address. At this time, there is no way to discover the Sarvega box remotely (e.g. like DHCP or a broadcast).

#### 4.5.3 port

The *port* field is used by the publication request to send published objects to the Sarvega XPE. The listener on this port ingests the published object and compares it to the subscription entries. If a match occurs, the object is duplicated and distributed. If no match is found, the Sarvega XPE returns a 404 to the publisher and discards the object.

#### 4.5.4 adminport

The *adminport* field is used by the subscription operation to register a subscription request with the Sarvega XPE. The listener on this port ingests the subscription request and sets up the internal state machine of the XPE based on the subscribers XPATH. As objects are published, the state machine determines when a match is present and then the subscriber receives the published object.

#### 4.5.5 hardwareoverride

A simple Boolean *hardwareoverride* flag determines if the client CAPI should use the XPE hardware or traditional software JBI methods.

true: Use the XPE hardware for publication/subscription requests

false: Use the software JBI for publication/subscription requests

#### 4.5.6 mode

The *mode* flag determines how hardware handles publication events. Settings for this flag are “sync” or “async”. Table 2 illustrates the async/sync hardware setting and the software sync setting. By comparing the three settings, it is possible to quantify the impact of the async/sync hardware setting.

| Method   | Objects Received | Approximate Time (ms) | Approximate Loss |
|----------|------------------|-----------------------|------------------|
| HW/Async | 34477            | 125000                | 0.56%            |
| HW/Sync  | 34672            | 188000                | 0.00%            |
| SW/Sync  | 34675            | 442000                | 0.00%            |

Table 2 - Sync/Async Comparison

**sync** behaves similar to TCP/IP in that you have guaranteed delivery. The XPE will not move past the current publishing object until the XPE receives an acknowledgement from the client. One caveat is if the acknowledgement does not arrive within approximately 20s, the router drops the object.

**async** behaves similar to UDP in that it is a best effort delivery system. In this case, the XPE sends objects to clients and does not attempt to determine if client received the object. The benefit of using the ASYNC setting is an approximate 35% speed improvement at the cost of about 0.5% dropped objects. The client still acknowledges delivery to the server for each object, but the server does not actually track the acknowledgements to ensure that one has occurred.

## 5 Implementation Details

The core JBI source code was modified to seamlessly incorporate a hardware-based solution. Source code changes were restricted to the Common API (CAPI) and did not modify the original software capability, but instead augmented the original code with alternate execution paths that recognize the hardware XML router. Execution paths are defined by successfully determining if

a hardware router is present, and if so, the alternate hardware execution paths are followed through the CAPI. If a hardware device is not found or if the choice to use present hardware is overridden, then the original CAPI execution paths are followed.

Note that in order to maintain as equal of comparison as possible, the hardware JBI executes almost the whole software JBI code sequence for both publish and subscribe. This essentially meant that wasted JMS messaging and RMI traffic was generated by the Hardware JBI that was superfluous and wasted bandwidth and time (approximately 200 packets per publish/subscriber operation). This methodology was chosen so that only the very last step in the publish/subscribe operation would use hardware while all other steps would be executed as in the software JBI. The removal of these extraneous operations would have led to an increase in hardware processing speed, but would have made the comparison uneven. Time values gathered only account for how quickly the actual publish/subscribe operation executes and do not take into account the overhead of each particular implementation (software or hardware).

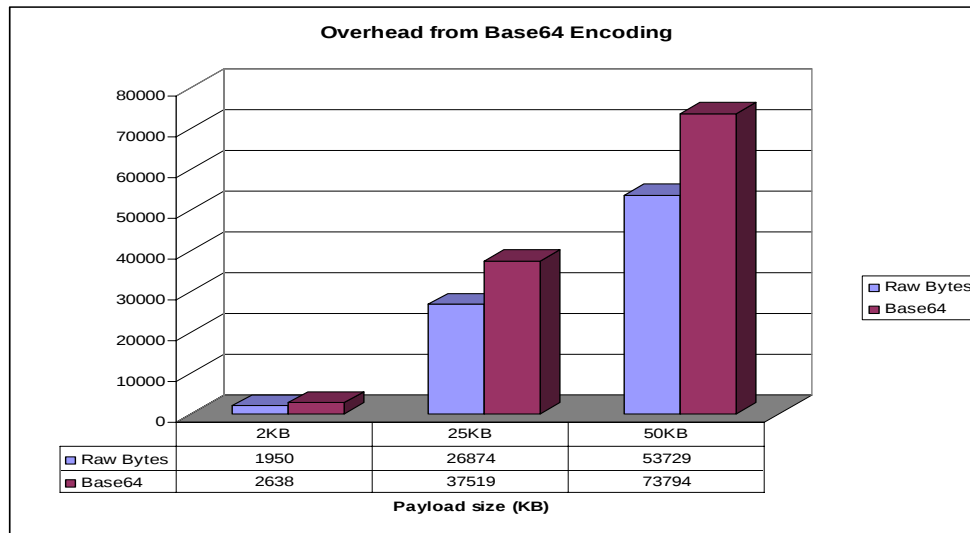
The following is a brief summary of source code changes; all files are located in \CommonAPI\J2EE\jms\src\main\mil\af\rl\jbi\commonAPI\substrate directory.

## **5.1 Software Changes**

Minimal changes were required to the JBI CAPI to incorporate the Sarvega XPE router. A brief description of the new source files and modified files is below:

### **5.1.1 Base64.java**

This source file is a freeware Base64 implementation used to encode the payload of binary publication objects. Base64 encoding adds roughly 33% to the size of the object but provides a text-only encoding of binary. The software JBI passes binary Java *Objects* via JMS. The hardware JBI does not utilize JMS and incorporates the binary objects into an XML wrapper. In order to utilize an XML wrapper, the Java object is Base64 encoded to a text based representation of the binary data. Upon receipt, the object is Base64 decoded and converted back into the generic *Object* type.



**Figure 3 - Base64 Encoding**

The experiment utilized 2KB, 25KB and 50KB payload sizes that were sent via JMS for the software JBI and encoded via Base64 for the hardware JBI. As can be seen from Figure 3, the XPE version of the JBI suffered an additional overhead cost of approximately 33% because all payloads were Base64 encoded.

### 5.1.2 Hardware.java

The main class file contains support for the hardware JBI implementation. Both `SubscriberSequence.java` and `PublisherSequence.java` incorporate the Hardware base object. If the hardware JBI is running, then the execution paths for both publishing and subscribing use hardware. The basic implementation for subscribing and publishing is as follows:

A subscription event creates a subscription request that registers itself as listening for a particular object. This causes two actions: the first action is the creation of a single listener on the client PC that is shared between all subsequent subscription requests and registering the particular callback with the listener so that objects are returned to the listener associated with the subscription. The second action is a message informing the XPE what the XPATH is for this particular subscription and associating the returned client ID from the XPE to the listening callback. By defining a unique subscription ID, the same listener can handle multiple subscription requests lessening computer resources and allowing site-specific firewall port configuration.

A publication event generates a publish request that is sent to the XPE containing the object to be published (metadata and payload). The metadata is compared to the subscription XPATH information internally on the XPE, and as matches are made, the object is copied and sent with the matching listener associated with a client ID. The client PC receives the published object,

checks for a matching client ID and listener callback, and if found, returns the object to the subscribing function. Figure 4 shows a functional representation of the complete subscription/publication process.

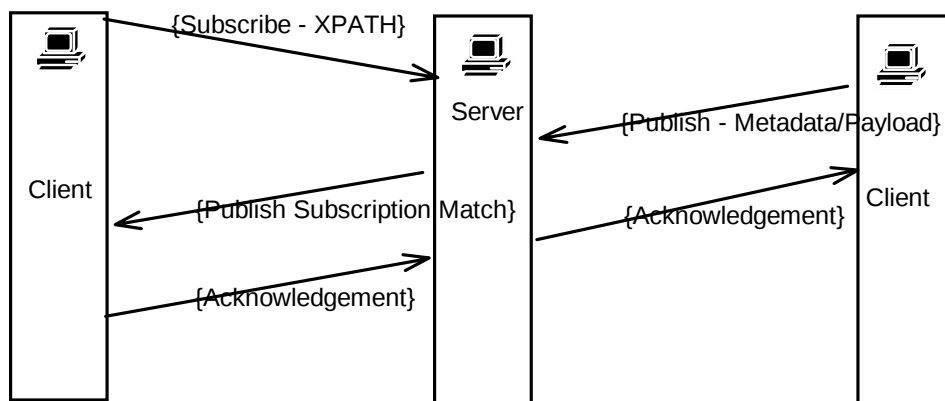


Figure 4 - Sub/Pub Sequence

### 5.1.3 Listener.java

In order to support connections expeditiously, a separate listener class was created and a single listener is implemented. After the listener is created on a specific port, all subsequent subscription requests are inserted into an array (for speed) containing the associated client ID from the XPE, the object token assigned by the JBI framework, and the callback. To facilitate cleaning up after a crashed or exited client, the XPE “pings” the listening port by opening and immediately closing a connection after approximately 30 seconds of inactivity. If the XPE realizes the port has closed, then the ports subscription entries are cleared. An “unsubscribe” is still the cleanest way to sever the connection, but the hardware JBI can recover gracefully without a formal unsubscribe.

A single shared listener is implemented using a Singleton class. A Singleton class employs synchronization objects to ensure that one and only one object is instantiated. Once a single object is made, all subsequent calls receive the pointer to the object.

The shared listener is implemented as a multithreaded listener. As each connection is accepted, a worker thread is created to handle the connection. The full object is ingested and parsed to separate metadata and payload. The Base64 encoded payload is converted back to a serializable Java *Object* and the correct callback is found and executed for the listening client ID.

### 5.1.4 SubscriberSequence.java

The SubscriberSequence was modified in the constructor to create the hardware object. The following functions were changed to add support for hardware via an initial check to see if the hardware object is present.

#### 5.1.4.1 setSequenceCallback

setSequenceCallback starts the listener and associates the callback message.

#### 5.1.4.2 activateSequence

ActivateSequence sends the subscribe request to the hardware and sets up callback information.

#### 5.1.5 PublisherSequence.java

The PublisherSequence was modified to support the hardware object. The following function was changed to add support for hardware by attempting to create the hardware object

#### 5.1.5.1 publishInfoObject

publishInfoObject publishes the object by sending the object to the hardware.

### 5.2 Message Representation:

A single root tag must wrap around the published object in order for the hardware JBI to delineate the scope and boundaries of the object. Referencing Figure 5, this leads to the following format for objects that are routing through the hardware JBI:

```
<root>
  <metadata>
    metadata
  </metadata>
  <payload>
    Base64 encoded payload
  </payload>
</root>
```

Figure 5 - Message Object Representation

The <root> and </root> tag are quietly ignored on the hardware and are only present to bracket the <metadata> and <payload>. The hardware JBI makes routing decisions based upon tags contained in the metadata section. The hardware JBI can be tuned to ignore the payload, compress the payload, sign the payload, etc., but presently the payload is just ingested and stored.

### 5.2.1 Subscribe Request

The subscription phase makes an HTTP connection to the subscribe daemon on the XPE with the actual subscription contained in the POST portion of the HTTP message.

URL:

<http://address:adminPort/cgi-bin/subscribe.cgi>

POST data:

xpath=[url encoded]predicate  
topic=object type  
port=listening port  
version=object type version  
ip=client IP address

If the XPE accepted the subscription request with no errors, it returns an HTTP response code of 200 and a unique client ID.

The POST method allows for essentially an unlimited amount of data to be POSTed. The GET method is also available, but the HTTP specification limits data available for subscriptions to 4096 bytes.

Figure 6 shows an example XPATH expression with embedded Boolean operations in the XPATH. Operations on numeric values did not require single ticks, whereas operations on string values required single ticks to process correctly.

```
(  
(<metadata><BasicTemporal><beginning_date_time_group><day>>=26) and  
(<metadata><BasicTemporal><beginning_date_time_group><hour_time>>=13) and  
(<metadata><BasicTemporal><beginning_date_time_group><minute_time>>=28) and  
(<metadata><BasicTemporal><beginning_date_time_group><month_name>='September') and  
(<metadata><BasicTemporal><beginning_date_time_group><year>>=2003) and  
(<metadata><BasicTemporal><ending_date_time_group><day><=26) and  
(<metadata><BasicTemporal><ending_date_time_group><hour_time><=4) and  
(<metadata><BasicTemporal><ending_date_time_group><minute_time><=0) and  
(<metadata><BasicTemporal><ending_date_time_group><year><=2003)  
)
```

Figure 6 - Logical XPATH Expression Example

### 5.2.2 Publish Request

The publication phase creates an HTTP connection to the publication daemon on the XPE with the payload contained in the POST portion of the HTTP message.

URL:

<http://this.address:port/index.html>

Content-type:

application/xml

POST data:

The metadata and payload of the object are sent. If there is no corresponding subscription for the publication request, the XPE returns a HTTP response code of 404 and drops the object.

### 5.3 Schemas

Figure 7 shows schemas and metadata installed on the JBI platform for this experiment. Note that the payload on each of object is textual, although the actual payload was handled as a generic “object” class so could have been any valid Java *Object* data type.

| Schema                              | Metadata                            |
|-------------------------------------|-------------------------------------|
| mil.af.rl.jbi.training.ato.xsd      | mil.af.rl.jbi.training.ato.xml      |
| mil.af.rl.jbi.training.basic.xsd    | mil.af.rl.jbi.training.basic.xml    |
| mil.af.rl.jbi.training.xmlxpath.xsd | mil.af.rl.jbi.training.xmlxpath.xml |

Figure 7 - Schema and Metadata Types

Figure 8-Figure 10 show a pictorial representation of the schemas used for this experiment. Circled fields are randomized for publish/subscribe operations. The methodology shown in Section 6 contains further information on how schema fields were randomized and assigned experimental values.

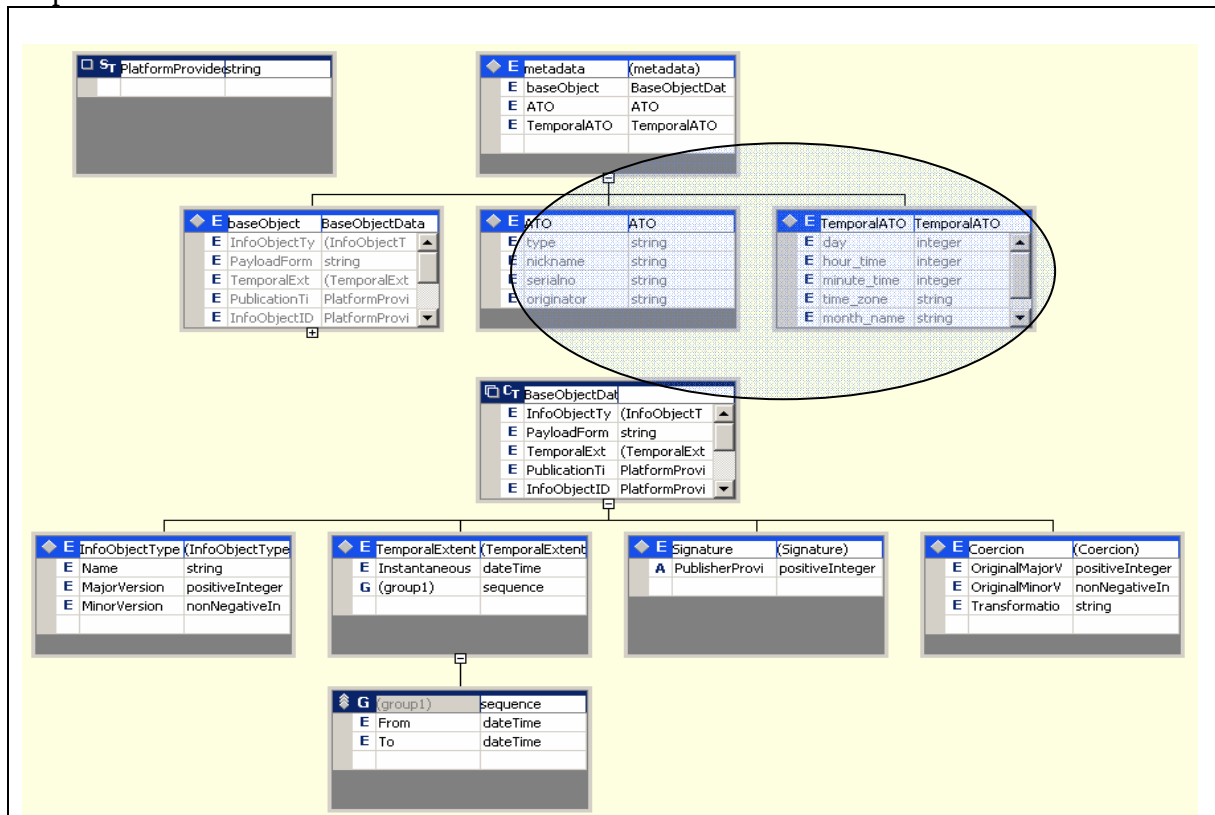
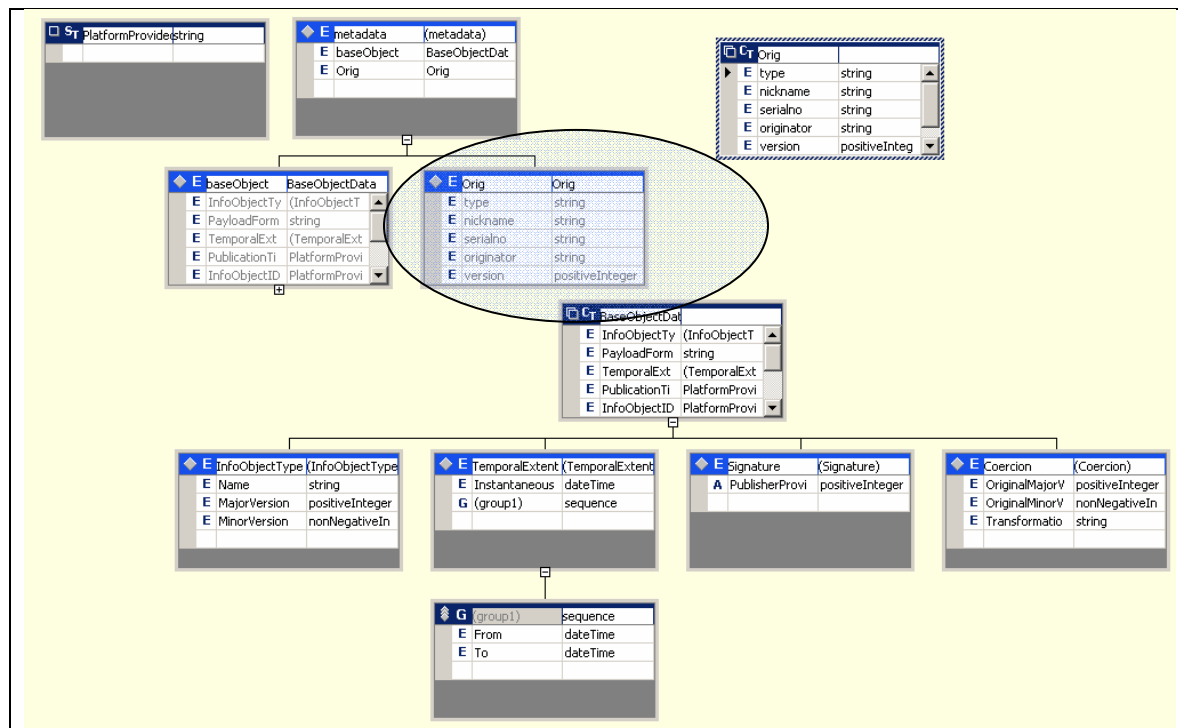
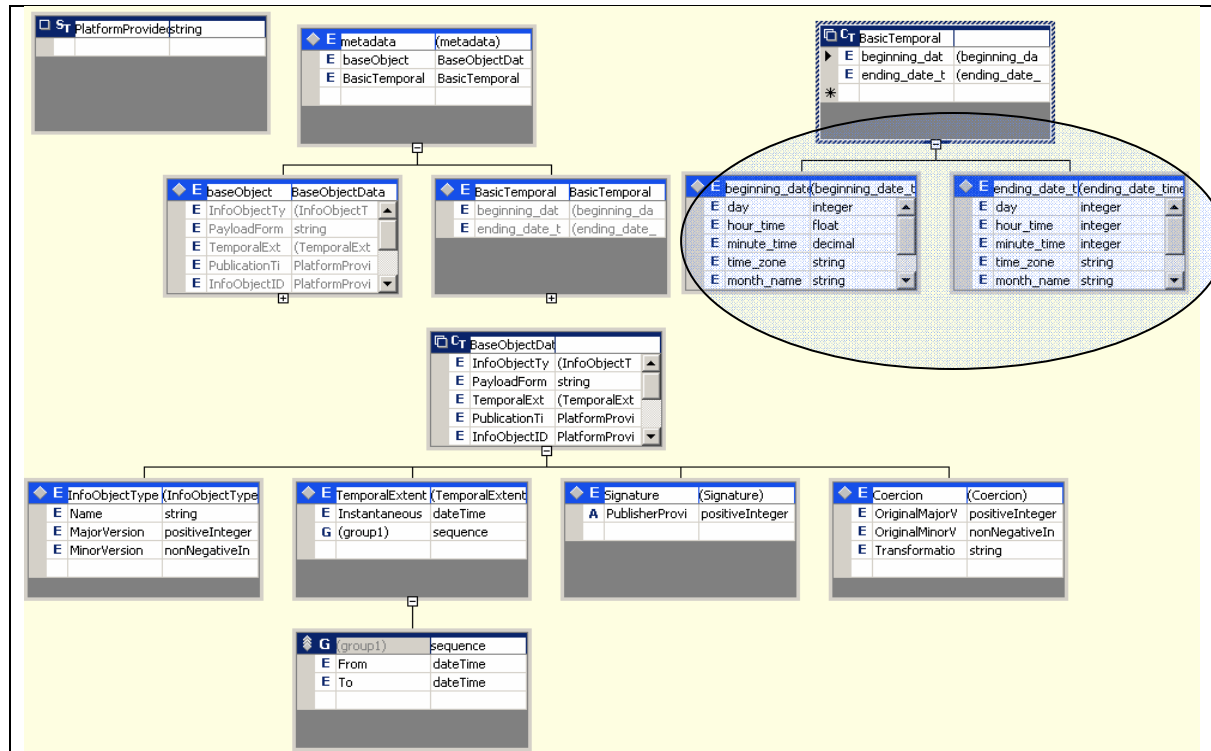


Figure 8 - jbi.rl.af.mil.ato Object Fields



## 6 Methodology

In order to isolate the speed of each solution, a linear programming model was employed on the publication program. This linear approach required the publisher to finish publishing an object and receive an acknowledgement before publishing the next object. Once the publisher had published all required objects it had completed its task. Timestamps were gathered from subscribers to determine how long it took each subscriber to consume all published objects. Speed comparisons can be made between software and hardware JBI instantiations to quantify the differences in implementation and execution speed.

To minimize the client output status messages impact, the subscription output windows were minimized to the system tray. Windows would still output the status messages showing the timestamp and total receipt counter, but did not have to draw that information to the display, which would have consumed resources. The publisher program(s) were then started and used to monitor the state of the experiment – once the publisher completed execution then the client program windows were maximized and timing data recorded for the run.

The software JBI experiments used unmodified configuration defaults. The hardware JBI had a large number of configurable parameters, but I chose to use the default parameters as configured by Sarvega. The one exception on the hardware JBI implementation dealt with the *maximum\_connections* value – this value played a role in asynchronous runs as it limits the maximum number of concurrent connections. In asynchronous mode, it was possible that the router would make too many concurrent connections to a given client PC and exceed the maximum listening connection backlog on the client PC (a Windows sockets limitation). *maximum\_connections* played no role in the synchronous testing that was performed for this experiment.

There are four loading levels to each experiment, consisting of a base subscription loading level of 250, 500, 1000, and 2000 subscribers. Depending on experimental loading, the base loading value was multiplied by a scalar value to create the required subscription/publication load. An iteration of an experiment consisted of executing a particular loading level three times and averaging the results.

The order of iteration execution changed on subsequent runs to mix up the tree building on the server; for example, the first iteration would be subscriber 1, subscriber 2, then subscriber 3 – the second iteration would be subscriber 2, subscriber 3, then subscriber 1. All commands were entered then started consecutively as quickly as the <enter> key could be pressed on the client machines. Once all commands were <enter>ed, the output windows provided status information indicating subscription loading progress. The results ignore the subscription times in the timing calculations as the intent of this JBI experiment was to document how quickly information dissemination occurs, and subscription times only setup the system to perform the dissemination.

Once the subscribed entries had “stabilized” (applicable to hardware), then each publisher was opened on the requisite computers and the publish command was entered. Once all commands

were keyed in, then they were also started in the same fashion as the subscription entries (e.g. as fast as the <enter> key could be depressed). Publish windows were not minimized so I could determine when the publish action completed. Once all publishes operations were complete – the subscriber windows were checked for the final object count/timestamp.

The subscriber consumed objects by reading in both the metadata and payload. After consuming the object, the callback simply incremented the object count variable and displayed the timestamp. Timestamp initialization occurred upon the first receipt of an object and was not reset until the subscriber process exited.

To further complicate the XPATH generation for subscribers and generate unique published output, certain fields had random parameters generated. The random values assigned to each run came from a seed value to allow for replication of behavior. These seed values were incorporated into the publish/subscribe applications to build the expected metadata. Figure 11 below, shows that for each object there is a corresponding XPATH generated with a random data type based on a probability. A uniform distribution determined the probability of a field being present in an XPATH expression or included in the published object. Figure 12 shows the random values generated as the data items for a given XPATH expression. A random uniform distribution determined the values assigned to a given XPATH expression. The basic structure of the XPATH and published objects was similar, but the actual data contained therein was randomly distributed over a wide range of possible values increasing the amount of work necessary for the server to determine if a match is present. Seed values for publishers and subscribers were always different so all matches came from the Boolean operations in the XPATH being satisfied.

| Object | XPath                                                         | Op | Type    | Probability |
|--------|---------------------------------------------------------------|----|---------|-------------|
| ato    | /metadata/ATO/nickname                                        | =  | String  | 80%         |
|        | /metadata/ATO/serialno                                        | =  | String  | 80%         |
|        | /metadata/ATO/originator                                      | =  | String  | 40%         |
|        | /metadata/TemporalATO/day                                     | >= | Integer | 30%         |
|        | /metadata/TemporalATO/hour_time                               | >= | Integer | 80%         |
|        | /metadata/TemporalATO/minute_time                             | >= | Integer | 60%         |
|        | /metadata/TemporalATO/month_name                              | =  | String  | 70%         |
|        | /metadata/TemporalATO/year                                    | >= | Integer | 80%         |
|        | /metadata/BasicTemporal/beginning_date_time_group/day         | >= | Integer | 30%         |
| basic  | /metadata/BasicTemporal/beginning_date_time_group/hour_time   | >= | Integer | 80%         |
|        | /metadata/BasicTemporal/beginning_date_time_group/minute_time | >= | Integer | 60%         |
|        | /metadata/BasicTemporal/beginning_date_time_group/month_name  | =  | String  | 70%         |
|        | /metadata/BasicTemporal/beginning_date_time_group/year        | >= | Integer | 80%         |
|        | /metadata/BasicTemporal/ending_date_time_group/day            | <= | Integer | 40%         |
|        | /metadata/BasicTemporal/ending_date_time_group/hour_time      | <= | Integer | 80%         |
|        | /metadata/BasicTemporal/ending_date_time_group/minute_time    | <= | Integer | 60%         |
|        | /metadata/BasicTemporal/ending_date_time_group/month_name     | =  | String  | 70%         |

|          |                                                     |    |         |     |
|----------|-----------------------------------------------------|----|---------|-----|
|          | /metadata/BasicTemporal/ending_date_time_group/year | <= | Integer | 80% |
| xmlxpath | /metadata/Orig/type                                 | =  | String  | 80% |
|          | /metadata/Orig/nickname                             | =  | String  | 80% |
|          | /metadata/Orig/serialno                             | =  | String  | 70% |
|          | /metadata/Orig/originator                           | =  | String  | 80% |
|          | /metadata/Orig/version                              | >= | Integer | 40% |

Figure 11 - Randomization Values for Fields

| XPath object  | Number  | Values                                                                                           |
|---------------|---------|--------------------------------------------------------------------------------------------------|
| month_name    | 12      | January, February, March, April, May, June, July, August, September, October, November, December |
| nickname      | 10      | KTJR, KMYX, KQAZ, KWXS, KDEC, KFRV, KTBG, KNHY, KUJM, KIKL                                       |
| originator    | 10      | AFRL, AFMC, ACC, AFSOC, AFCA, AFC2ISR, AFSPC, AFWIC, NSA, CIA                                    |
| type          | 5       | Targets, Protected, Critical, Friendly, Moving                                                   |
| nick          | 5       | targ, prot, crit, frnd, move                                                                     |
| alphabet      | 26      | A, B, C, ... , Y, Z                                                                              |
| ATO/serialno  | 50,000  | nick(1)+random(9999)                                                                             |
| Orig/serialno | 260,000 | "JBI"+random(9999)+alphabet(1)                                                                   |

Figure 12 - Randomized Key Words for Fields

## 7 Results and Discussion

The experimental data is presented in six groupings. In all cases, the experiment was executed three times and the data recorded for each loading level (250, 500, 1000, and 2000). The data was averaged across all three experiments to determine the final data value. The focus of the experiment was on publishing speed – or how quickly the JBI hardware/software server could distribute the information to the subscribers. I performed two hundred and four total experiments and recorded data in Excel for ease of calculation/analysis. The subscription setup speed was not tracked, although the software subscription speed was much faster than the hardware subscription speed – which is directly related to the fact that the hardware is a rough prototype and minimal changes were made to a commercial product to support our experimentation.

The data are presented under each experimental type, further subdivided by software, hardware, and an analysis section. The format of the presented data is a graphical representation of the tabulated data elements. Where applicable, the title of the chart defines how many fast and slow publishers and subscribers are present, for example: “**Operation [Fast/Slow] : Pub [4/0], Sub [0/1]**” – means that there were four fast publishers, no slow publishers and zero fast subscribers, with one slow subscriber.

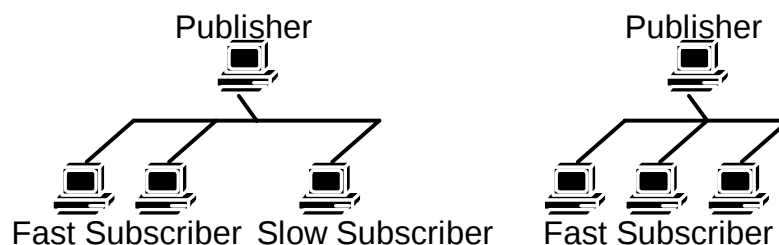
In all cases, two common data elements illustrate the loading level of the experiments. The first, “Objects Published,” indicates the number of objects created by the publishing application and

delivered into the JBI system. The second, “Objects Delivered,” indicates the amount of additional work that had to occur inside the JBI system to deliver the published objects to the requesting subscriber. This value shows the amount of replication that occurred – for example, if a publisher published 2000 objects causing 69237 object deliveries, then on average each published object was replicated 35 times and fulfilled multiple subscription entries. The level of effort on the JBI system correlates to how many replications occur, how many unique communication end-points are present, and how many fulfilled subscription entries are satisfied (based on a complete XPATH evaluation of all loaded subscribers). Therefore, especially at the higher loading levels (x1000 and x2000), the number of replications has a pronounced affect on the speed of the JBI system, but also gives a more accurate indicator of how a fielded JBI system would behave. Replications and complex XPATH expression evaluation will be an operational constant and efficient handling of these operations will increase the JBI system throughput substantially.

Trend lines illustrate the current behavior of the system and the expected behavior of the system beyond the data points gathered. The regression trend lines chosen had the highest  $R^2$  values and most accurately followed the data that was gathered. In many instances, differing trend line types are present on the same graph – this was in an effort to match the data most accurately and oftentimes the differences in data did not facilitate choosing the same type of trend line. *The trend line extensions beyond the data gathered are not necessarily accurate and would require further testing to validate, but do give a quick visual “what-if” look at possible “future” data points/behavior.*

## 7.1 Fast/Slow Subscription Case

The fast and slow subscription case quantifies the behavior of both the hardware and software JBI when a possible slow subscriber is present. Realistically, not all subscribers to a given object will process the object as quickly and this experiment compares the performance difference between the two scenarios. The expected result of a “slow” subscriber should be a decrease in aggregate publishing fulfillment speed, as the server must consume additional time to fulfill the slower subscription endpoint delaying follow-on subscription fulfillments.



**Figure 13 – Slow/Fast Subscriber Network Setup**

Figure 13 above shows the basic configuration of the system for the slow subscriber/fast subscriber experiment. All other client systems stayed the same with the exception of the one system swapped out to provide the necessary fast/slow operation. Figure 14 below is divided into two sections for comparison of the fast/slow (a) and fast/fast (b) cases for both hardware and

software, the (c) chart shows the combined data points. Figure 14 (c) is a combination of an overlay of (a) and (b) for easy comparison between hardware and software JBIs.

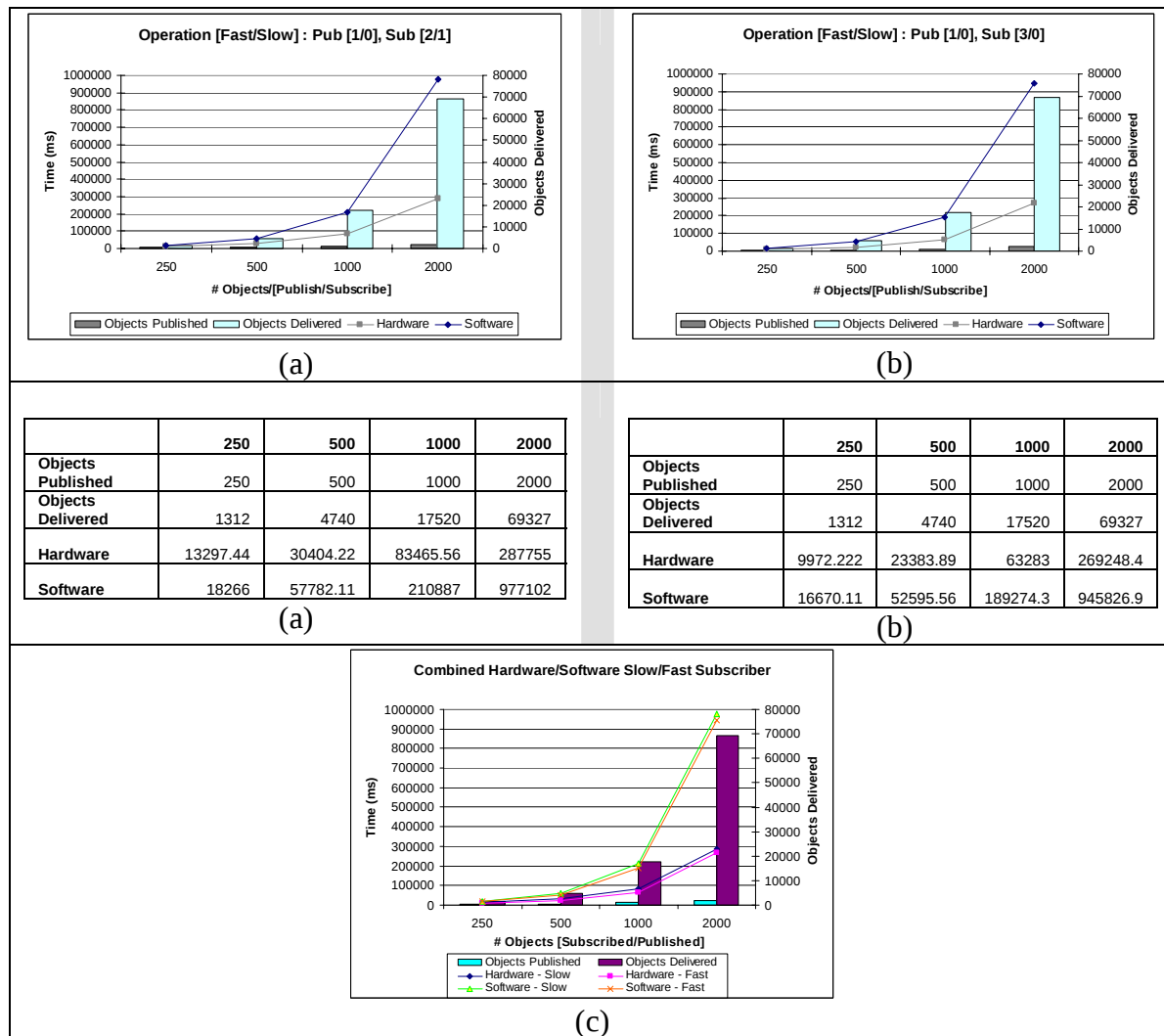


Figure 14 - (a) Slow Subscriber Data, (b) Fast Subscriber Data, (c) Combined

| Impact of "Slow" subscriber | 250   | 500   | 1000  | 2000 |
|-----------------------------|-------|-------|-------|------|
| Hardware                    | 33.3% | 30.0% | 31.9% | 6.9% |
| Software                    | 9.6%  | 9.9%  | 11.4% | 3.3% |

Table 3 - Impact of "Slow" Subscriber

### 7.1.1 Software JBI

The software JBI results differed slightly when a "slow" subscriber was present as compared to all fast subscribers. Both Figure 14 (a) and (b) above, show similar response times between the two cases. Figure 14 (c) illustrates that in general, the software JBI performed much slower

than the hardware JBI as the load increased, and at a loading level of 2000 was approximately 3x slower.

The impact as shown in Table 3 of the slow subscriber is approximately 10% for the 250, 500, and 1000 loading level. The impact decreases to only 3% for the 2000 loading level – this smaller impact is due to the larger server load offsetting the fact that there is a slow subscriber present. In general, the slow subscriber causes the software JBI to perform approximately 10% worse, which is a relatively minor impact. Examining Figure 14 (c) further supports this conclusion as the two software curves are close together.

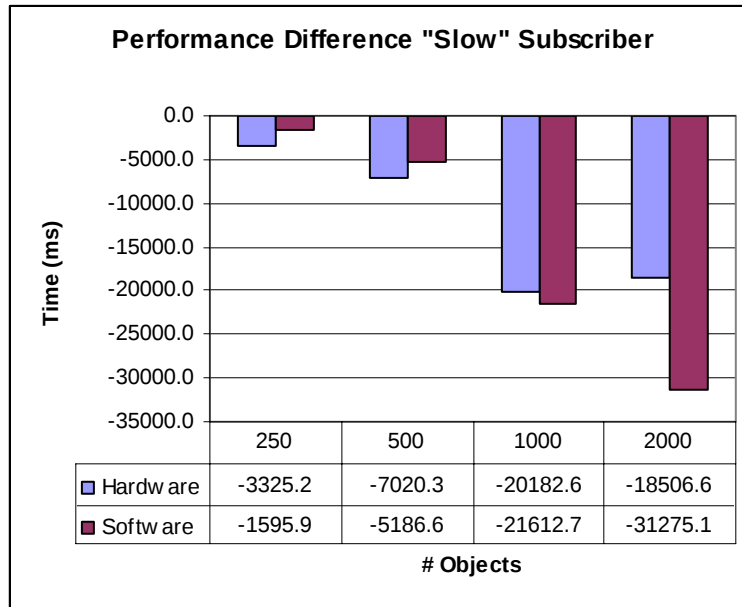
### **7.1.2 Hardware JBI**

As was the case for the software JBI, the hardware JBI exhibited similar behavior when a “slow” subscriber was present to all fast subscribers. Figure 14 (a) and (b) above show a similar response between both cases. Figure 14 (c) illustrates that the hardware JBI outperforms the software JBI by about a factor of 3x at the 2000 loading level.

The impact chart shown in Table 3 actually highlights that the hardware is more susceptible to a slow subscriber. There is slightly over 30% impact for the 250, 500, and 1000 loading levels when a slow subscriber is present. As was shown in the software JBI case, the 7% impact at the 2000 level is smaller due to the increased load on the system offsetting the slower subscriber speed. In general, the slow subscriber impacts the hardware JBI at a cost of approximately 30% and is most noticeable at lower loading levels – although the hardware JBI still outperforms the software JBI considerably regardless of the greater impact of a slow subscriber.

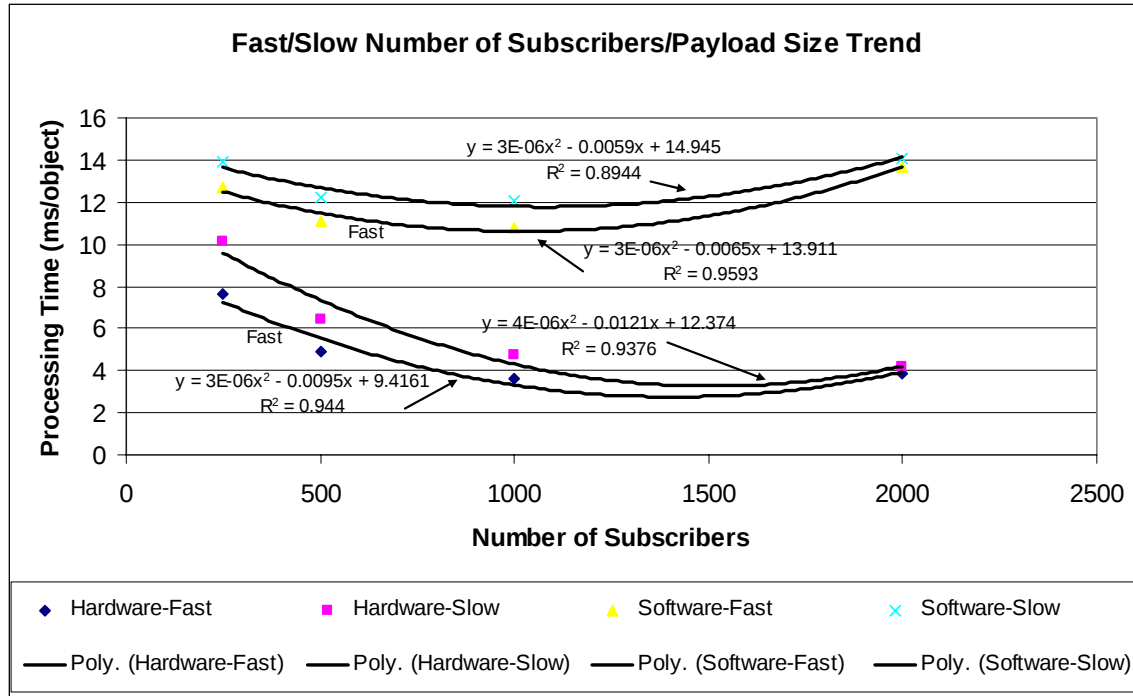
### **7.1.3 Analysis**

The results for this scenario were surprising in both the software and hardware case. As shown in Figure 15, the impact of a slow subscriber was relatively minor, adding 18.5s (6.8%) to the 2000 subscriber run for hardware and 31.2s (3.3%) to the 2000 subscriber run for software. Interestingly, the software JBI more efficiently handles slow subscribers as shown by the 250 and 500 run, where the impact of the slow subscriber is less than the impact for the hardware JBI. The software JBI performs slightly slower than the hardware JBI on the 1000 run. The 2000 run shows that the software JBI took just over 31s to handle the data, but this was on a total run time of approximately 960s (approximately 3%). The hardware JBI’s handling of the 2000 run shows it outperforms the 1000 run – this is likely due to one of the HW/fast runs taking an inexplicable extra 16,000ms to execute, inflating the average by approximately 6,000ms. With the increased loading of the 2000 run, the impact of a slower subscriber decreased due to the increased packet replication and extra routing.



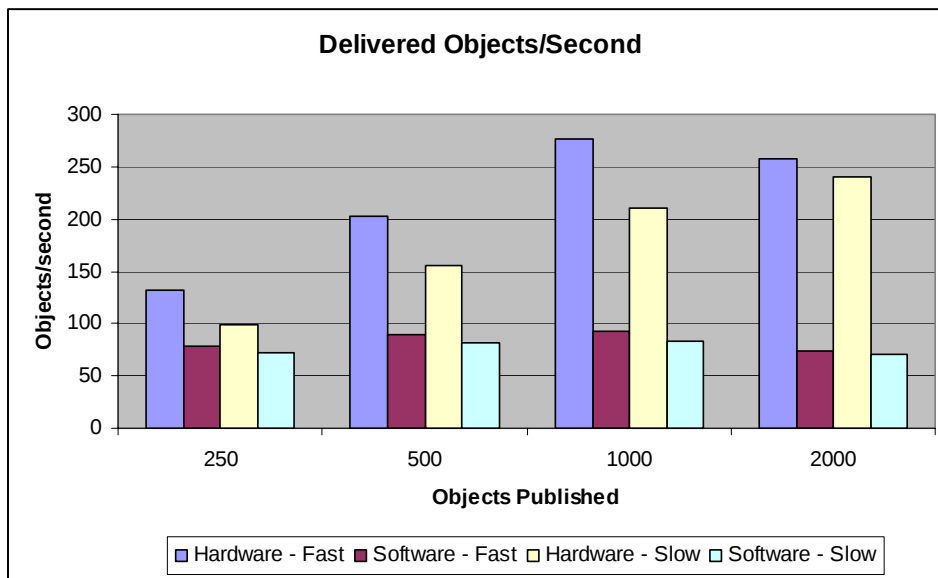
**Figure 15 - Performance Difference of "Slow" Subscriber**

Fitting a curve to both hardware and software JBI results provides insight into behavior beyond the tested 2000 subscribing nodes. According to the software and hardware analysis shown in Figure 14 (c) and the curve fitting shown in Figure 16 (below), the data tracks very closely for both the hardware and software cases in comparison of the impact of a fast/slow subscriber. Observing the behavior of both hardware and software curves shows that as the number of subscriber nodes increases, grouping of the slow and fast data points converges.



**Figure 16 - Fast/Slow Subscriber Trend**

The delivered Objects/Second speed shown in Figure 17 highlights that hardware provides a much higher throughput than software. Hardware – Fast and Hardware – Slow are within the increasing range of 98.7obj/s and 276.9obj/s whereas Software – Fast and Software – Slow are within a steadier range of 71.0obj/s and 92.6obj/s. The experiments best performance is at the 1000 publishing level where three out of the four experiments have their highest object/second processing speed.



|                   |          |          |          |          |
|-------------------|----------|----------|----------|----------|
|                   | 250      | 500      | 1000     | 2000     |
| Objects Published | 250      | 500      | 1000     | 2000     |
| Objects Delivered | 1312     | 4740     | 17520    | 69327    |
| Hardware - Fast   | 131.5655 | 202.7037 | 276.8516 | 257.4834 |
| Software - Fast   | 78.70373 | 90.12168 | 92.56406 | 73.29777 |
| Hardware - Slow   | 98.66557 | 155.8994 | 209.9069 | 240.9237 |
| Software - Slow   | 71.82744 | 82.03231 | 83.07767 | 70.95165 |

**Figure 17 - Delivered Objects/Second**

Furthermore, the hardware JBI data shows that as the number of subscribers increases, processing time per object decreases – or that the hardware JBI is more efficient with heavier loading where subscriber overlap is higher (e.g. object replication increases as a single published object fulfills multiple subscriptions). The software JBI has an initial decrease in processing time per object as the number of subscribers trends toward 1000 subscribers as it is able to efficiently handle the offered load. After the 1000 subscriber point, the software JBI trends upward indicating that a higher number of subscribers is causing subscriber overlap to be higher and the software JBI is paying a greater price to handle object replication.

## 7.2 Steady Increase in Subscribers Case

This experiment tested the hardware and software JBI's ability to handle a steadily increasing subscriber load. The first run had one publisher and two subscribers, the second run had one publisher and three subscribers, and the last run had one publisher and four subscribers. The number of subscribed/published objects was constant (factor \* 1000) and was tested at factors 2, 3, and 4 (2x1000, 3x1000, and 4x1000). The expected increase in processing speed would be at worst linear indicating that increased load was handled in a similar fashion to lighter loading levels. A non-linear **increase** in processing time indicates that a particular JBI is getting "behind" in its processing and not processing load increases as efficiently as was possible with a smaller load.

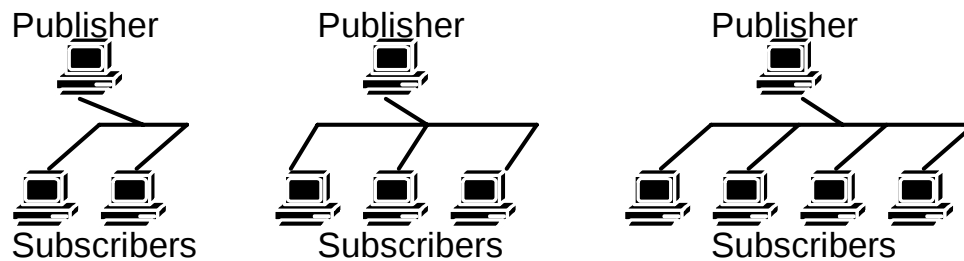
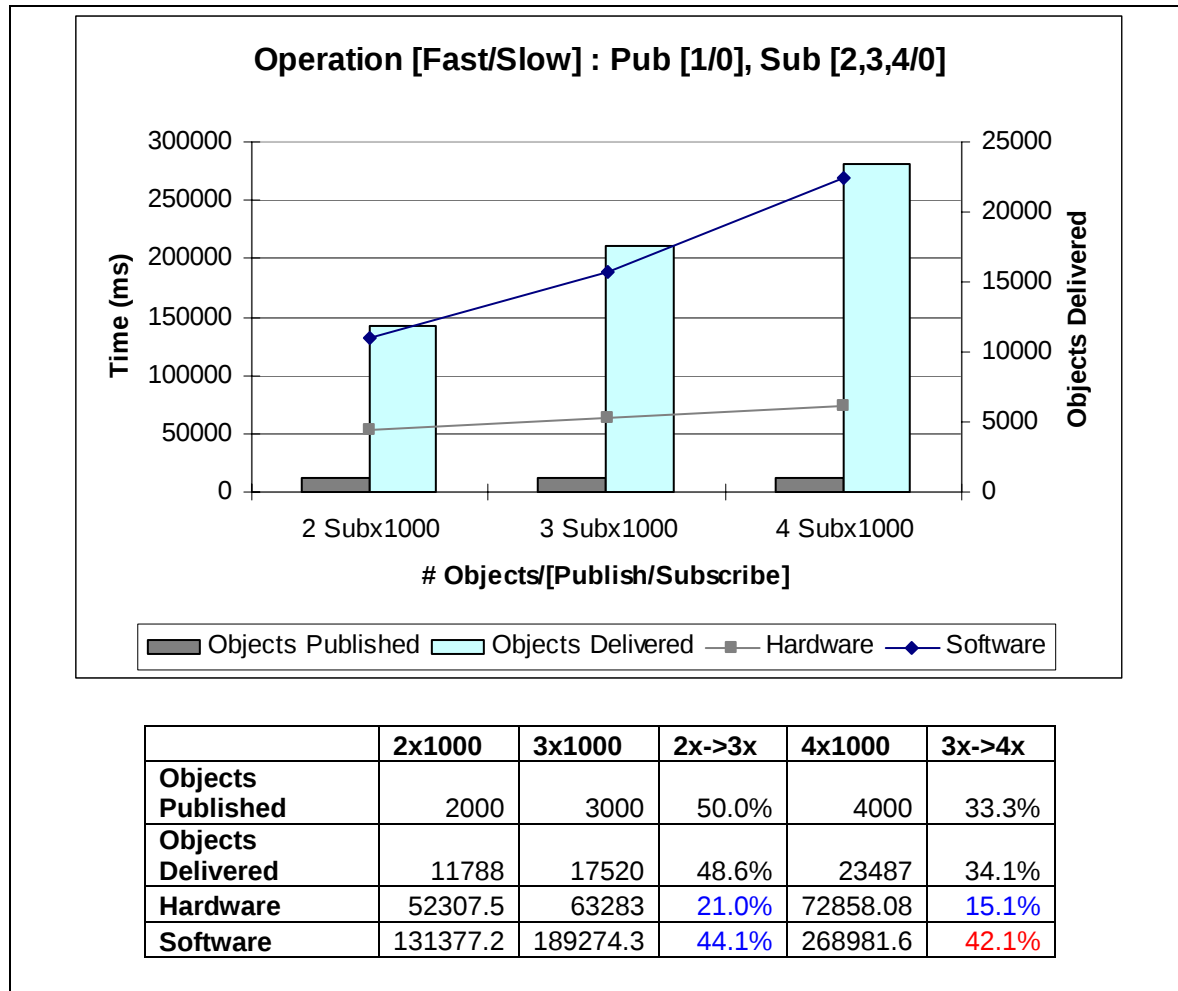


Figure 18 - Steady Subscriber Increase Network Setup

As is shown in Figure 19, the objects published increases by 50% and 33% across the experimental range – this is mirrored by the 49% and 34% of objects delivered across the same range. Therefore, the two sets of experiments will determine if a similar behavior is evident for either software or hardware JBI.



**Figure 19 - Steady Subscriber Increase Data**

### 7.2.1 Software JBI

Figure 19 illustrates that the software JBI experiences a much steeper increase in processing time than the hardware JBI. The software JBI is showing signs of heading toward a saturation condition as the increase in subscriber load caused a greater than expected increase in processing time. As the “objects delivered” load increases 49% and 34% (11788 to 17520, and 17520 to 23487) processing time increases 44% and 42% respectively. The increased load of the 4x1000 run suffers a greater performance impact than the 3x1000 run based on the increased load. In this case, software JBI performance is degrading with increased subscriber load, as a 49% increase in load causes a 44% increase in time (slightly better than the expected 49%), and a 34% increase in load causes a 42% increase in processing time (expected processing time increase would have been only 34%).

### 7.2.2 Hardware JBI

Figure 19 shows the linear impact of the hardware JBI with increased subscriber load. This increased processing time tracks closely with the increase in subscriber load. The “objects delivered” load increase of 49% and 34% (11788 to 17520, and 17520 to 23487) exhibits only a 21% and 15% increase in processing time. Unlike the software JBI case, the hardware JBI exhibits a better than expected decrease in load behavior: a 49% increase in load causes a 21% increase in processing time (the expected value was 49%), and the 34% increase in load causes a 15% increase in processing time (the expected value was 34%).

### 7.2.3 Analysis

The software JBI non-linear increase highlights potential scalability concerns – as an increase in loading leading to a saturation condition is not ideal. The hardware JBI is more successful in handling the increased load and exhibits a decrease in processing time across the experiment. By examining the trend line shown in Figure 20, the trend for software is increasing as the subscribing load increases, whereas the trend for hardware is decreasing under increased subscribing load. The hardware downward trend relates to efficiencies gained through packet replication in this experimental setup, as packet replication is less costly in terms of processing time in the hardware scenario.

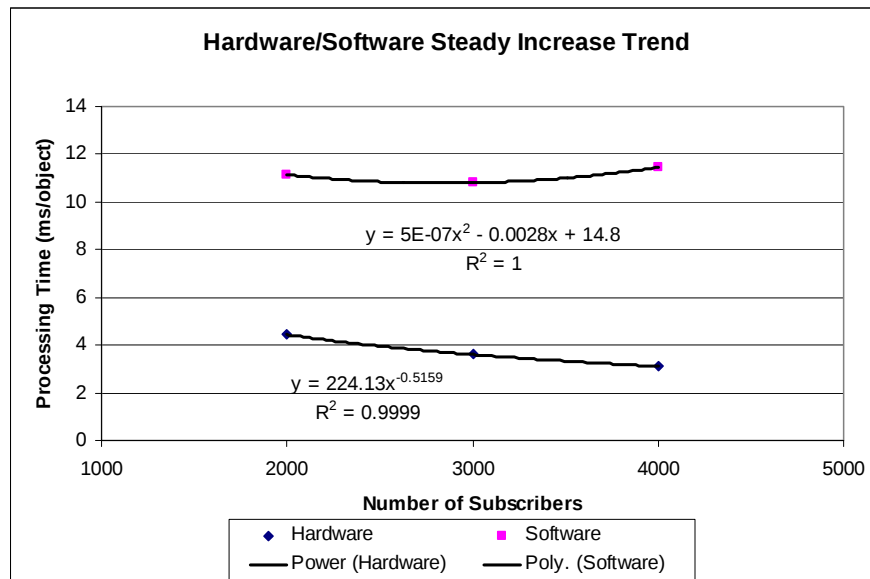


Figure 20 - Steady Increase Trend

Furthermore, hardware JBI data shows that as the number of subscribers increases, processing time per object decreases – or that the hardware JBI is more efficient with heavier loading where subscriber overlap is higher (e.g. object replication increases as a single published object fulfills multiple subscriptions). This data is further supported by Figure 21 which shows that as processing time per object decreases, the number of objects handled per second increases from 225.4obj/s to 322.4obj/s. The software JBI has a relatively steady processing time per object that is trending higher and is further illustrated by Figure 21 showing that as the processing time per

object fluctuates, the objects/second processing goes from 89.7obj/s to 92.6obj/s and back down to 87.3obj/s.

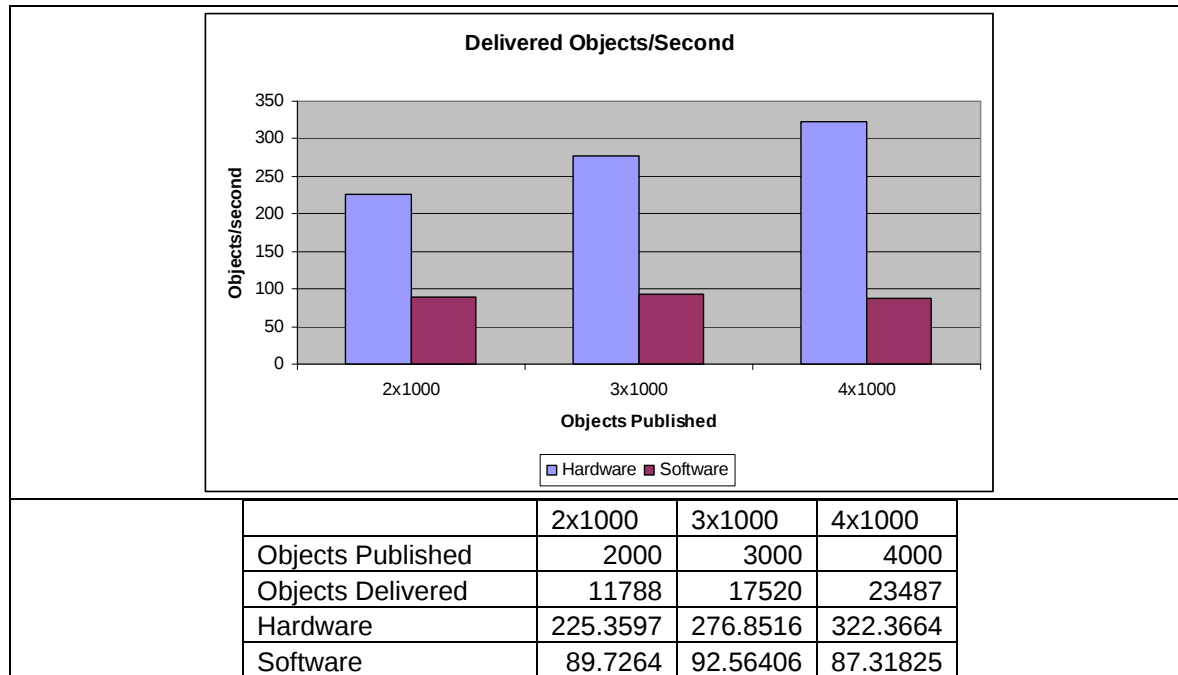


Figure 21 - Delivered Objects/Second

### 7.3 Payload Increase Case

The payload increase experiment determined how much additional processing time was necessary as payload size increased. The experiment ranged over three values, base payload value of approximately 2KB, a medium value of approximately 25KB, and maximum value of approximately 50KB. Payload size is representative of what could be present in a true JBI scenario – a small text message (2KB), a small image or larger text document (25KB), and a couple of small images or one larger image (50KB). Realistically, while multi-megabyte objects may be possible in some scenarios, for the purposes of this experiment object sizes greater than 50KB are ignored.

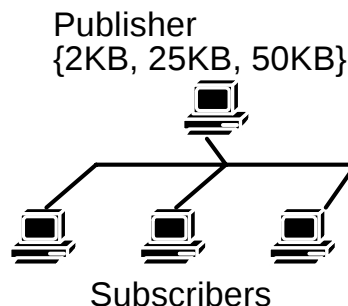


Figure 22 - Payload Increase Network Setup

As is shown by the network diagram of Figure 22, a single publisher sent three payload sizes to each of three subscribers. The scenario published all of one payload size for each experiment – no mixing of sizes occurred inside an experimental iteration. The expectation was that increased payload size would cause minimal impact to the object delivery time and that routing/delivery would be independent of object sizes.

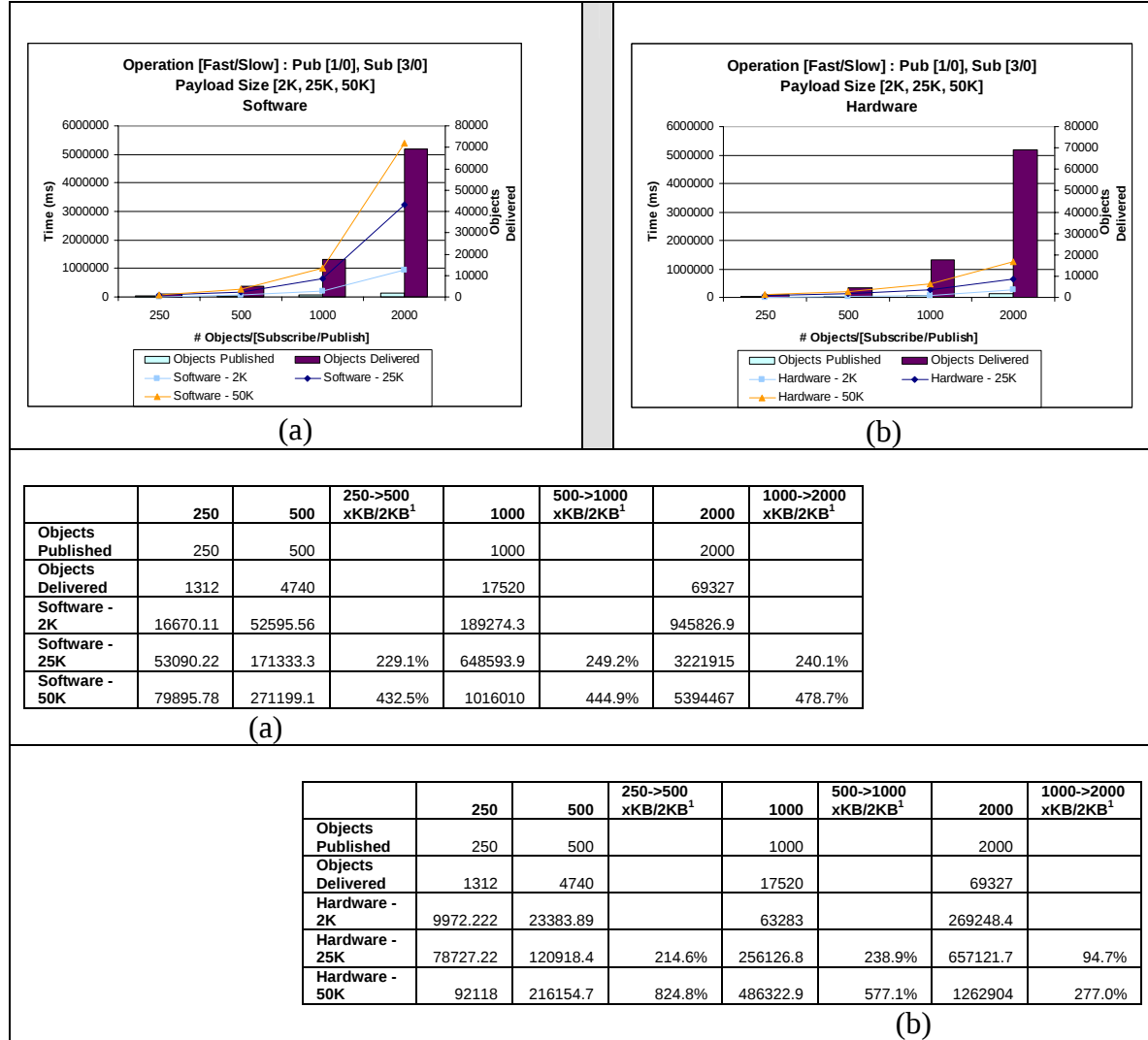


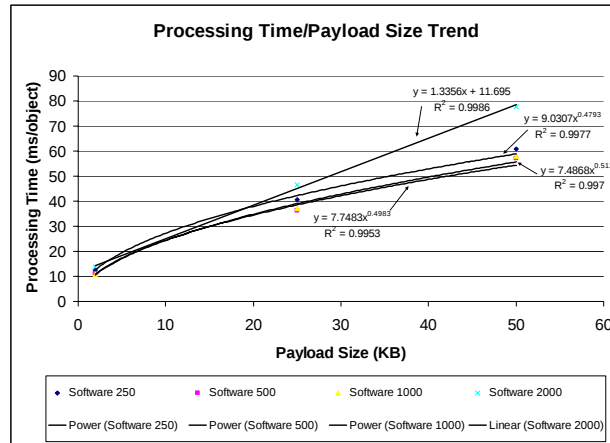
Figure 23 - Payload Increase Data, (a) Software, (b) Hardware

<sup>1</sup> Values are calculated as follows:  $\frac{(Hardware25K3x - Hardware25K2x)}{Hardware2K3x - Hardware2K2x} - 1$  or  $\frac{120918.4 - 78727.22}{23383.89 - 9972.22} - 1$

The numerator is the present object size delivery time and is the difference in delivery time at the two loading levels. The denominator is the prior object size delivery time and is the difference in delivery time at the two loading levels. The fractional comparison of these two values and subtracting one gives a percentage difference in time indicating how much longer the present object size required as compared to the prior object size in terms of delivery time. Greater than 0 means it took comparatively longer, equal to 0 means it took the same amount of time, and less than 0 means it took comparatively less time to do the same action.

### 7.3.1 Software JBI

The software JBI is in general less capable of handling increased subscriber size. As is shown in Figure 23(a), the base software JBI experiment with a 2KB payload performs slightly better than the hardware JBI with a 50KB payload and 2000 subscribers. Furthermore, subsequent experiments with a 25KB payload and a 50KB payload present a drastic degradation in processing speed, most noticeable at the 1000 and 2000 subscription loading-levels.



**Figure 24 - Payload Increase Trend**

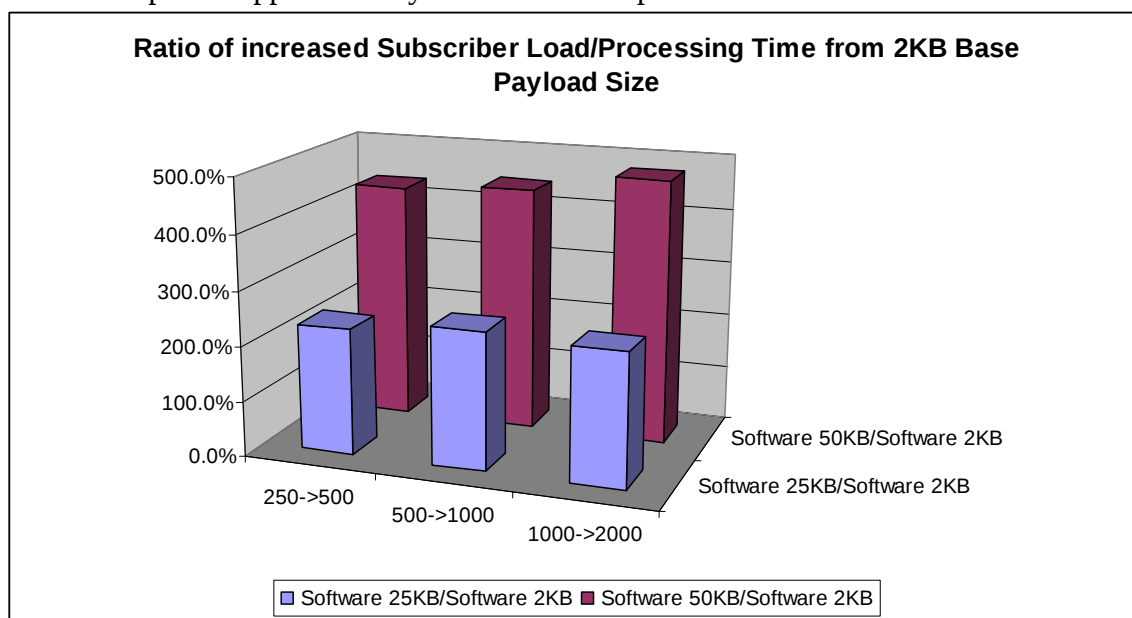
The software JBI trend lines present interesting observations on behavior. The first observation is that the Software 2000 run is linear, while all other runs fit to a power regression curve. The key difference between the four curves is size of the subscribing payload. The data indicate that for 250, 500, and 1000 loading levels processing time is grouped based on payload size. Three curves (250, 500, 1000) track closely and indicate that the software JBI system handles a given object size similarly for a given subscription loading level. A 2KB object size at the 250-2000 loading level requires  $12.1\text{ms/object} \pm 1.2\text{ms/object}$ . Including the Software-2000 data in the averages for the 25KB and 50KB payload size is meaningless due to the large variance introduced. For a 25KB object size at the 250-1000 loading level the average processing time is  $37.9\text{ms/object} \pm 1.9\text{ms/object}$ . Lastly, the 50KB object size at the 250-1000 loading level requires  $58.7\text{ms/object} \pm 1.6\text{ms/object}$  to process. The Software-2000 run shows where the software JBI system is heading – and that is as subscribers and payload size increase processing time increases. The Software-2000 curve tracks the other three curves closely until just after the 20KB object size, after which it increases much more rapidly and takes an additional processing time of  $20\text{ms/object}$  at the 50KB object size.

The 250, 500, and 1000 runs behave similarly and “cost” about the same amount to process – so the JBI system can handle the 2KB-50KB range of data sizes with less than a 1000 subscribers in about equal time. Only at the 2000 loading level does the number of subscribers and payload size impact processing time.

In order to draw conclusions about the efficiency of processing larger payload sizes and determine impact, a 2-axis comparison is performed. Referencing the data table in Figure 23(a)

and the chart in Figure 24, a percentage value is assigned to the processing time as the subscriber load increases from 250->500, 500->1000, and 1000->2000 and the payload increases from 2KB->25KB and 2KB->50KB. This ratio is graphed in Figure 25 (below)– the floor of the 3-D chart is the base 2KB case, the bars indicate for the given payload size at a given loading level, how much longer the system took to process the data as compared to the base case.

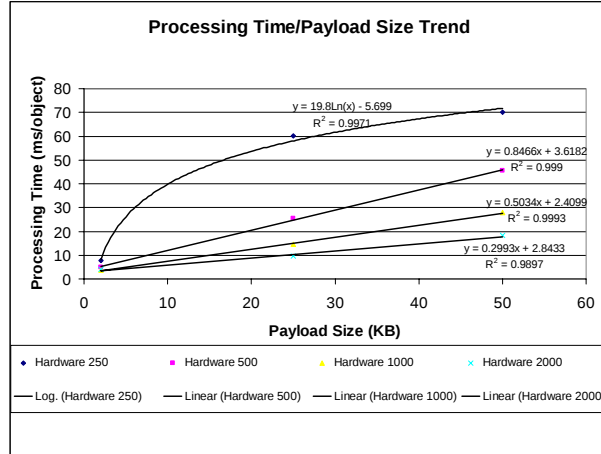
Results are as expected for the software JBI system – as the number of subscribers increase and payload size increases, the system requires longer to process subscriptions. An increase from a 2KB payload to a 25KB payload causes about a 200% increase in processing time, noting that the bars are all approximately level. The increase from the 2KB payload to a 50KB payload causes around a 400% increase and shows a decidedly upward trend on the bars as the loading level increases. The conclusion drawn from this data is that not only does the system perform slower when payload size and number of subscribers increase; the trend is that each loading increase impact is approximately double than the prior increase.



**Figure 25 - Percent Increase over Base Object Size**

### 7.3.2 Hardware JBI

The hardware JBI is in general much more capable of handling increased subscriber size. As is shown in Figure 23(b), the base hardware JBI experiment outperforms the software JBI in all instances with the exception that the hardware 50KB payload is slightly slower than software JBI with a 2KB payload and 2000 subscribers. This is somewhat interesting from the fact that the hardware object size is 25x larger than the software object size and hardware performs only slightly slower; clearly the hardware JBI solution provides a great benefit in increasing performance.



**Figure 26 - Payload Increase Trend**

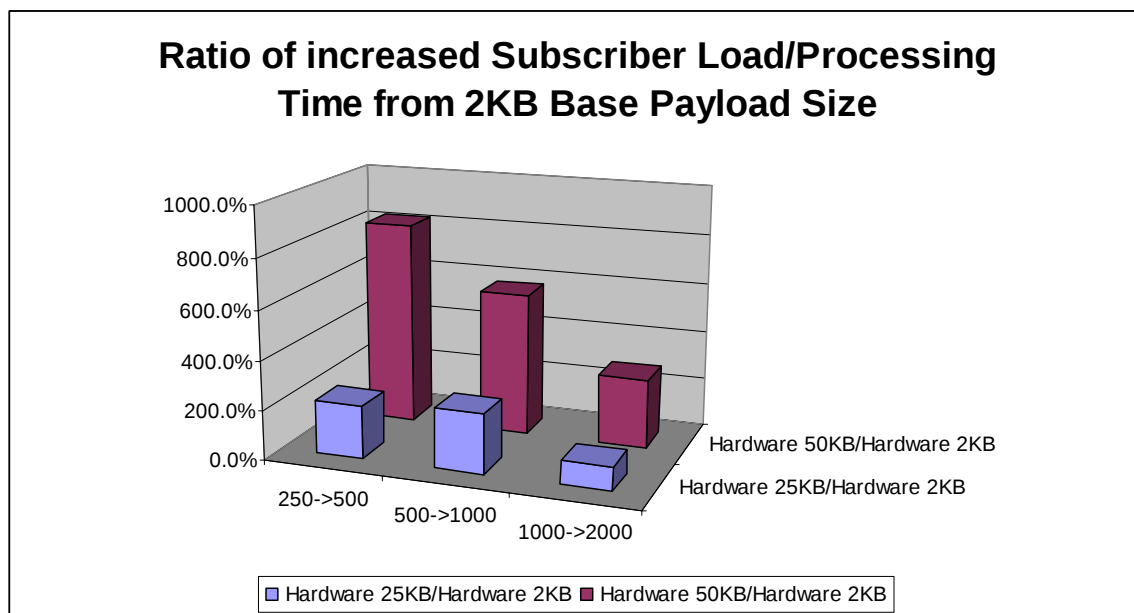
Trend lines shown above in Figure 26 are interesting for the behavior illustrated. All regression lines except for Hardware 250 were linear; the Hardware 250 case was logarithmic. As was the case in the software analysis above, the key difference between the trends is object size, but in this case, the smaller object size is behaving differently. The logarithmic nature of the curve is explained as follows – at small loading levels (250 subscribers in this case), increased payload size causes a dramatic increase as the object size goes from 2KB->25KB, and a much more gradual impact as the object size goes from 25KB->50KB. In this case, the loading level is such that there is minimal duplication of published packets (ratio is approximately 1:5 “*objects published:objects delivered*”) so hardware pays a steeper cost for routing each object and this cost is more costly as object size increases.

As subscription loading levels increase, duplicated packet ratio increases (1:9 @ 500, 1:18 @ 1000, 1:34 @ 2000) and the curve clearly indicates that hardware performs better under these increased loading scenarios. The linear trend lines show that increasing the subscribers to 500, then 1000, and finally 2000 yields in improvement in object routing speed, as processing time per object is decreasing. This is evidence that the hardware JBI is leveraging its ability to rapidly replicate and route packets to fulfill large subscription requests from a single publication – causing a decrease in processing time as loading/packet sizes increase.

As was stated in the prior section, in order to draw a conclusion about the efficiency of processing larger payload sizes and determine the impact, a 2-axis comparison is performed. Referencing the data table in Figure 23, a percentage value is assigned to processing time as subscriber load increases from 250->500, 500->1000, and 1000->2000 and payload increases from 2KB->25KB and 2KB->50KB. This ratio is graphed in Figure 27 – the floor of the 3-D chart is the base 2KB case, the bars indicate for the given payload size at a given loading level, how much longer the system took to process the data as compared to the base case.

In the hardware JBI experiment the results are surprising – the increase in the number of subscribers and payload size actually leads to a dramatic decrease in processing time ratio, most noticeably in comparing the Hardware 50KB/Hardware 2KB ratio. The results presented in the

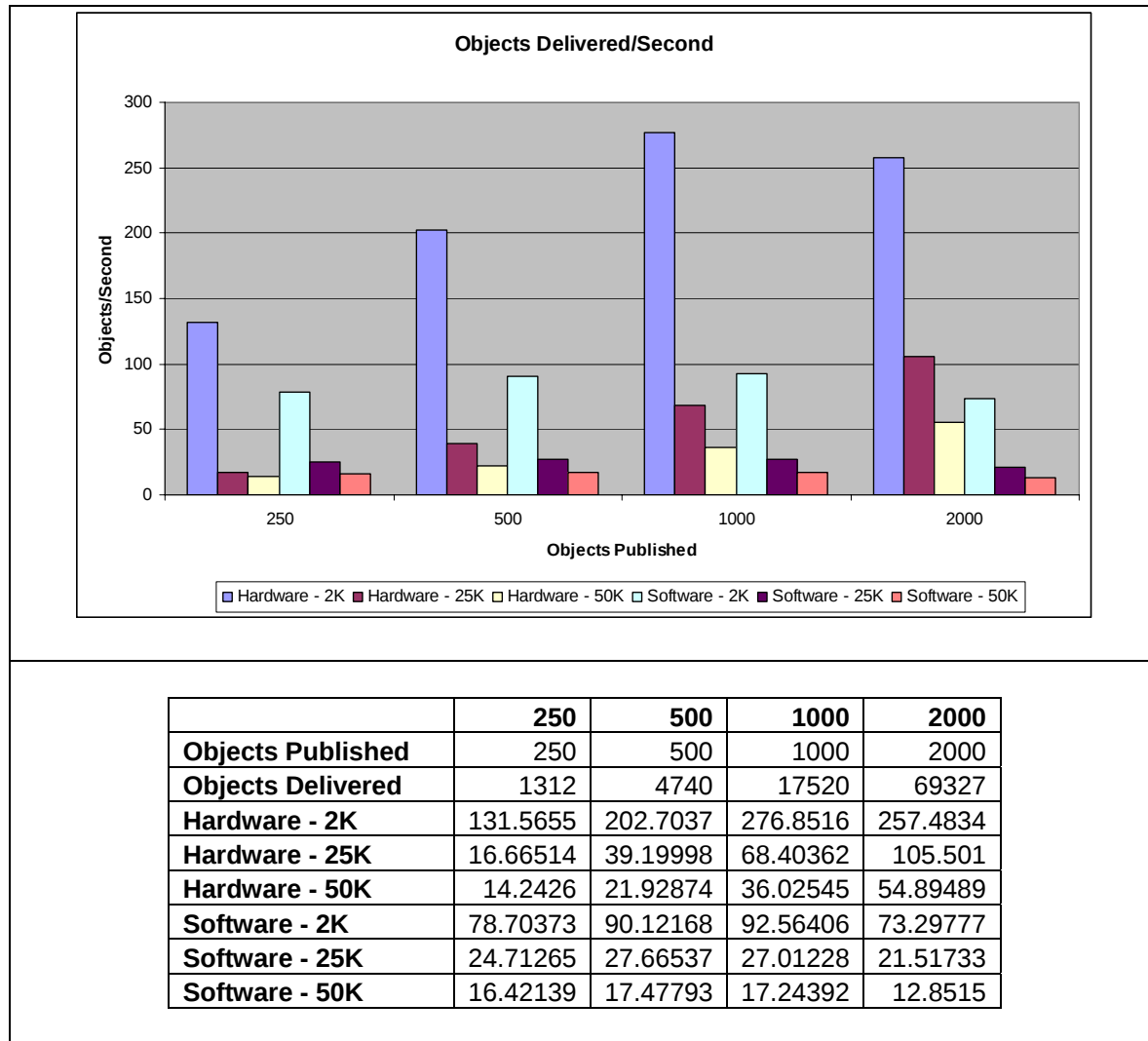
Hardware 25KB/Hardware 2KB experiment have an unusual blip in that there is an increase at the 500->1000 level but otherwise it shows a downward trend much like the Hardware 50KB/Hardware 2KB trend. Hardware 50KB/Hardware 2KB exhibits a clear downward drop in processing time as the load increases due to increased subscribers and greater object size. The conclusion drawn from the data is that as subscriber load increases - the amount of replications also increases – and replications are handled efficiently by the hardware JBI. Increased replication allowed the hardware to excel and increase the effective processing speed even though object size was also increasing. Furthermore, greater object size coupled with increased subscriber workload still allowed for improvements in processing time. In this case, results for the hardware JBI are better than was expected.



**Figure 27 - Percent Increase over Base Object Size**

### 7.3.3 Analysis

The delivered Objects/Second speed shown in Figure 28 highlights that hardware provides a much higher throughput than software. There is a dramatic decrease in objects/second once the payload size increases beyond 2KB for both the hardware and software instantiations. Hardware with a 2KB payload ranges from 131.6obj/s to 257.5obj/s and hardware with a 50KB payload ranges from 14.2obj/s to 54.9obj/s. In the case of software, a 2KB payload ranges from 78.7obj/s to 92.6obj/s, and software with a 50KB payload ranges from 16.4obj/s to 17.2obj/s. The experiments best performance is at the 1000 publishing level for software where three out of the three experiments have their highest object/second processing speed. In the case of hardware, the 2KB case has its best performance at the 1000 level, but both the 25KB and 50KB case perform best at the 2000 level.



**Figure 28 - Objects Delivered/Second**

The impact on the software JBI due to increased object size is substantial. Experimental results indicate that the software JBI suffers a 200% increase in time as object size increases from 2KB to 25KB, and another 200% (400% from 2KB to 50KB) increase as object size increases from 25KB to 50KB. The explanation for this behavior is tied to the software foundation of the JBI – each object must be ingested via the TCP/IP stack, processed AND duplicated at the application layer, before traversing that stack again for transmission. The hardware JBI on the other hand, processes and duplicates objects as an inline network router so it is not impacted by the TCP/IP stack and can use rapid hardware routines for packet duplication. The more efficient hardware implementation of the JBI shows a decrease in relative processing time as the number of objects increases and a much smaller impact as the packet size increases from 2KB to 25KB (200% decreasing to 100%) and 25KB to 50KB (800% decreasing to 275%). Hardware impact is quite substantial as the number of subscribers' increases from 250 to 500, and much less dramatic as the subscribers increase from 1000 to 2000. The subscriber increase from 250 to 500 as object

size increases illustrates that overhead to handle packets outweighs benefits of being able to rapidly distribute/duplicate an object. Whereas, as load increases from 1000 to 2000 with increased object size, overhead associated with processing an object is much smaller than the gain realized by being able to rapidly duplicate and transmit an object to a subscribing node. Therefore, in the hardware JBI case as object size increases and subscriber load increases, impact of the processing decreases as hardware efficiencies are realized, which is somewhat counter-intuitive.

#### **7.4 Steady Increase/Payload Increase Case**

The following experiment expanded and combined the approach taken in 6.2 and 6.3 and determined what happens in the case of payload increases and a steady increase in the number of subscribers. The expected outcome is that increased payload size would cause a minimal impact to object routing/delivery time. The expected increase in processing speed caused by increasing the number of subscribers would be at worst linear indicating that increased load was handled similarly to a prior lighter load. A non-linear increase indicates that particular JBI implementation is getting “behind” in its processing and not processing load as efficiently as was possible with a smaller load. Figure 29 shows the experimental configuration indicating various loading levels. Figure 29 also shows that the number of subscribers was increased by a consistent factor, ranging from 2,3 to 4 for a subscription loading level of 2x1000, 3x1000, and 4x1000 respectively.

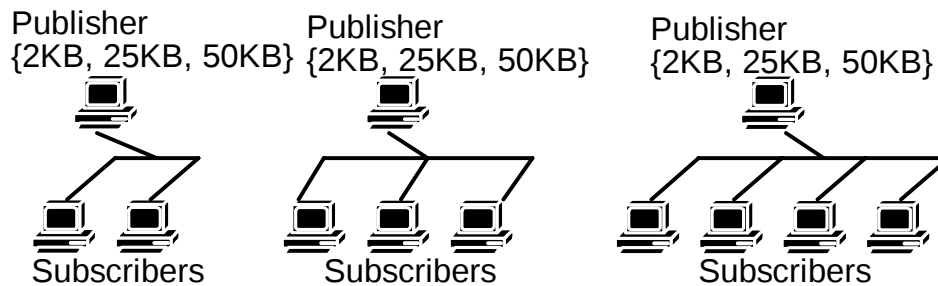


Figure 29 - Steady Increase/Payload Increase Network Setup

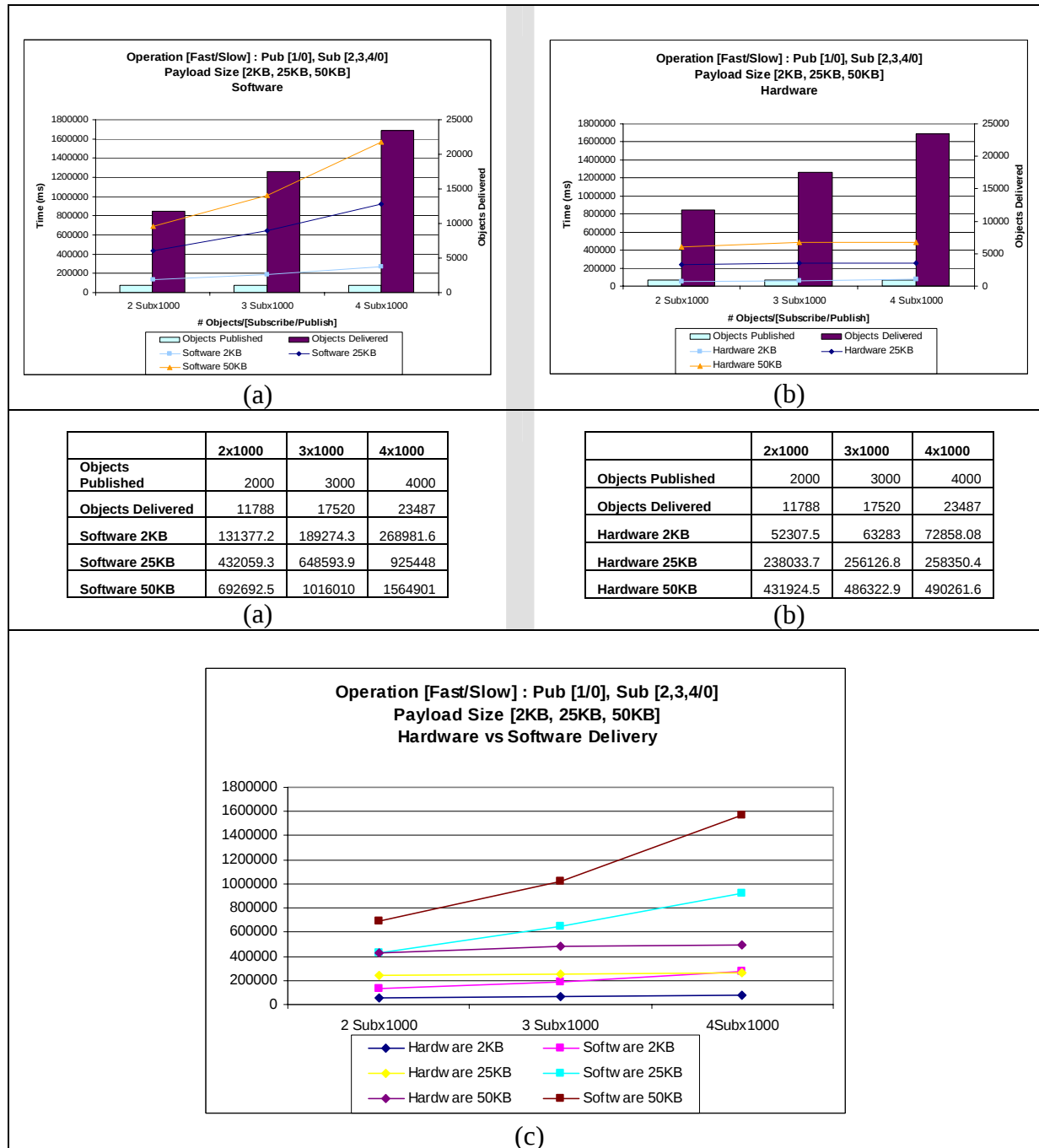


Figure 30 - (a) Software Data, (b) Hardware Data, (c) Combined

### 7.4.1 Software JBI

Figure 30 (a) shows the software JBI experiences a much steeper increase in processing time than the hardware JBI (shown in Figure 30 (b)). The software JBI is showing signs of heading toward a saturation condition as increased subscriber load coupled with increases in payload size causes a greater than expected increase in processing time. Referencing Table 4 below, “objects

delivered” load increases 49% and 34% (11788 to 17520, and 17520 to 23487), processing time increases 44.1% and 42.1% as was shown in section 7.2. For the 25KB payload, processing time increases 50.1% and 42.7% and for the 50KB payload, processing time increases 46.7% and 54.0%. The expected increase was a 49% increase in load causes a 44% increase in time (the 2KB is as expected, the 25KB and 50KB are slightly over at 50.1% and 46.7% respectively). For the 34% increase in load, the expected increase in processing time would have been 31% (all three cases, 2KB, 25KB, and 50KB were higher than 31% at 42.1%, 42.7% and 54% respectively).

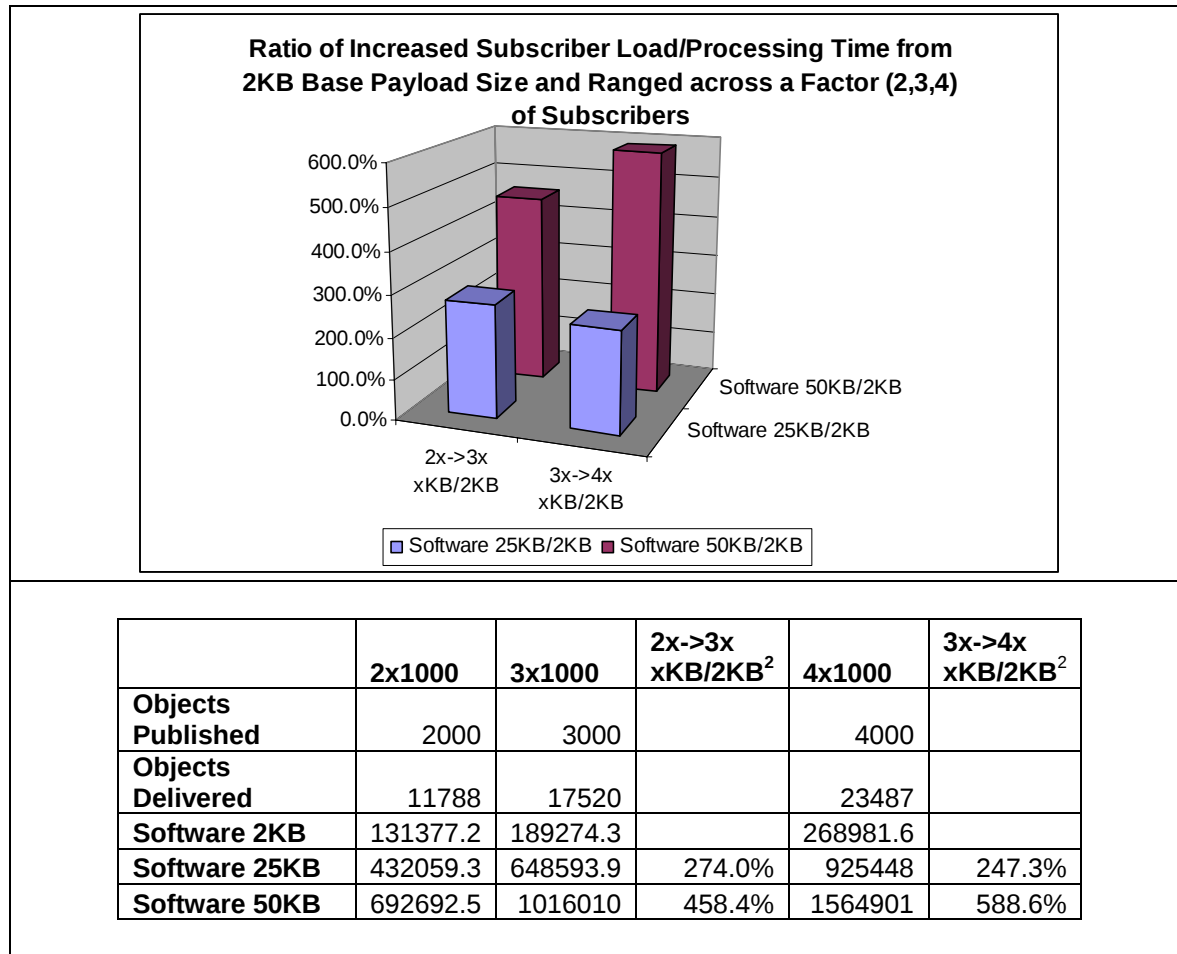
|                          | <b>2x1000</b> | <b>3x1000</b> | <b>2x-&gt;3x</b> | <b>4x1000</b> | <b>2x-&gt;4x</b> | <b>3x-&gt;4x</b> |
|--------------------------|---------------|---------------|------------------|---------------|------------------|------------------|
| <b>Objects Published</b> | 2000          | 3000          | 50.0%            | 4000          | 100.0%           | 33.3%            |
| <b>Objects Delivered</b> | 11788         | 17520         | 48.6%            | 23487         | 99.2%            | 34.1%            |
| <b>Software 2KB</b>      | 131377.2      | 189274.3      | 44.1%            | 268981.6      | 104.7%           | 42.1%            |
| <b>Software 25KB</b>     | 432059.3      | 648593.9      | 50.1%            | 925448        | 114.2%           | 42.7%            |
| <b>Software 50KB</b>     | 692692.5      | 1016010       | 46.7%            | 1564901       | 125.9%           | 54.0%            |

**Table 4 - Percent Increase due to Subscriber Load Increase**

Table 5 below shows the percent increase associated with increased payload size as compared to the baseline 2KB payload based on the 2x1000, 3x1000, and 4x1000 subscriber loads. As object size increases to 25KB, the software JBI undergoes an approximate 239% increase in processing time across the three subscriber loading levels. Examining the increase from 2KB to 50KB shows a processing increase of around 449% across subscriber loading levels. If the increase from 25KB to 50KB is examined, the average processing increase is only about 62% indicating that most of the processing delay has already been realized – or that the ability of the software JBI to route a 50KB payload size is only slightly worse than its ability to route a 25KB payload size. Clearly, payload size generates a larger impact on the performance of the software JBI than just an increase in subscriber load from 2x1000, 3x1000, and 4x1000.

| <b>Software</b> | <b>2KB</b> | <b>25KB</b> | <b>2KB-&gt;25KB</b> | <b>50KB</b> | <b>2KB-&gt;50KB</b> | <b>25KB-&gt;50KB</b> |
|-----------------|------------|-------------|---------------------|-------------|---------------------|----------------------|
| <b>2x1000</b>   | 131377.2   | 432059.3    | 228.9%              | 692692.5    | 427.3%              | 60.3%                |
| <b>3x1000</b>   | 189274.3   | 648593.9    | 242.7%              | 1016010     | 436.8%              | 56.6%                |
| <b>4x1000</b>   | 268981.6   | 925448      | 244.1%              | 1564901     | 481.8%              | 69.1%                |

**Table 5 - Percent Increase due to Payload Size Increase**



**Figure 31 – Software Object Size Percentage Impact**

Combining the results of both the increased payload size and increase in subscriber load allows for a complete impact analysis. Referencing Figure 31 above, we see that going from 2000->3000 (with a 2KB payload) caused an increase of 274.0% and 247.3% for the 25KB payload case, and 458.4% and 588.6% for the 50KB payload case.

Therefore, the software JBI system is performing less efficiently from the impact of both the increased subscriber load and increased object size. Note that payload size has a greater impact on this experiment than the subscriber load based on the above analysis. Combined impact is approximately 260% for the software-25KB and approximately 515% for the software-50KB. From this combined impact, the increase to 3x->4x took much longer to process than the increase

<sup>2</sup> Values are calculated as follows:  $\frac{(\text{Software}25K3x - \text{Software}25K2x)}{\text{Software}2K3x - \text{Software}2K2x} - 1$  or  $\frac{648593.9 - 432059.3}{189274.3 - 131377.2} - 1$

The numerator is the present object size delivery time and is the difference in delivery time at the two loading levels. The denominator is the prior object size delivery time and is the difference in delivery time at the two loading levels. The fractional comparison of these two values and subtracting one gives a percentage difference in time indicating how much longer the present object size required as compared to the prior object size in terms of delivery time. Greater than 0 means it took comparatively longer, equal to 0 means it took the same amount of time, and less than 0 means it took comparatively less time to do the same action.

of 2x->3x indicating that the software JBI is getting behind in its ability to handle subscriber load.

## 7.4.2 Hardware JBI

Figure 30 (b) shows that impact on the hardware JBI with increased load and increased payload size is relatively small as compared to the software JBI (Figure 30 (a)). The 2KB payload size results were as expected, but the results for the 25KB and 50KB payload size generated better than expected results.

As was the case for the software JBI, “objects delivered” load increases 49% and 34% respectively (11788 to 17520, and 17520 to 23487), processing time increases 21.0% and 15.1% as shown in section 7.2. As shown in Table 6, for the 25KB payload, the processing time increases 7.6% and 0.9% and for the 50KB payload, the processing time increases 12.6% and 0.8%. The increases shown for the 25KB and 50KB are better than expected as they not only show a clear downward trend in the processing time as the load on the system increases, but also indicate that the hardware JBI becomes more efficient under increased load and suffers minimal performance impact related to an increased number of subscribers.

|                          | 2x1000   | 3x1000   | 2x->3x | 4x1000   | 2x->4x | 3x->4x |
|--------------------------|----------|----------|--------|----------|--------|--------|
| <b>Objects Published</b> | 2000     | 3000     | 50.0%  | 4000     | 100.0% | 33.3%  |
| <b>Objects Delivered</b> | 11788    | 17520    | 48.6%  | 23487    | 99.2%  | 34.1%  |
| <b>Hardware 2KB</b>      | 52307.5  | 63283    | 21.0%  | 72858.08 | 39.3%  | 15.1%  |
| <b>Hardware 25KB</b>     | 238033.7 | 256126.8 | 7.6%   | 258350.4 | 8.5%   | 0.9%   |
| <b>Hardware 50KB</b>     | 431924.5 | 486322.9 | 12.6%  | 490261.6 | 13.5%  | 0.8%   |

Table 6 - Percent Increase due to Subscriber Load Increase

Table 7 below shows the percent increase associated with the increased payload size as compared to the 2KB baseline payload based on 2x1000, 3x1000, and 4x1000 subscriber loads. As object size increases to 25KB, the hardware JBI has an approximate 305% increase in processing time across the three subscriber loads. Looking at the increase from 2KB to 50KB the processing time increase is approximately 650% across subscriber loading levels. As was the case in the software JBI, the hardware JBI undergoes an approximate 85% increase in processing time as object size increases from 25KB to 50KB – or that the hardware JBI is able to route a 50KB payload size slightly slower than a 25KB payload size. The performance impact on the hardware JBI is more pronounced due to the payload size increase than to subscriber load increase.

| Hardware      | 2KB      | 25KB     | 2KB->25KB | 50KB     | 2KB->50KB | 25KB->50KB |
|---------------|----------|----------|-----------|----------|-----------|------------|
| <b>2x1000</b> | 52307.5  | 238033.7 | 355.1%    | 431924.5 | 725.7%    | 81.5%      |
| <b>3x1000</b> | 63283    | 256126.8 | 304.7%    | 486322.9 | 668.5%    | 89.9%      |
| <b>4x1000</b> | 72858.08 | 258350.4 | 254.6%    | 490261.6 | 572.9%    | 89.8%      |

Table 7 - Percent Increase due to Payload Size Increase

Combining the results of both increased payload size and increase in subscriber load allows for a complete analysis of impact. Referencing Figure 32 below, the graph is significantly different then what was present in the software JBI graph shown in Figure 31. The subscriber load increases from 2000->3000 (with a 2KB payload) caused an increase of 64.8% and a decrease of -76.8% for the 25KB payload case, and a 395.6% increase followed by a -58.9% decrease for the 50KB payload case. These results are definitely unusual but are a based on the hardware JBI's unique ability to handle packets at line speed for rapid routing, duplication, XML processing, and subscription fulfillment. Although the results are somewhat counter-intuitive, the hardware JBI is clearly becoming more efficient in its ability to handle the increased subscriber load coupled with increased payload size.

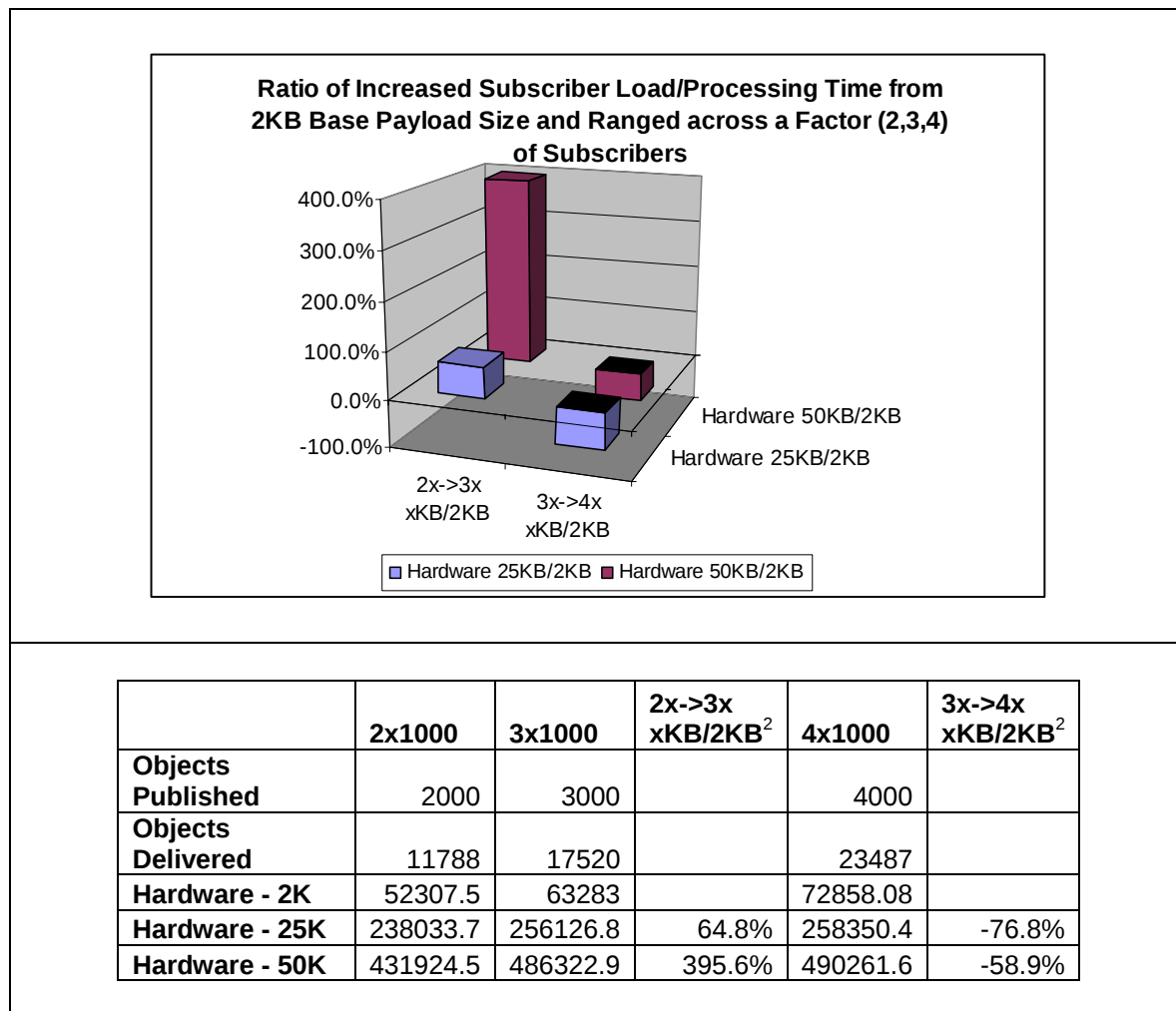
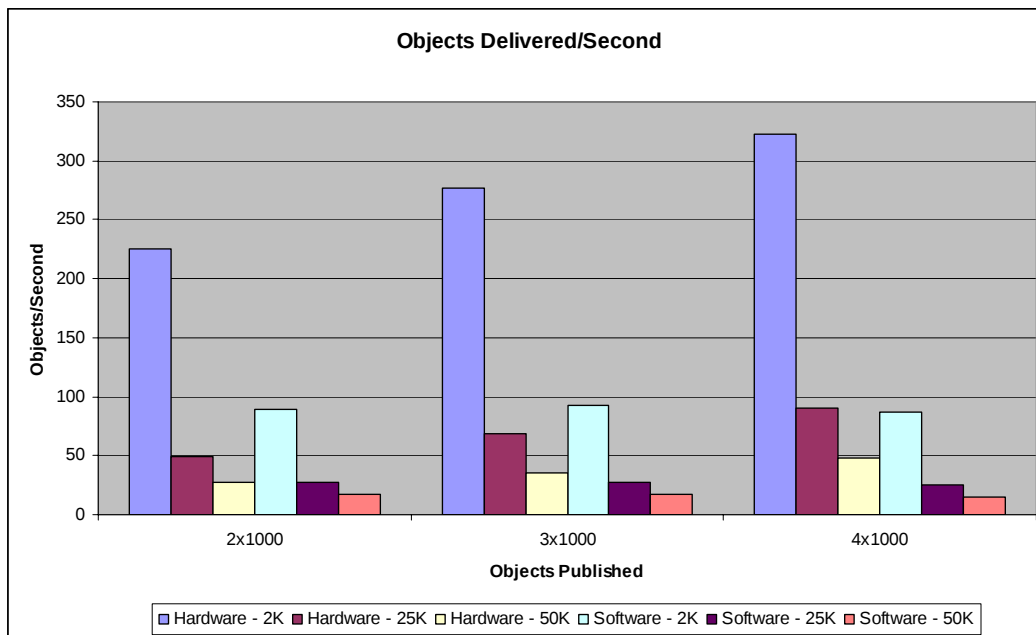


Figure 32 - Hardware Object Size Percentage Impact

### 7.4.3 Analysis

The delivered Objects/Second speed shown in Figure 28 highlights that hardware provides a much higher throughput than software. The increased loading of this experiment contributes

minimally to objects delivery processing times, but as was shown in section 7.3, there is a dramatic decrease in objects/second once the payload size increases beyond 2KB for both the hardware and software instantiations. Hardware with a 2KB payload ranges from 225.4obj/s to 322.4obj/s and hardware with a 50KB payload ranges from 27.3obj/s to 47.9obj/s. These processing speeds per object are continuing the upward trend shown in section 7.3 as related to loading. In the case of software, a 2KB payload ranges from 89.7obj/s to 92.6obj/s, and software with a 50KB payload ranges from 17.0obj/s to 17.2obj/s. As compared to section 7.3, the hardware exhibits a further increase in the number of objects processed per second with increased subscriber load, whereas the software exhibits similar behavior to the results in section 7.3 as the range is almost identical regardless of the increased subscriber load.



|                          | 2x1000   | 3x1000   | 4x1000   |
|--------------------------|----------|----------|----------|
| <b>Objects Published</b> | 2000     | 3000     | 4000     |
| <b>Objects Delivered</b> | 11788    | 17520    | 23487    |
| <b>Hardware 2KB</b>      | 225.3597 | 276.8516 | 322.3664 |
| <b>Hardware 25KB</b>     | 49.52241 | 68.40362 | 90.91141 |
| <b>Hardware 50KB</b>     | 27.29181 | 36.02545 | 47.90708 |
| <b>Software 2KB</b>      | 89.7264  | 92.56406 | 87.31825 |
| <b>Software 25KB</b>     | 27.28329 | 27.01228 | 25.37906 |
| <b>Software 50KB</b>     | 17.01765 | 17.24392 | 15.00862 |

**Figure 33 - Objects Delivered/Second**

The data supports the conclusion that payload size affects performance more than increased subscriber loading for both hardware and software JBI systems. Referencing Figure 30 (c) it is clear that the software JBI is not nearly as capable as the hardware JBI in its ability to handle this experiments increased loading. The relatively flat lines associated with the hardware JBI are in stark contrast to the steeply increasing performance figures of the software JBI.

## 7.5 Heavy Load Case

The following experiment generated a heavy load on subscriber nodes with multiple publisher nodes. Increased load was created through two actions – increase to three the number of concurrent object publishers (one object type per publisher) and have each subscriber subscribe to all three object types. The publishers publish each object type simultaneously. Subscribers have three subscription operations in progress subscribing to each unique published object. This experiment will give visibility into how a JBI system would handle a large number of subscribers and how efficiently data can be distributed to multiple publishing nodes. Figure 34 shows the network diagram indicating the object type for each publisher node and the multiple object types at each subscribing node.

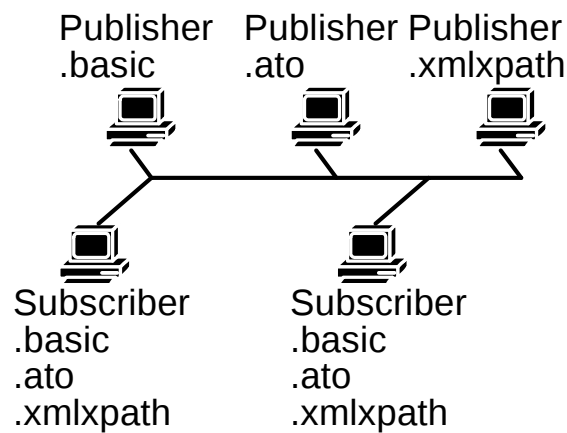
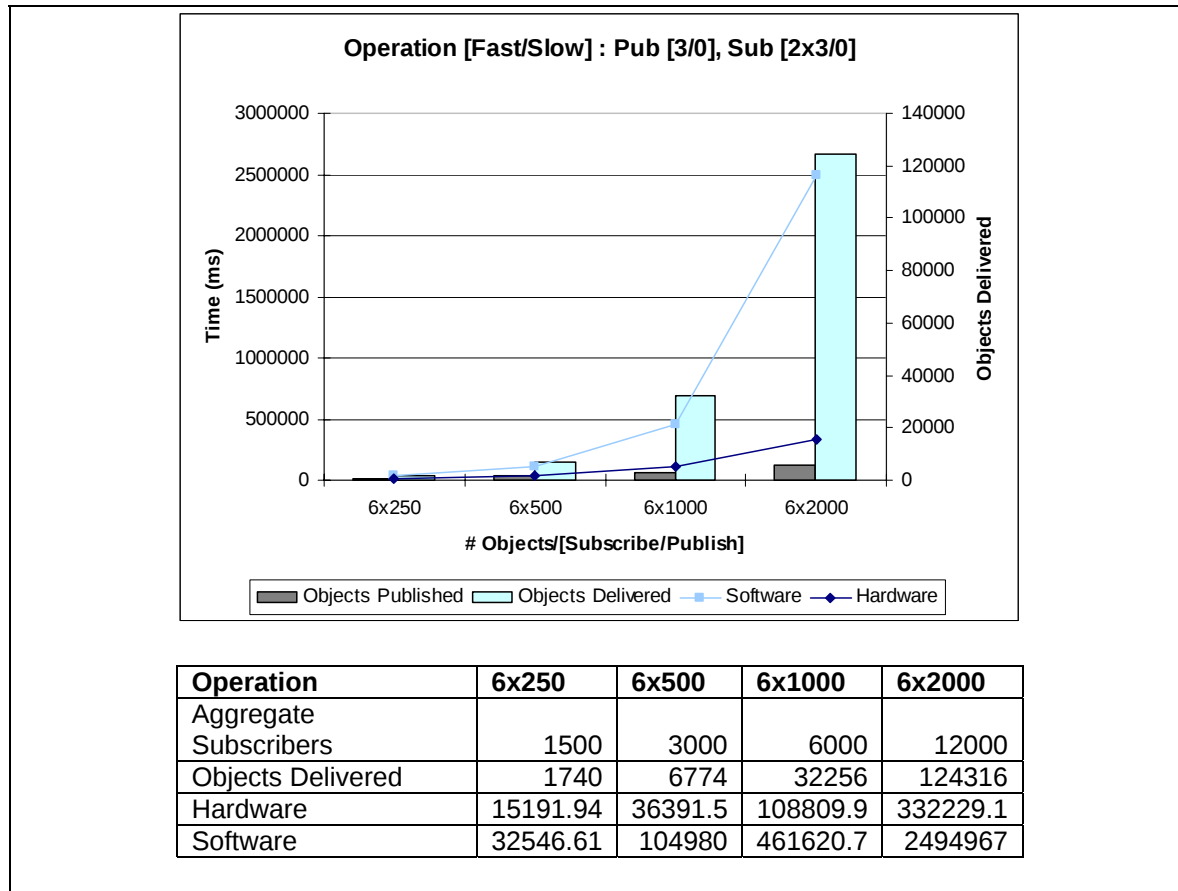


Figure 34 - Heavy Load Network Setup

Figure 35 indicates the actual loading that occurred. The total number of subscriptions spread between the two subscribing nodes was 1500, 3000, 6000, and 12000. The number of ObjectsDelivered ranged from 1740 to 124316 so clearly there was a large load present.



**Figure 35 - Heavy Load Data**

### 7.5.1 Software JBI

Software JBI data points shown in Figure 35 were approximately 2.1x slower than the hardware JBI for the 6x250 loading level, and almost 7.5x slower for the 6x2000 loading level. Clearly, as the load increases, the software based system struggles to maintain processing speed. The loading increase of 1500 to 12000 subscription requests (8x increase) generates a 76x increase in the time required to handle the subscriptions. The substantial increase in load leads to the much greater increase in processing time.

### 7.5.2 Hardware JBI

Hardware JBI results presented in Figure 35 exhibited relatively steady behavior regardless of loading. The example above shows that even with the subscription load increased from 1500 to 12000 objects (8x) causing a large increase in subscribed objects (1740 to 124316), the hardware realized only a 2x increase in time required to handle subscriptions. Hardware efficiently handled the increased load, both in the processing of the publish/subscribe operations, but also in the duplication/routing decisions that occurred.

### 7.5.3 Analysis

The heavy load case illustrates a number of factors on how both systems scale under heavy load. In the software JBI instance, as load increases, software gets further and further behind in its ability to process the publication matches, and ultimately suffers from a 76x increase in processing time from the 6x250 loading case to the 6x2000 case. On the other hand, hardware experiences a roughly 2x increase in processing time for the same load increase. The hardware is able to realize efficiencies that just are not possible with software, related to the ability to process information at line speed, avoid the TCP/IP stack, and perform internal duplication of objects for subscription fulfillment. Additionally, the hardware's XML processing ability comes into play as a large subscription tree has minimal impact on how quickly published objects are routed to a subscribing client.

Furthermore, this experiment brought out a unique behavior in that the *.ato* object being delivered was in five of the eight cases smaller than the number of published objects (so there was little or no duplication and many published objects were just ignored since there was not a corresponding subscription).

| Object | ObjectType | ObjRecv | Ratio |
|--------|------------|---------|-------|
| 250    | .basic     | 248     | 1:1   |
|        | .xmlxpath  | 550     | 1:2   |
|        | .ato       | 72      | 1:1/4 |
| 500    | .basic     | 1149    | 1:2   |
|        | .xmlxpath  | 1909    | 1:4   |
|        | .ato       | 329     | 1:1/2 |
| 1000   | .basic     | 5616    | 1:5   |
|        | .xmlxpath  | 9553.5  | 1:10  |
|        | .ato       | 958.5   | 1:1   |
| 2000   | .basic     | 22371.5 | 1:10  |
|        | .xmlxpath  | 36672   | 1:18  |
|        | .ato       | 3114.5  | 1:1.5 |

**Table 8 - Ratio of Published Objects to Delivered Objects**

The ratios for this experiment are shown in Table 8 and contribute to the unique experimental outcome. In the case of the *.basic* object, the ratio grew from 1:1, 1:2, 1:5, to 1:10. In the case of the *.xmlxpath* object, the ratio grew 1:2, 1:4, 1:10, to 1:18. Lastly, in the case of the *.ato* object, the ratio was 1:0.25, 1:0.5, 1:1, to 1:1.5. The introduction of the slower growing *.ato* object ratio into this experiment caused the results to trend down dramatically and led to the decrease shown in Figure 36 for the 6x250, 6x500, and 6x1000 runs. The *.ato* object had a smaller number of delivered objects per offered load and this allowed the system to consume a greater number of delivered objects from the other two publishers. For the object range of 6x250 through 6x1000 subscriber load and duplication where small enough that the system was able to process the offered load somewhat faster than for the prior loading level (trend line is downward). Only at the 6x2000 where the object duplication ratio was maximized for two of the three cases and greater than one for the last case, did the time trend reverse and head higher for

the software JBI. Note that the lower values present because of low subscription fulfillment attributed to the *.ato* object type imply that the averages would have increased considerably had the *.ato* object type behaved as either of the other two object types (e.g. duplication ratios of either *.basic* and *.xmlxpath*).

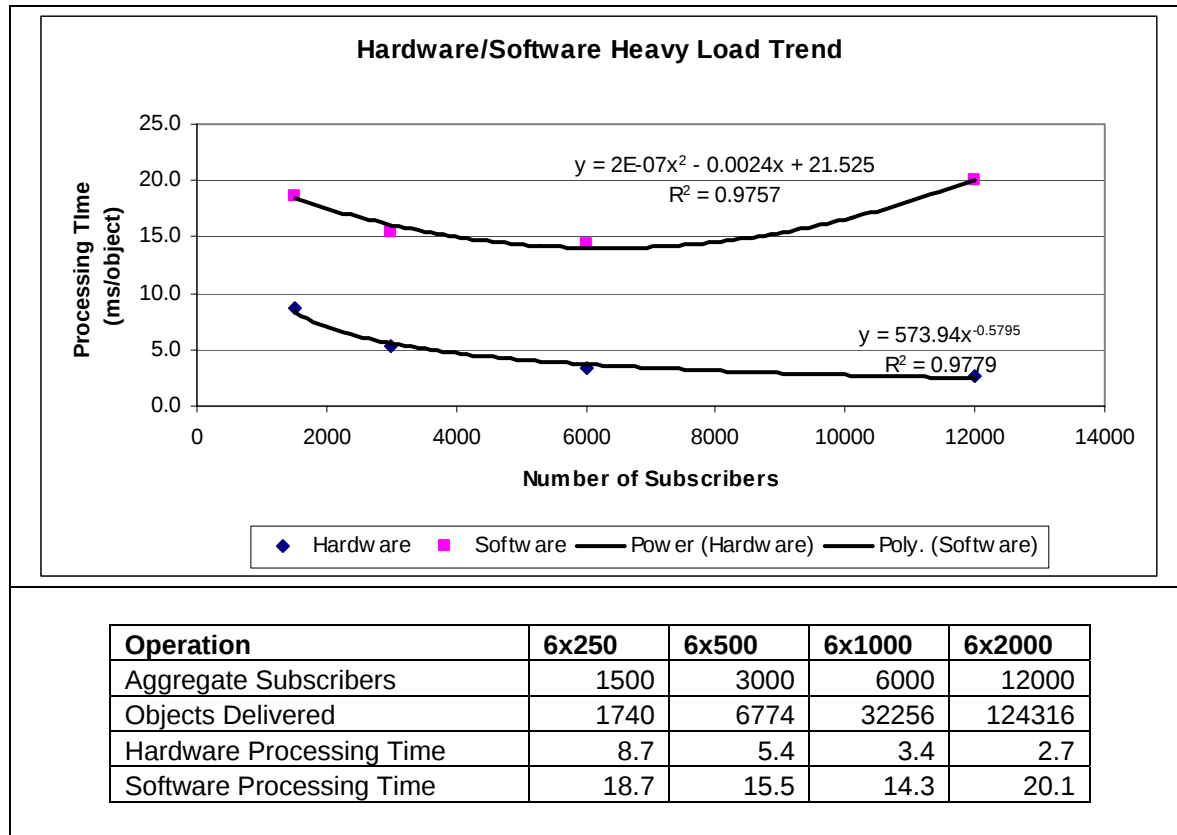
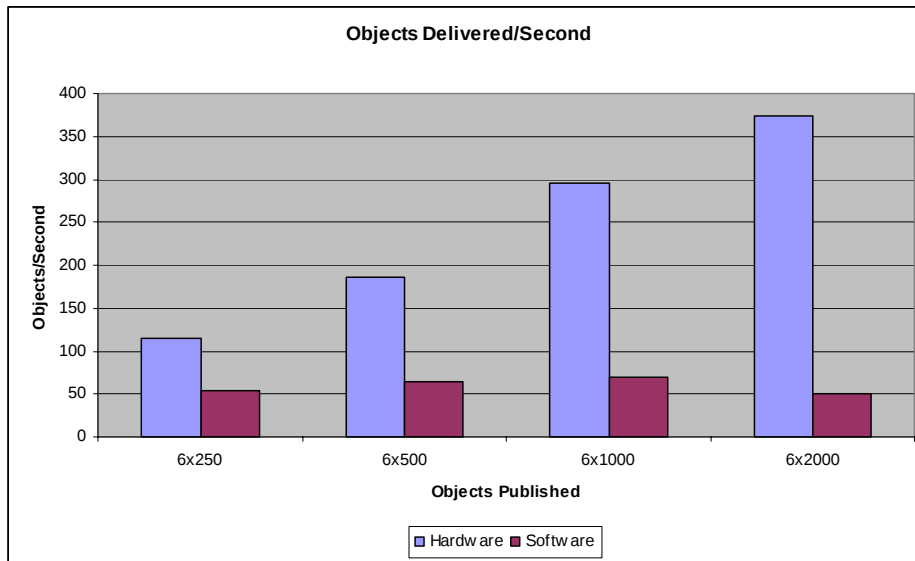


Figure 36- Heavy Load Trend

Upon examining the trend line shown in Figure 36, it appears that the software JBI is trending up under increased load after seeming to trend downward initially. The initial downward trend can be attributed to how the six subscriber nodes are distributed – three subscribers on each client with a total of 6x250, 6x500, and 6x1000 per subscriber – where the duplication ratios were small and the actual *ObjectsDelivered* load was also relatively small. Only at the 6x2000 run does the duplication ratio increase sufficiently based on the number of object delivered that the trend line increases as would be expected. In the hardware JBI instance, we see a steady trend downward. This trend downward is counter-intuitive, but based on the fact that the hardware get more efficient with the greater load and quickly handles the replications that occur, this is actually the expected result.

From an objects delivered/second perspective, Figure 37 highlights that hardware provides a much higher throughput than software. Hardware is within the increasing range of 114.5obj/s and 374.2obj/s whereas Software is within a steadier range of 53.5obj/s and 69.9obj/s. The

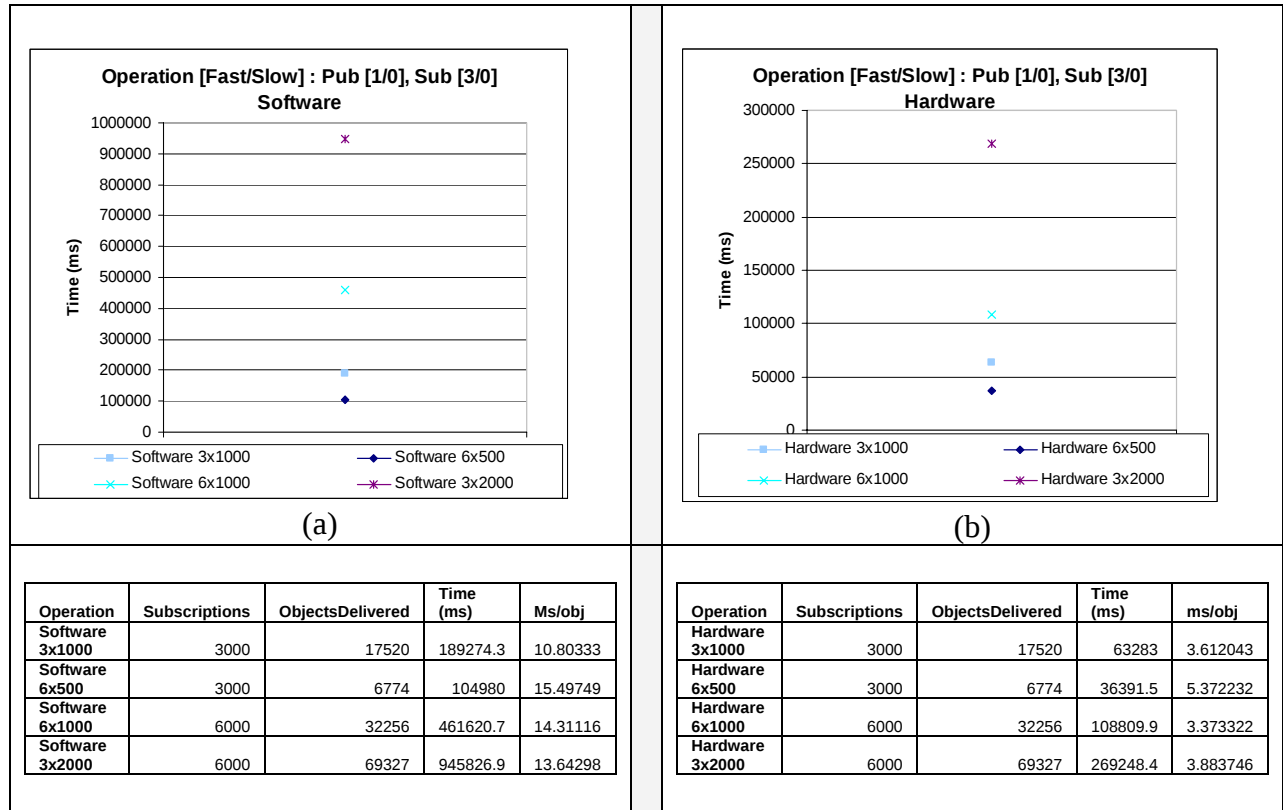
heavy subscriber load causes little impact on the hardware, but causes the software to publish fewer objects per second when compared to the prior experiments at this payload size.



| Operation             | 6x250    | 6x500    | 6x1000  | 6x2000   |
|-----------------------|----------|----------|---------|----------|
| Aggregate Subscribers | 1500     | 3000     | 6000    | 12000    |
| Objects Delivered     | 1740     | 6774     | 32256   | 124316   |
| Hardware              | 114.5344 | 186.1424 | 296.444 | 374.1876 |
| Software              | 53.46179 | 64.52658 | 69.876  | 49.82671 |

Figure 37 - Objects Delivered/Second

To elaborate further on hardware speed, data shown in Figure 38 (a) and (b) are compared from an aggregate subscription standpoint and the *ObjectsDelivered* to fulfill the aggregate subscriptions. The number of *ObjectsDelivered* to fill the aggregate subscription is actually a more accurate indicator, as that is how many objects routed through particular solution.



**Figure 38 - Subscriber Count Comparison**

The following evaluation is an attempt to compare facets of two experiments to see how the results compare. Results from section 7.1 (fast subscriber) are compared to the results in the present section.

- The 3x1000 and 6x500 runs were chosen since they both had 3000 subscribers
- The 6x1000 and 3x2000 runs were chosen since they both had 6000 subscribers.

Looking at a similar number of subscriptions from two experiments gives visibility into how accurate a subscription count is at predicting behavior. In the case of the 3000 subscriber load, the number of subscribing objects match ( $6x500 = 3x1000$ ) but the number of *DeliveredObjects* differs by a factor of three ( $6774 = 17520$ ). As can be seen in Figure 38 (a) and (b), both the hardware and software JBI experience an approximate doubling in time, or 105s to 189s and 36s to 63s respectively. In the case of the 6000 subscriber load, the *DeliveredObjects* is also a factor of two differences ( $32256 = 69327$ ) and a similar behavior is observed as the software and hardware double from 461s to 946s and 109s to 269s respectively.

Differing experimental setups would have caused routing object trees of differing complexity. While the total number of subscribing nodes was similar, the number of *ObjectsDelivered* differed considerably. The 6xValue runs were spread more “thinly” than the 3xValue runs – so the subscriber density was less. As such, the publishers were only publishing 500 and 1000 for the 6xValue runs, whereas for the 3xValue runs the publishers published 1000 and 2000 objects

respectively. The larger number of published objects provided a greater possibility that the object would fulfill multiple subscription entries.

Therefore, the conclusion drawn is that the 3xValue runs will be more efficient due to the greater amount of subscriber fulfillment from a given published object (e.g. duplication). This is supported from the object/millisecond times shown above: for the 3xValue runs the times are 10.8ms/object and 13.6ms/object for software and 3.6ms/object and 3.9ms/object for hardware. For the 6xValue runs the data are 15.5ms/object and 14.3ms/object for software and 5.4ms/object and 3.4ms/object for hardware. In the software case, it is easy to see that the 6xValue runs as slower than the 3xValue runs. In the hardware case, the 6x500 run takes longer as expected but the 6x1000 run is actually faster – which is directly related to prior findings that hardware in this particular solution setup performs better under increased load due to the efficiencies available for object duplication and distribution.

## 7.6 Saturation Case

The following experiment showed a saturation case. Figure 39 shows that experimental setup consisted of one subscriber node on either a fast or slow system with four remaining nodes as fast publishers. Publishing nodes were simultaneously publishing the same object type as rapidly as possible and the subscribing node had to consume the published objects as quickly as possible. The speed of publish operations on each computer was controlled by how quickly the subscriber could process the object received and acknowledge to the server that the object had been received. A saturation condition occurs if the subscriber is unable to successfully process the offered load from the publishers. The expected outcome is that both the hardware and software JBI would be able to consume all published objects at the subscriber regardless of the speed of the subscriber, although it is expected that the slow subscriber will take significantly longer than the fast subscriber.

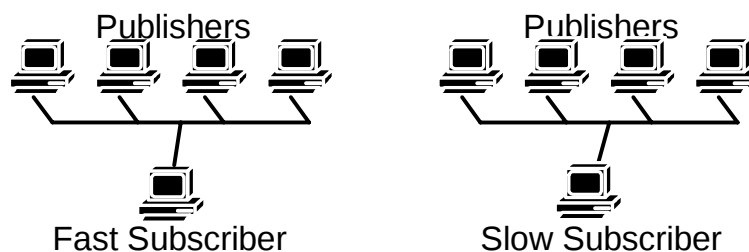


Figure 39 - Saturation Network Setup

### 7.6.1 Software JBI

Software JBI results for both the fast and slow subscriber case are presented below in Figure 40. A saturation condition occurred as neither subscriber (fast/slow) could successfully complete the 4x2000 loading run. Examining the 4x250, 4x500, and 4x1000 runs it is evident that similar behavior is exhibited for both the 4x250 and 4x500 runs for both fast and slow subscribers. Only at the 4x1000 run is there a large divergence of behavior. In the 4x250 and 4x500 cases, the load

is small enough that the subscriber is able to process published objects regardless of subscriber speed. The number of *DeliveredObjects* increases 359% and 387% as loading levels increase to 4x500 and 4x1000 – an approximate equal increase in load. Although the increase in *DeliveredObjects* is approximately equal, there is a much larger increase in processing time of approximately 255% as load increases to 4x500 and 563% (fast)/668% (slow) as load increases to 4x1000 as shown in Table 9. The 4x1000 experiment shows a significant difference in behavior for the slow/fast subscribers as the load increase is similar but the processing time increase is substantial. Examining the charts in Figure 40 and extrapolating, it is easy to see that the 4x2000 run would have taken an inordinately large amount of time had it been able to execute due to the increased size of the 4x2000 run.

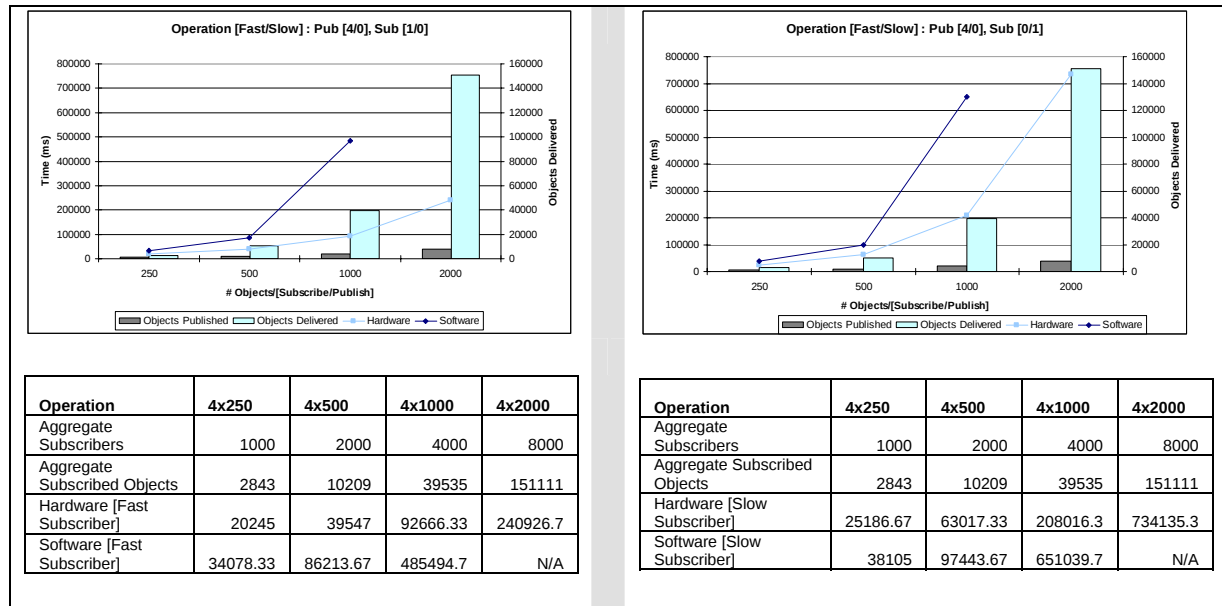


Figure 40 - Saturation Data

| Base 4x250       | 4x500   | 4x1000  | 4x2000  |  | Base 4x250       | 4x500   | 4x1000  | 4x2000  |
|------------------|---------|---------|---------|--|------------------|---------|---------|---------|
| DeliveredObjects | 359.09% | 387.26% | 382.22% |  | DeliveredObjects | 359.09% | 387.26% | 382.22% |
| HardwareFast     | 195.34% | 234.32% | 259.99% |  | HardwareSlow     | 250.20% | 330.09% | 352.92% |
| SoftwareFast     | 252.99% | 563.13% | N/A     |  | SoftwareSlow     | 255.72% | 668.12% | N/A     |

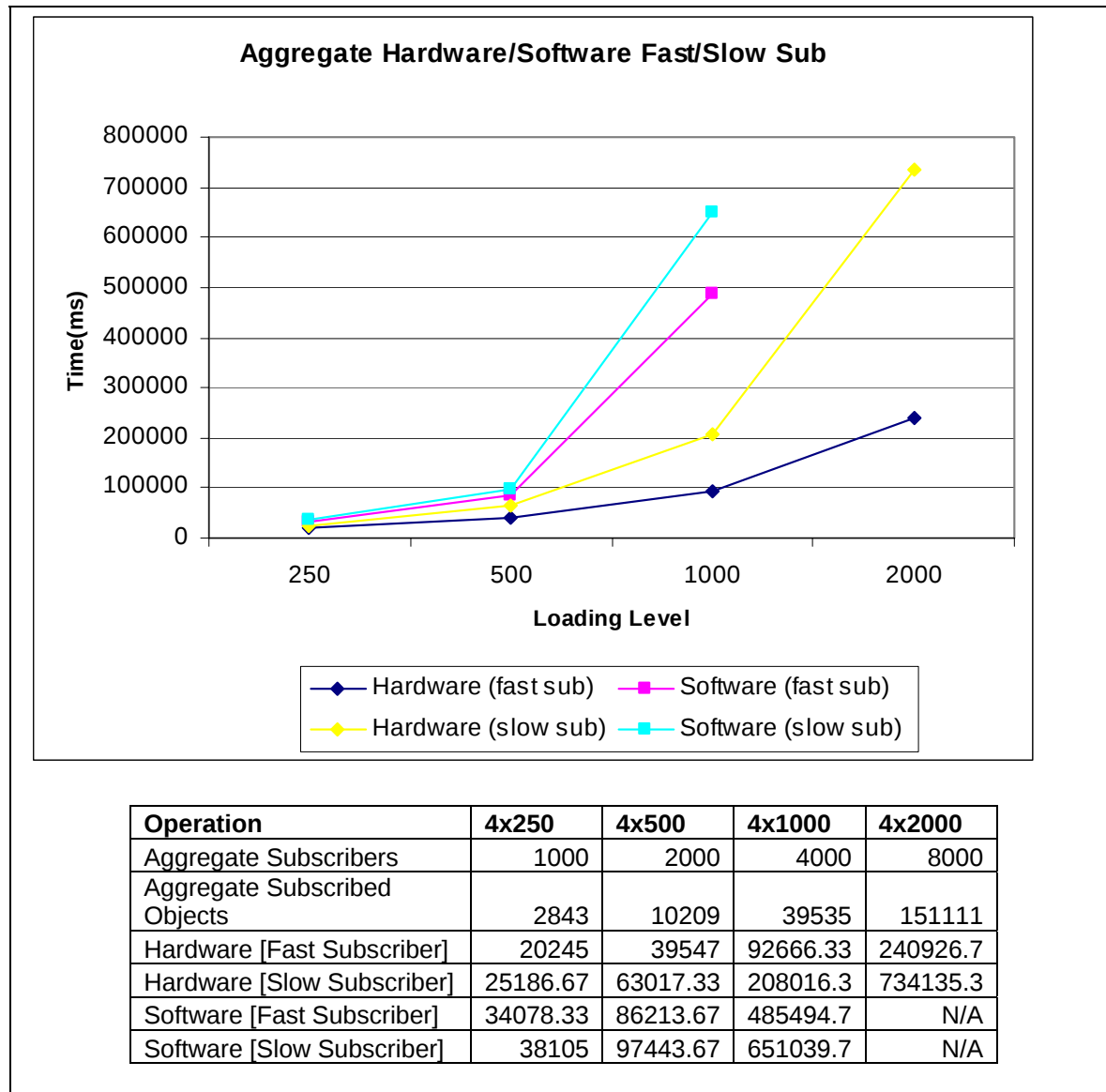
Table 9 - Percent Increase in Object and Processing Speed from Base of 4x250

## 7.6.2 Hardware JBI

Hardware JBI is able to avoid entering a saturation condition for both fast and slow subscribing nodes. Unlike the software JBI case, the hardware JBI shows a progressive increase across all four loading scenarios, although this increase is lower than the software JBI in every instance. As load steadily increases from 4x250, to 4x500, to 4x1000, and to 4x2000 the percentage increase in load is approximately 359-387% across loading levels - or an approximate equal increase in load. Processing time also increases across this loading but at a more gradual pace, increasing 195%(fast) / 250%(slow) from 2x250 to 2x500, increasing 235% (fast) / 330%(slow)

from 2x500 to 2x1000, and finally increasing 260%(fast) / 353%(slow) from 2x1000 to 2x2000. This steady increase in processing time highlights additional latency of processing increased load, but overall processing time is increasing in a gradual manner with increased load.

### 7.6.3 Analysis

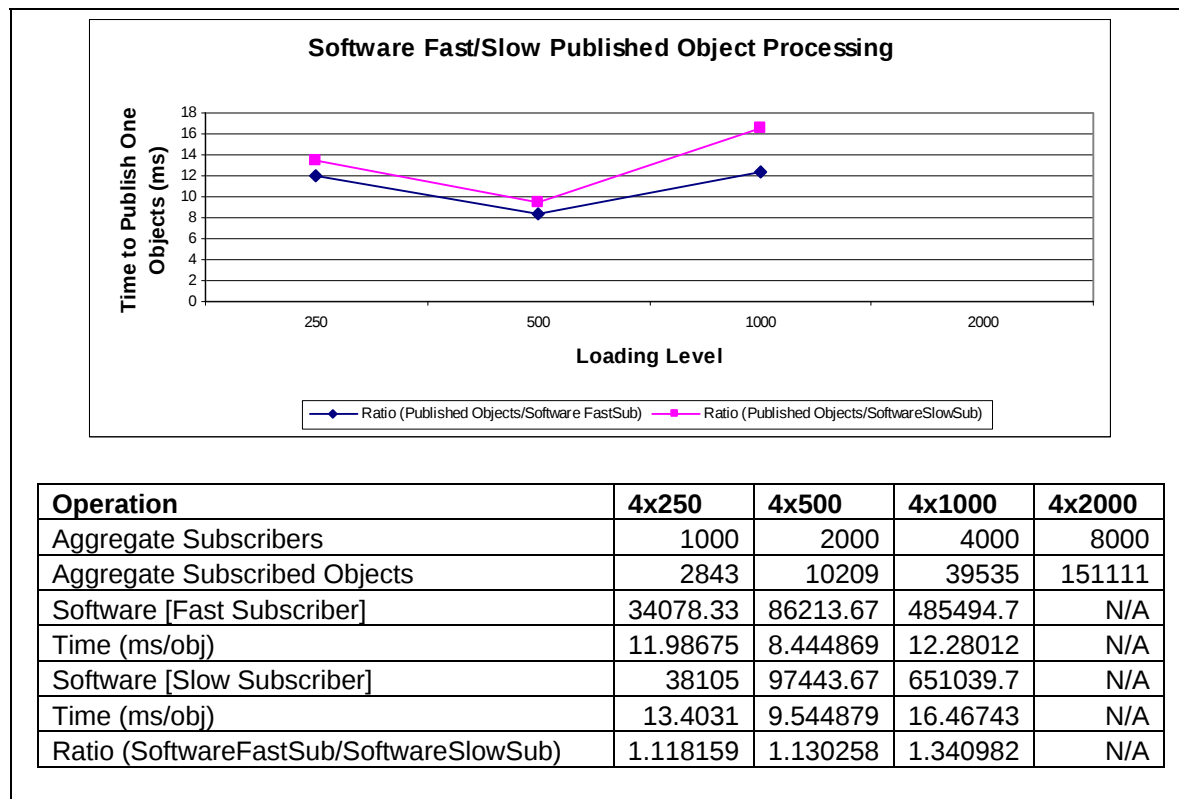


**Figure 41 - Aggregate Hardware/Software Fast/Slow Subscriber**

Figure 41 (above) shows aggregation of the time for hardware/software and fast/slow subscriber. As was expected, as loading levels increase the impact of a slow subscriber node is more pronounced. The 4x250 run show similar behavior for all four cases. In the 4x500 case a divergence starts to appear as the hardware JBI more noticeably separates from the software JBI

and this time is further separated based on the fast/slow subscriber. The 4x1000 shows a pronounced difference between all runs as compared to hardware/fast, ranging from 100% slower for hardware/slow, 500% slower for software/fast, and 650% slower for software/slow. Lastly, the 4x2000 case was only successful on the hardware configuration. The hardware/slow indicates that the hardware JBI was having difficulty maintaining its processing speed at the higher loading levels of this experiment as the slow subscriber was dramatically slowing the process down.

The above results translate into real world metrics for processing time and present a picture of how rapidly a subscribing node can get data distributed to it.



**Figure 42 - Software Fast/Slow Object Processing**

The ratio of the software/fast to software/slow shows that as the load increases the processing time per *PublishedObject* initially decreases then increases as shown in Figure 42. Software/fast consumes 12.0ms for the smallest loading level, this drops to 8.4ms for the 4x500 loading level, then increases to 12.3ms for the highest loading level. The software/slow follows a similar loading level curve as the 4x250 run takes 13.4ms, the 4x500 runs takes 9.5ms, and the 4x1000 takes 16.5ms per *PublishedObject*. The reason for the drop in processing time of the 4x500 run is unclear for both the software/fast and software/slow. An increase in processing time is expected as the software system is under heavier loading as the subscribing levels increase – and each object must traverse the TCP/IP stack and be ingested fully before being processed. The

ratio of *SoftwareFastSub/SoftwareSlowSub* is increasing indicating that the time difference between the slow and fast subscriber is increasing (which is visible in the graph shown in Figure 42). Had the 4x2000 run been successful, a more complete conclusion might have been available.

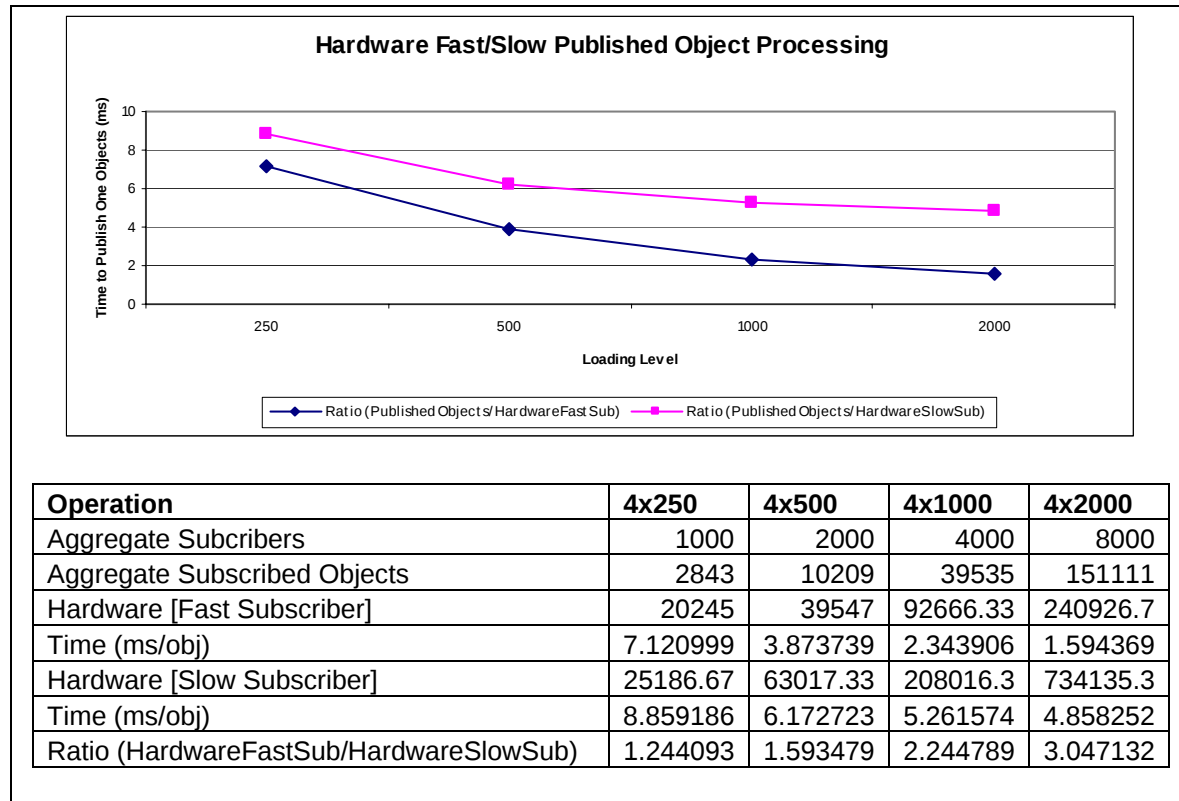
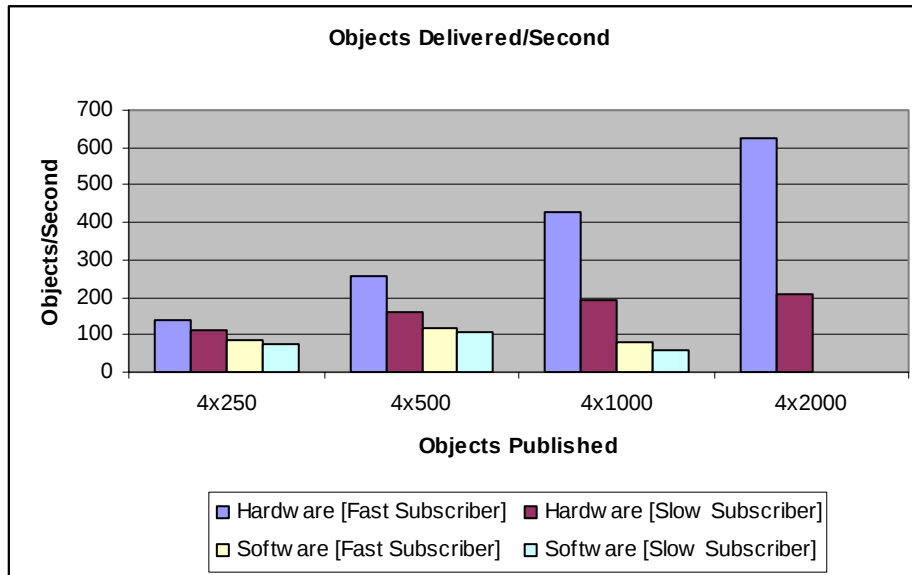


Figure 43 - Hardware Fast/Slow Object Processing

Using Figure 43, and comparing the ratio of the hardware/fast to hardware/slow we see that as load increases, processing time per *PublishedObject* decreases from 7.12ms per subscriber fulfillment object then to 1.6ms per subscriber object. A similar decrease is present on the hardware/slow as it takes 8.9ms per *PublishedObject* initially to 4.9ms at the end. Hardware realizes efficiencies under higher loading as packet duplication and transmission are accomplished internally at close to line speed. The hardware/slow is expected to be higher as the additional latency caused by the slow subscriber will cause the systems routing speed to decrease as the system awaits the acknowledgement from the slow subscriber. Lastly, the ratio of *HardwareFastSub/HardwareSlowSub* shows a dramatic increase from 1.24, 1.59, 2.24, and 3.04 – this value indicates that as the loading level increases the slow subscriber causes more delay and the difference in processing time between a slow subscriber and a fast subscriber grows considerably (3.04x difference at 4x2000).

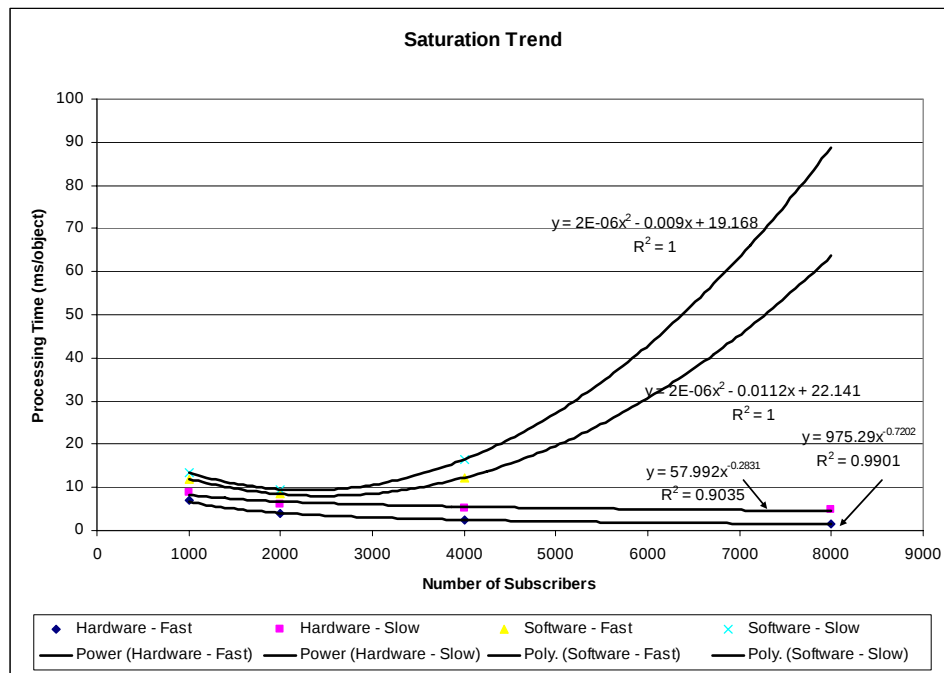
From an objects delivered/second perspective, Figure 44 highlights that hardware provides a much higher throughput than software. Hardware with a fast subscriber is within the increasing

range of 140.4obj/s and 627.2obj/s whereas hardware with a slow subscriber is within the range of 112.9obj/s and 205.8obj/s – clearly the slow subscriber affects the speed at which the JBI can fulfill publication requests. Software with a fast subscriber is within a smaller range of 83.4obj/s and 118.4obj/s, and software with a slow subscriber ranges across 60.7obj/s and 104.8obj/s. In the software case, the 4x1000 run indicates that the obj/s processing speed is trending downward rapidly. The results for the 4x1000 run were not available due to the JBI becoming saturated and crashing during the experiment execution.



| Operation                    | 4x250    | 4x500    | 4x1000   | 4x2000   |
|------------------------------|----------|----------|----------|----------|
| Aggregate Subscribers        | 1000     | 2000     | 4000     | 8000     |
| Aggregate Subscribed Objects | 2843     | 10209    | 39535    | 151111   |
| Hardware [Fast Subscriber]   | 140.4297 | 258.1485 | 426.6382 | 627.2074 |
| Hardware [Slow Subscriber]   | 112.8772 | 162.003  | 190.0572 | 205.8353 |
| Software [Fast Subscriber]   | 83.42544 | 118.4151 | 81.43241 | N/A      |
| Software [Slow Subscriber]   | 74.60963 | 104.7682 | 60.72595 | N/A      |

Figure 44 - Objects Delivered/Second



**Figure 45 - Saturation Trend**

Figure 45 presents processing time trends associated with hardware and software JBI systems. As the trend lines indicate, the trend associated with the software JBI is polynomial increasing as subscription load increases – leading to greater processing time per object. The trend associated with the hardware JBI is decreasing according to a power regression line indicating that hardware becomes more efficient with increased subscription load (which has a higher level of duplicated objects).

## 8 Follow-on Work

There are a number of areas that have yet to be explored looking at hardware acceleration for the JBI system.

- **Accurate Network Topology:** Include multiple subnets, routers, firewalls, and XML routers with a variety of publishing systems. Base client applications on what is expected to be fielded and determine how the network topology influences the speed of the JBI system. Furthermore, examine cascading of XML routers for complicated network scenarios.
- **Traffic Generation:** insert a traffic generator into the network to increase load. This would more accurately resemble what a fielded JBI system would have to contend with in its normal operational environment
- **Deeper XPATH/Metadata trees:** Increase complexity and depth of XML messages. Increased XML adds to complexity and allows for a more accurate depiction of traffic available on the network. It is probably reasonable to assume that a sample published

object (intel, weather, etc.) would contain multiple XML levels, multiple fields, and verbose descriptions of content.

- Payload sizes: Randomize payload sizes, number of embedded objects, and types. Increase realism of data objects to more accurately reflect real-world type data.
- Encryption/Signing/Hashing: Include these real-world operations in the experiment as these are processor and time consuming operations. A fielded JBI would need to support these types of operations.
- Control duplication: By duplication I am referring to subscriber fulfillment. I opted to allow for a single published object to fulfill as many subscriptions as possible in order to increase load. The side effect of this action was that the hardware JBI could do this very efficiently. The data that was not gathered relates to “what if” all publications/subscriptions are 1:1 or maybe only have 5% duplication. Adding in a setting for duplication is critical to determining a more accurate picture of publish/subscribe operations.
- Full Hardware JBI: Implement the hardware JBI using only the steps required of the CAPI. For comparison purposes only minimal changes were made to the CAPI to incorporate hardware so many wasteful operations were executed on the hardware path. Remove all extraneous software processing from the hardware path to get the best hardware JBI implementation.
- Incorporate a true Hardware JBI router: The router provided by Sarvega was a commercial product that they modified the software to support the JBI experiment. A true hardware router would be optimized in both hardware and software and should provide the most efficient JBI implementation for execution.

## 9 Conclusion

The integration of a hardware based eXtensible Markup Language (XML) processor into the JBI architecture was a success. The integration was invisible to the user and worked seamlessly with no changes to JBI client software and only minor changes to the JBI provided core framework. Results gathered reflect the simplest change to the JBI framework operation in that the final step of the publish/subscribe operation was done differently using hardware – this was to make comparisons between both solutions as simple and fair as possible. Had all of the extraneous RMI/JMS operations been removed from the Hardware JBI results for the Hardware JBI would have been even better – as the present solution allows approximately 200 packets to traverse the network for each publish/subscribe to support RMI/JMS (which aren’t used by the hardware JBI). Additionally, Base64 encoding added approximately 33% overhead to all hardware JBI packets that was not present with the software JBI implementation.

The experiment outcomes were not surprising. The software JBI is facing a number of hurdles not present with the hardware JBI, most notably that the software JBI is not running on a dedicated, XML aware, network router. The ability of the hardware JBI to leverage the unique capabilities provided by fact that it is a network appliance are crucial to its ability to rapidly outperform the software JBI. The hardware JBI could process at line speed, as it is an in-line network router and could parse the Layer 7 XML data without traversing a TCP/IP stack.

Additionally, the hardware JBI could natively understand XML and XPATH expression logic to perform logical operations on the data and could rapidly route published data to subscribers using a state machine mechanism that provided efficient packet duplication for multiple subscriber fulfillments.

Dedicated XML aware hardware is crucial to the successful fielding of a JBI system as that is the only hope of processing the sheer volume of information expected to be present. XML aware, Layer 7 routing of the payload at line speeds is a unique operation that is only possible with a hardware type device – which additionally can leverage custom designed hardware components to more efficiently process the tree structures presented by XML. Insert encryption, document signing and hashing, secure connection setup and the software JBI falls further and further behind since it does not contain dedicated hardware to perform these CPU intensive operations – the hardware device contained hardware encryption capability although this was not explored in this experiment – but hardware encryption far surpasses any software based encryption scheme.

At low loading levels, the two implementations behaved similarly as was expected. Only at higher loading levels did the differences in implementation become apparent – most notably the time required to handle the offered subscription load. Payload size appeared to have a greater impact on both solutions than increased subscriber load. Payload size is going to be relatively arbitrary and spread across a wide range of values – ranging from a small text based message to a large image or video with a huge XML metadata portion. The JBI system can more efficiently distribute a smaller message to more subscribers and this should be taken into account for subsequent JBI development.

The hardware JBI presented the counter-intuitive result of becoming more efficient under greater loading. This result was due to the ability of the hardware to rapidly disseminate a processed publication request at minimal cost, so under higher loading where a single published object will fulfill a larger number of subscriptions, the hardware paid an upfront penalty then could cheaply replicate and send it multiple times. The software JBI had to perform a software duplication of each packet then traverse the TCP/IP stack to distribute the fulfillment to the subscribing nodes.

The data distribution abilities of both systems were captured by the objects delivered per second metric. The hardware had a much wider range of values than the software and generally performed better with increased subscriber loading – leveraging its inherent ability to rapidly route published data to multiple subscribers. In the case of software, the range of behavior was much more restricted and never exceeded the hardware speed for a given experiment. Across all loading levels and payload sizes, hardware ranged from 14.2obj/s to 627.2obj/s and software ranged from 16.4obj/s to 118.4obj/s. The lower bounds were similar, but the upper bounds show an over 5x increase in objects per second processed for hardware.

In conclusion, the hardware JBI system was faster than the software JBI for all experiments performed. The range of improvement was dependent on the configuration of the experiment, but improvements ranged from 340% to 750% faster at the x2000 loading levels. Therefore, the

JBI of the future will have to have XML hardware aware routers at its foundation in order to provide the best performance. Tangible performance and scalability benefits available from hardware will far exceed the larger upfront cost of a dedicated hardware router.

## 10 Appendix A – Raw Data

### 10.1 Experiment 1

#### 10.1.1 Hardware JBI – Object Size 2KB

| Computer | Action | ObjectType | Seed | Object | ObjRecv | TotalTime | ObjRecv | TotalTime | ObjRecv | TotalTime | AvgTime  |          |
|----------|--------|------------|------|--------|---------|-----------|---------|-----------|---------|-----------|----------|----------|
| #1       | pub    | .basic     | 1141 | 250    |         |           |         |           |         |           |          |          |
| #2       | sub    | .basic     | 141  | 250    | 451     | 16640     | 451     | 11890     | 451     | 11485     | 13338.33 | 13297.44 |
| #3       | sub    | .basic     | 142  | 250    | 476     | 16625     | 476     | 11875     | 476     | 11484     | 13328    |          |
| #4       | sub    | .basic     | 143  | 250    | 385     | 16504     | 385     | 11797     | 385     | 11377     | 13226    |          |
| #1       | pub    | .basic     | 1141 | 500    |         |           |         |           |         |           |          |          |
| #2       | sub    | .basic     | 141  | 500    | 1519    | 30797     | 1519    | 30750     | 1519    | 29656     | 30401    | 30404.22 |
| #3       | sub    | .basic     | 142  | 500    | 1672    | 30797     | 1672    | 30765     | 1672    | 29672     | 30411.33 |          |
| #4       | sub    | .basic     | 143  | 500    | 1549    | 30794     | 1549    | 30754     | 1549    | 29653     | 30400.33 |          |
| #1       | pub    | .basic     | 1141 | 1000   |         |           |         |           |         |           |          |          |
| #2       | sub    | .basic     | 141  | 1000   | 5492    | 88515     | 5492    | 82625     | 5492    | 80266     | 83802    | 83465.56 |
| #3       | sub    | .basic     | 142  | 1000   | 6296    | 88500     | 6296    | 79688     | 6296    | 80265     | 82817.67 |          |
| #4       | sub    | .basic     | 143  | 1000   | 5732    | 88487     | 5732    | 82608     | 5732    | 80236     | 83777    |          |
| #1       | pub    | .basic     | 1141 | 2000   |         |           |         |           |         |           |          |          |
| #2       | sub    | .basic     | 141  | 2000   | 22860   | 287282    | 22860   | 288109    | 22860   | 287906    | 287765.7 | 287755   |
| #3       | sub    | .basic     | 142  | 2000   | 22765   | 287281    | 22765   | 288094    | 22765   | 287891    | 287755.3 |          |
| #4       | sub    | .basic     | 143  | 2000   | 23702   | 287263    | 23702   | 288085    | 23702   | 287884    | 287744   |          |

#### 10.1.2 Software JBI – Object Size 2KB

| Computer | Action | ObjectType | Seed | Object | ObjRecv | TotalTime | ObjRecv | TotalTime | ObjRecv | TotalTime | AvgTime  |          |
|----------|--------|------------|------|--------|---------|-----------|---------|-----------|---------|-----------|----------|----------|
| #1       | pub    | .basic     | 1141 | 250    |         |           |         |           |         |           |          |          |
| #2       | sub    | .basic     | 141  | 250    | 451     | 17016     | 451     | 21953     | 451     | 16672     | 18547    | 18266    |
| #3       | sub    | .basic     | 142  | 250    | 476     | 17031     | 476     | 21953     | 476     | 16672     | 18552    |          |
| #4       | sub    | .basic     | 143  | 250    | 385     | 16865     | 385     | 19788     | 385     | 16444     | 17699    |          |
| #1       | pub    | .basic     | 1141 | 500    |         |           |         |           |         |           |          |          |
| #2       | sub    | .basic     | 141  | 500    | 1519    | 53547     | 1519    | 58922     | 1519    | 60922     | 57797    | 57782.11 |
| #3       | sub    | .basic     | 142  | 500    | 1672    | 53578     | 1672    | 58969     | 1672    | 60922     | 57823    |          |
| #4       | sub    | .basic     | 143  | 500    | 1549    | 53507     | 1549    | 58845     | 1549    | 60827     | 57726.33 |          |
| #1       | pub    | .basic     | 1141 | 1000   |         |           |         |           |         |           |          |          |
| #2       | sub    | .basic     | 141  | 1000   | 5492    | 198969    | 5492    | 212469    | 5492    | 221219    | 210885.7 | 210887   |
| #3       | sub    | .basic     | 142  | 1000   | 6296    | 198984    | 6296    | 212578    | 6296    | 221266    | 210942.7 |          |
| #4       | sub    | .basic     | 143  | 1000   | 5732    | 198865    | 5732    | 212475    | 5732    | 221158    | 210832.7 |          |
| #1       | pub    | .basic     | 1141 | 2000   |         |           |         |           |         |           |          |          |
| #2       | sub    | .basic     | 141  | 2000   | 22860   | 992672    | 22860   | 896766    | 22860   | 1041969   | 977135.7 | 977102   |
| #3       | sub    | .basic     | 142  | 2000   | 23702   | 992610    | 22765   | 896781    | 22765   | 1041969   | 977120   |          |
| #4       | sub    | .basic     | 143  | 2000   | 22765   | 992703    | 23702   | 896570    | 23702   | 1041878   | 977050.3 |          |

## 10.2 Experiment 2

### 10.2.1 Hardware JBI – Object Size 2KB

| Computer | Action | ObjectType | Seed | Object |       | ObjRecv | TotalTime | ObjRecv | TotalTime | ObjRecv | TotalTime | AvgTime  |          |
|----------|--------|------------|------|--------|-------|---------|-----------|---------|-----------|---------|-----------|----------|----------|
| #2       | pub    | .basic     | 1141 | 1000   |       |         |           |         |           |         |           |          |          |
| #3       | sub    | .basic     | 142  | 1000   | 11788 | 6296    | 53406     | 6296    | 51313     | 6296    | 52203     | 52307.33 | 52307.5  |
| #5       | sub    | .basic     | 141  | 1000   |       | 5492    | 53407     | 5492    | 51313     | 5492    | 52203     | 52307.67 |          |
| #1       | sub    | .basic     | 141  | 250    | 1312  | 451     | 10359     | 451     | 9953      | 451     | 9688      | 10000    | 9972.222 |
| #2       | pub    | .basic     | 1141 | 250    |       |         |           |         |           |         |           |          |          |
| #3       | sub    | .basic     | 143  | 250    |       | 385     | 10265     | 385     | 9875      | 385     | 9610      | 9916.667 |          |
| #5       | sub    | .basic     | 142  | 250    |       | 476     | 10375     | 476     | 9953      | 476     | 9672      | 10000    |          |
| #1       | sub    | .basic     | 141  | 500    | 4740  | 1519    | 23078     | 1519    | 24000     | 1519    | 23078     | 23385.33 | 23383.89 |
| #2       | pub    | .basic     | 1141 | 500    |       |         |           |         |           |         |           |          |          |
| #3       | sub    | .basic     | 143  | 500    |       | 1549    | 23079     | 1549    | 24000     | 1549    | 23078     | 23385.67 |          |
| #5       | sub    | .basic     | 142  | 500    |       | 1672    | 23079     | 1672    | 24000     | 1672    | 23063     | 23380.67 |          |
| #1       | sub    | .basic     | 141  | 1000   | 17520 | 5492    | 64093     | 5492    | 63938     | 5492    | 61828     | 63286.33 | 63283    |
| #2       | pub    | .basic     | 1141 | 1000   |       |         |           |         |           |         |           |          |          |
| #3       | sub    | .basic     | 143  | 1000   |       | 5732    | 64094     | 5732    | 63922     | 5732    | 61828     | 63281.33 |          |
| #5       | sub    | .basic     | 142  | 1000   |       | 6296    | 64078     | 6296    | 63938     | 6296    | 61828     | 63281.33 |          |
| #1       | sub    | .basic     | 141  | 2000   | 69327 | 22860   | 264125    | 22860   | 280297    | 22860   | 263328    | 269250   | 269248.4 |
| #2       | pub    | .basic     | 1141 | 2000   |       |         |           |         |           |         |           |          |          |
| #3       | sub    | .basic     | 143  | 2000   |       | 23702   | 264110    | 23702   | 280312    | 23702   | 263313    | 269245   |          |
| #5       | sub    | .basic     | 142  | 2000   |       | 22765   | 264094    | 22765   | 280328    | 22765   | 263329    | 269250.3 |          |
| #1       | sub    | .basic     | 141  | 1000   | 23487 | 5492    | 73109     | 5492    | 73078     | 5492    | 72360     | 72849    | 72858.08 |
| #2       | sub    | .basic     | 144  | 1000   |       | 5967    | 73141     | 5967    | 73078     | 5967    | 72375     | 72864.67 |          |
| #3       | pub    | .basic     | 1141 | 1000   |       |         |           |         |           |         |           |          |          |
| #5       | sub    | .basic     | 143  | 1000   |       | 5732    | 73125     | 5732    | 73094     | 5732    | 72375     | 72864.67 |          |
| #6       | sub    | .basic     | 142  | 1000   |       | 6296    | 73141     | 6296    | 73062     | 6296    | 72359     | 72854    |          |

### 10.2.2 Software JBI – Object Size 2KB

| Computer | Action | ObjectType | Seed | Object |       | ObjRecv | TotalTime | ObjRecv | TotalTime | ObjRecv | TotalTime | AvgTime  |          |
|----------|--------|------------|------|--------|-------|---------|-----------|---------|-----------|---------|-----------|----------|----------|
| #2       | pub    | .basic     | 1141 | 1000   |       |         |           |         |           |         |           |          |          |
| #3       | sub    | .basic     | 142  | 1000   | 11788 | 6296    | 134265    | 6296    | 130576    | 6296    | 129344    | 131395   | 131377.2 |
| #5       | sub    | .basic     | 141  | 1000   |       | 5492    | 134250    | 5492    | 130500    | 5492    | 129328    | 131359.3 |          |
| #1       | sub    | .basic     | 141  | 250    | 1312  | 451     | 16765     | 451     | 16735     | 451     | 16610     | 16703.33 | 16670.11 |
| #2       | pub    | .basic     | 1141 | 250    |       |         |           |         |           |         |           |          |          |
| #3       | sub    | .basic     | 143  | 250    |       | 385     | 16687     | 385     | 16640     | 385     | 16515     | 16614    |          |
| #5       | sub    | .basic     | 142  | 250    |       | 476     | 16766     | 476     | 16719     | 476     | 16594     | 16693    |          |
| #1       | sub    | .basic     | 141  | 500    | 4740  | 1519    | 51516     | 1519    | 54862     | 1519    | 51609     | 52662.33 | 52595.56 |
| #2       | pub    | .basic     | 1141 | 500    |       |         |           |         |           |         |           |          |          |
| #3       | sub    | .basic     | 143  | 500    |       | 1549    | 51531     | 1549    | 54563     | 1549    | 51609     | 52567.67 |          |
| #5       | sub    | .basic     | 142  | 500    |       | 1672    | 51515     | 1672    | 54562     | 1672    | 51593     | 52556.67 |          |
| #1       | sub    | .basic     | 141  | 1000   | 17520 | 5492    | 189078    | 5492    | 188938    | 5492    | 189703    | 189239.7 | 189274.3 |
| #2       | pub    | .basic     | 1141 | 1000   |       |         |           |         |           |         |           |          |          |
| #3       | sub    | .basic     | 143  | 1000   |       | 5732    | 189109    | 5732    | 189000    | 5732    | 189750    | 189286.3 |          |
| #5       | sub    | .basic     | 142  | 1000   |       | 6296    | 189172    | 6296    | 189000    | 6296    | 189719    | 189297   |          |
| #1       | sub    | .basic     | 141  | 2000   | 69327 | 22860   | 941000    | 22860   | 877782    | 22860   | 1005406   | 941396   | 945826.9 |
| #2       | pub    | .basic     | 1141 | 2000   |       |         |           |         |           |         |           |          |          |
| #3       | sub    | .basic     | 143  | 2000   |       | 22765   | 941031    | 23702   | 897812    | 23702   | 1005407   | 948083.3 |          |
| #5       | sub    | .basic     | 142  | 2000   |       | 23702   | 940813    | 22765   | 897797    | 22765   | 1005394   | 948001.3 |          |
| #1       | sub    | .basic     | 141  | 1000   | 23487 | 5492    | 269623    | 5492    | 267859    | 5492    | 269422    | 268968   | 268981.6 |
| #2       | sub    | .basic     | 144  | 1000   |       | 5967    | 269703    | 5967    | 267859    | 5967    | 269437    | 268999.7 |          |
| #3       | pub    | .basic     | 1141 | 1000   |       |         |           |         |           |         |           |          |          |
| #5       | sub    | .basic     | 143  | 1000   |       | 5732    | 269609    | 5732    | 267828    | 5732    | 269438    | 268958.3 |          |
| #6       | sub    | .basic     | 142  | 1000   |       | 6296    | 269688    | 6296    | 267860    | 6296    | 269453    | 269000.3 |          |

## 10.3 Experiment 3

### 10.3.1 Hardware JBI – 25KB and 50KB Object Size

25KB Object Size

50KB Object Size

| Computer | Action | ObjectType | Seed | Object | ObjRecv | Time   | ObjRecv | Time   | AvgTime  | ObjRecv  | Time  | ObjRecv | Time    | AvgTime  |          |
|----------|--------|------------|------|--------|---------|--------|---------|--------|----------|----------|-------|---------|---------|----------|----------|
| #2       | pub    | .basic     | 1141 | 1000   |         |        |         |        |          |          |       |         |         |          |          |
| #3       | sub    | .basic     | 142  | 1000   | 6296    | 224297 | 6296    | 256785 | 238010   | 238033.7 | 6296  | 453641  | 423672  | 431911.7 | 431924.5 |
| #5       | sub    | .basic     | 141  | 1000   | 5492    | 224329 | 5492    | 256812 | 238057.3 |          | 5492  | 453656  | 423703  | 431937.3 |          |
| #1       | sub    | .basic     | 141  | 250    |         |        |         |        |          |          |       |         |         |          |          |
| #2       | sub    | .basic     | 141  | 250    | 451     | 62125  | 451     | 57563  | 79010.33 | 78727.22 | 451   | 90828   | 92438   | 92437.67 | 92118    |
| #3       | sub    | .basic     | 141  | 250    |         |        |         |        |          |          |       |         |         |          |          |
| #5       | sub    | .basic     | 142  | 250    | 385     | 61735  | 385     | 57110  | 78270.67 |          | 385   | 89969   | 91562   | 91567.67 |          |
|          |        |            |      |        | 476     | 62109  | 476     | 57500  | 78900.67 |          | 476   | 90796   | 92234   | 92348.67 |          |
| #1       | sub    | .basic     | 141  | 500    |         |        |         |        |          |          |       |         |         |          |          |
| #2       | sub    | .basic     | 1141 | 500    | 1519    | 125297 | 1519    | 125500 | 121260.3 | 120918.4 | 1519  | 225562  | 216407  | 215052.3 | 216154.7 |
| #3       | sub    | .basic     | 143  | 500    | 1549    | 125328 | 1549    | 125500 | 121270.7 |          | 1549  | 225641  | 216406  | 215088.7 |          |
| #5       | sub    | .basic     | 142  | 500    | 1672    | 125266 | 1672    | 122469 | 120224.3 |          | 1672  | 225547  | 216344  | 213078   | 218323   |
| #1       | sub    | .basic     | 141  | 1000   |         |        |         |        |          |          |       |         |         |          |          |
| #2       | sub    | .basic     | 1141 | 1000   | 5492    | 261281 | 5492    | 253172 | 256161.3 | 256126.8 | 5492  | 484547  | 498438  | 486333.7 | 486322.9 |
| #3       | sub    | .basic     | 143  | 1000   | 5732    | 261265 | 5732    | 253110 | 256130.3 |          | 5732  | 484562  | 498406  | 486333   |          |
| #5       | sub    | .basic     | 142  | 1000   | 6296    | 261219 | 6296    | 253078 | 256088.7 |          | 6296  | 484516  | 498422  | 486302   |          |
| #1       | sub    | .basic     | 141  | 2000   |         |        |         |        |          |          |       |         |         |          |          |
| #2       | sub    | .basic     | 1141 | 2000   | 22860   | 653735 | 22860   | 656437 | 657177.3 | 657121.7 | 22860 | 1282843 | 1278265 | 1262921  | 1262904  |
| #3       | sub    | .basic     | 143  | 2000   | 23702   | 653672 | 23702   | 656407 | 657120   |          | 23702 | 1282765 | 1278172 | 1262859  |          |
| #5       | sub    | .basic     | 142  | 2000   | 22765   | 653390 | 22765   | 656453 | 657067.7 |          | 22765 | 1282812 | 1278282 | 1262932  |          |
| #1       | sub    | .basic     | 141  | 1000   |         |        |         |        |          |          |       |         |         |          |          |
| #2       | sub    | .basic     | 1141 | 1000   | 5492    | 253985 | 5492    | 251969 | 258364.7 | 258350.4 | 5492  | 493625  | 480109  | 490526   | 490261.6 |
| #3       | sub    | .basic     | 144  | 1000   | 5967    | 253985 | 5967    | 251985 | 258370   |          | 5967  | 493687  | 480094  | 490526   |          |
| #5       | sub    | .basic     | 143  | 1000   | 5732    | 253969 | 5732    | 251969 | 258359.7 |          | 5732  | 493656  | 480094  | 490531.3 |          |
| #6       | sub    | .basic     | 142  | 1000   | 6296    | 253907 | 6296    | 251906 | 258307.3 |          | 6296  | 493546  | 477015  | 497828   | 489463   |

### 10.3.2

### Software JBI – 25KB and 50KB Object Size

#### 25KB Object Size

#### 50KB Object Size

| Computer | Action | ObjectType | Seed | Object | ObjRecv | Time    | ObjRecv | Time    | AvgTime  | ObjRecv | Time    | ObjRecv | Time    | AvgTime |          |
|----------|--------|------------|------|--------|---------|---------|---------|---------|----------|---------|---------|---------|---------|---------|----------|
| #2       | pub    | basic      | 1141 | 1000   |         |         |         |         |          |         |         |         |         |         |          |
| #3       | sub    | basic      | 142  | 1000   | 6296    | 423545  | 6296    | 435359  | 432108.7 | 6296    | 681359  | 6296    | 700969  | 696000  | 692692.5 |
| #5       | sub    | basic      | 141  | 1000   | 5492    | 423390  | 5492    | 435359  | 432010   | 5492    | 681140  | 5492    | 700750  | 695937  | 692609   |
| #1       | sub    | basic      | 141  | 250    |         |         |         |         |          |         |         |         |         |         |          |
| #2       | pub    | basic      | 1141 | 250    | 451     | 54500   | 451     | 55078   | 53406.33 | 451     | 80297   | 451     | 79484   | 80218   | 79999.67 |
| #3       | sub    | basic      | 143  | 250    | 385     | 51968   | 385     | 54954   | 52463.33 | 385     | 80000   | 385     | 79140   | 79860   | 79666.67 |
| #5       | sub    | basic      | 142  | 250    | 476     | 54438   | 476     | 55109   | 53401    | 476     | 80359   | 476     | 79594   | 80110   | 80021    |
| #1       | sub    | basic      | 141  | 500    |         |         |         |         |          |         |         |         |         |         |          |
| #2       | pub    | basic      | 1141 | 500    | 1519    | 169797  | 1519    | 171672  | 171317.7 | 1519    | 269656  | 1519    | 276760  | 267062  | 271159.3 |
| #3       | sub    | basic      | 143  | 500    | 1549    | 168594  | 1549    | 171785  | 171302.3 | 1549    | 269719  | 1549    | 276844  | 267031  | 271198   |
| #5       | sub    | basic      | 142  | 500    | 1672    | 169765  | 1672    | 171766  | 171380   | 1672    | 269797  | 1672    | 276844  | 267079  | 271240   |
| #1       | sub    | basic      | 141  | 1000   | 5492    | 656063  | 5492    | 646266  | 648526.3 | 5492    | 1018531 | 5492    | 1016516 | 1012375 | 1015807  |
| #2       | pub    | basic      | 1141 | 1000   |         |         |         |         |          |         |         |         |         |         |          |
| #3       | sub    | basic      | 143  | 1000   | 5732    | 656157  | 5732    | 646141  | 648615   | 5732    | 1018875 | 5732    | 1016594 | 1012250 | 1015906  |
| #5       | sub    | basic      | 142  | 1000   | 6296    | 656219  | 6296    | 646234  | 648640.3 | 6296    | 1019015 | 6296    | 1016937 | 1013000 | 1016317  |
| #1       | sub    | basic      | 141  | 2000   | 22860   | 3254390 | 22860   | 3260141 | 3221844  | 22860   | 5609266 | 22860   | 5235297 | 5338532 | 5394467  |
| #2       | pub    | basic      | 1141 | 2000   |         |         |         |         |          |         |         |         |         |         |          |
| #3       | sub    | basic      | 143  | 2000   | 23702   | 3254562 | 23702   | 3260109 | 3221849  | 23702   | 5609281 | 23702   | 5235296 | 5338454 | 5394344  |
| #5       | sub    | basic      | 142  | 2000   | 22765   | 3254641 | 22765   | 3260282 | 3222053  | 22765   | 5609719 | 22765   | 5235625 | 5338734 | 5394693  |
| #1       | sub    | basic      | 141  | 1000   | 5492    | 957781  | 5492    | 879078  | 925323   | 5492    | 1638375 | 5492    | 1524437 | 1531328 | 1564901  |
| #2       | pub    | basic      | 1141 | 1000   |         |         |         |         |          |         |         |         |         |         |          |
| #3       | sub    | basic      | 144  | 1000   | 5967    | 957516  | 5967    | 879360  | 925411.7 | 5967    | 1638421 | 5967    | 1524500 | 1532109 | 1565010  |
| #5       | sub    | basic      | 143  | 1000   | 5732    | 957812  | 5732    | 879250  | 925505   | 5732    | 1637985 | 5732    | 1524453 | 1532062 | 1564833  |
| #6       | sub    | basic      | 142  | 1000   | 6296    | 957813  | 6296    | 879360  | 925552.3 | 6296    | 1638469 | 6296    | 1524563 | 1532109 | 1565047  |

#### 10.4.1 Hardware JBI – Object Size 2KB

62

## 10.4.2 Software JBI – Object Size 2KB

| Computer | Action | ObjectType           | Seed    | Object | ObjRecv                                                         | Time | ObjRecv                                  | Time | ObjRecv                                  | Time | ObjRecv       | Time     | AvgTime  |  |
|----------|--------|----------------------|---------|--------|-----------------------------------------------------------------|------|------------------------------------------|------|------------------------------------------|------|---------------|----------|----------|--|
| #1       | pub    | basic                | 1151    | 250    |                                                                 |      |                                          |      |                                          |      |               |          |          |  |
| #2       | pub    | .xmlxpath            | 1152    | 250    |                                                                 |      |                                          |      |                                          |      |               |          |          |  |
| #3       | pub    | .ato                 | 1153    | 250    | ObjRecv                                                         | Time | ObjRecv                                  | Time | ObjRecv                                  | Time | ObjRecv       | Time     |          |  |
| #5       | sub    | basic, xmlxpath, ato | 151-153 | 250    | 235 36906 313 35625 35865 660 36141 44 33828                    |      | 235 33844 313 22535                      |      | 235 34562 660 33593 44 32625             |      | 313 33593     | 30584.33 | 32546.61 |  |
| #6       | sub    | basic, xmlxpath, ato | 154-156 | 250    | 261 37000 267 36104                                             |      | 261 34032 267 33464                      |      | 261 34672 440 33656 100 33547            |      | 267 33958     | 34508.89 |          |  |
| #1       | pub    | basic                | 1151    | 500    |                                                                 |      |                                          |      |                                          |      |               |          |          |  |
| #2       | pub    | .xmlxpath            | 1152    | 500    |                                                                 |      |                                          |      |                                          |      |               |          |          |  |
| #3       | pub    | .ato                 | 1153    | 500    | ObjRecv                                                         | Time | ObjRecv                                  | Time | ObjRecv                                  | Time | ObjRecv       | Time     |          |  |
| #5       | sub    | basic, xmlxpath, ato | 151-153 | 500    | 1001 116453 1187 111713 111724 1001 108094 1187 102609          |      | 1001 107703 2180 98203 379 95922         |      | 1001 107703 2180 98203 379 95922         |      | 1187 100609   | 104977.2 | 104980   |  |
| #6       | sub    | basic, xmlxpath, ato | 154-156 | 500    | 1297 116531 1071 111734                                         |      | 1297 108110 1071 102610                  |      | 1297 107828 1638 98188 279 95797         |      | 1071 100604   | 104982.8 |          |  |
| #1       | pub    | basic                | 1151    | 1000   |                                                                 |      |                                          |      |                                          |      |               |          |          |  |
| #2       | pub    | .xmlxpath            | 1152    | 1000   |                                                                 |      |                                          |      |                                          |      |               |          |          |  |
| #3       | pub    | .ato                 | 1153    | 1000   | ObjRecv                                                         | Time | ObjRecv                                  | Time | ObjRecv                                  | Time | ObjRecv       | Time     |          |  |
| #5       | sub    | basic, xmlxpath, ato | 151-153 | 1000   | 5656 504750 5732 481313 481659 5656 448359 5732 433339          |      | 5656 484844 10414 473625 1126 449797     |      | 5656 484844 10414 473625 1126 449797     |      | 5732 469422   | 461357.8 | 461620.7 |  |
| #6       | sub    | basic, xmlxpath, ato | 154-156 | 1000   | 5576 504719 5020 482005                                         |      | 5576 448344 5020 433818                  |      | 5576 484859 8693 473609 791 451016       |      | 5020 469828   | 461883.7 |          |  |
| #1       | pub    | basic                | 1151    |        |                                                                 |      |                                          |      |                                          |      |               |          |          |  |
| #2       | pub    | .xmlxpath            | 1152    |        |                                                                 |      |                                          |      |                                          |      |               |          |          |  |
| #3       | pub    | .ato                 | 1153    |        | ObjRecv                                                         | Time | ObjRecv                                  | Time | ObjRecv                                  | Time | ObjRecv       | Time     |          |  |
| #5       | sub    | basic, xmlxpath, ato | 151-153 | 2000   | 22057 2637094 21127 2458479 2458180 22057 2790453 21127 2591843 |      | 22057 2579532 37860 2508687 3463 2216016 |      | 22057 2579532 37860 2508687 3463 2216016 |      | 21127 2434745 | 2495023  | 2494967  |  |
| #6       | sub    | basic, xmlxpath, ato | 154-156 | 2000   | 22686 2635047 20312 2457880                                     |      | 22686 2790453 20312 2591917              |      | 22686 2579609 35484 2508703 2766 2216500 |      | 20312 2434937 | 2494911  |          |  |

## 10.5 Experiment 5

### 10.5.1 Hardware JBI – Object Size 2KB

| Computer | Action | ObjectType | Seed | Object | ObjRecv | Time   | ObjRecv | Time   | ObjRecv | Time   | AvgTime  |
|----------|--------|------------|------|--------|---------|--------|---------|--------|---------|--------|----------|
| #1       | sub    | .xmlkpath  | 171  | 250    | 2843    | 21172  | 2843    | 19813  | 2843    | 19750  | 20245    |
| #2       | pub    | .xmlkpath  | 1171 | 250    |         |        |         |        |         |        |          |
| #3       | pub    | .xmlkpath  | 1172 | 250    |         |        |         |        |         |        |          |
| #5       | pub    | .xmlkpath  | 1173 | 250    |         |        |         |        |         |        |          |
| #6       | pub    | .xmlkpath  | 1174 | 250    |         |        |         |        |         |        |          |
| #1       | sub    | .xmlkpath  | 171  | 500    | 10209   | 39860  | 10209   | 39031  | 10209   | 39750  | 39547    |
| #2       | pub    | .xmlkpath  | 1171 | 500    |         |        |         |        |         |        |          |
| #3       | pub    | .xmlkpath  | 1172 | 500    |         |        |         |        |         |        |          |
| #5       | pub    | .xmlkpath  | 1173 | 500    |         |        |         |        |         |        |          |
| #6       | pub    | .xmlkpath  | 1174 | 500    |         |        |         |        |         |        |          |
| #1       | sub    | .xmlkpath  | 171  | 1000   | 39535   | 90781  | 39535   | 91109  | 39535   | 96109  | 92666.33 |
| #2       | pub    | .xmlkpath  | 1171 | 1000   |         |        |         |        |         |        |          |
| #3       | pub    | .xmlkpath  | 1172 | 1000   |         |        |         |        |         |        |          |
| #5       | pub    | .xmlkpath  | 1173 | 1000   |         |        |         |        |         |        |          |
| #6       | pub    | .xmlkpath  | 1174 | 1000   |         |        |         |        |         |        |          |
| #1       | sub    | .xmlkpath  | 171  | 2000   | 151111  | 239359 | 151111  | 239837 | 151111  | 243484 | 240926.7 |
| #2       | pub    | .xmlkpath  | 1171 | 2000   |         |        |         |        |         |        |          |
| #3       | pub    | .xmlkpath  | 1172 | 2000   |         |        |         |        |         |        |          |
| #5       | pub    | .xmlkpath  | 1173 | 2000   |         |        |         |        |         |        |          |
| #6       | pub    | .xmlkpath  | 1174 | 2000   |         |        |         |        |         |        |          |

## 10.5.2 Software JBI – Object Size 2KB

| Computer | Action | ObjectType | Seed | Object | ObjRecv | Time   | ObjRecv | Time   | ObjRecv | Time   | AvgTime  |
|----------|--------|------------|------|--------|---------|--------|---------|--------|---------|--------|----------|
| #1       | sub    | .xmlkpath  | 171  | 250    | 2843    | 34657  | 2843    | 34125  | 2843    | 33453  | 34078.33 |
| #2       | pub    | .xmlkpath  | 1171 | 250    |         |        |         |        |         |        |          |
| #3       | pub    | .xmlkpath  | 1172 | 250    |         |        |         |        |         |        |          |
| #5       | pub    | .xmlkpath  | 1173 | 250    |         |        |         |        |         |        |          |
| #6       | pub    | .xmlkpath  | 1174 | 250    |         |        |         |        |         |        |          |
| #1       | sub    | .xmlkpath  | 171  | 500    | 10209   | 87641  | 10209   | 85578  | 10209   | 85422  | 86213.67 |
| #2       | pub    | .xmlkpath  | 1171 | 500    |         |        |         |        |         |        |          |
| #3       | pub    | .xmlkpath  | 1172 | 500    |         |        |         |        |         |        |          |
| #5       | pub    | .xmlkpath  | 1173 | 500    |         |        |         |        |         |        |          |
| #6       | pub    | .xmlkpath  | 1174 | 500    |         |        |         |        |         |        |          |
| #1       | sub    | .xmlkpath  | 171  | 1000   | 39535   | 476859 | 39535   | 494750 | 39535   | 484875 | 485494.7 |
| #2       | pub    | .xmlkpath  | 1171 | 1000   |         |        |         |        |         |        |          |
| #3       | pub    | .xmlkpath  | 1172 | 1000   |         |        |         |        |         |        |          |
| #5       | pub    | .xmlkpath  | 1173 | 1000   |         |        |         |        |         |        |          |
| #6       | pub    | .xmlkpath  | 1174 | 1000   |         |        |         |        |         |        |          |
| #1       | sub    | .xmlkpath  | 171  | 2000   | 151111  | crash  | 151111  | crash  | 151111  | crash  | #D/V/O!  |
| #2       | pub    | .xmlkpath  | 1171 | 2000   |         |        |         |        |         |        |          |
| #3       | pub    | .xmlkpath  | 1172 | 2000   |         |        |         |        |         |        |          |
| #5       | pub    | .xmlkpath  | 1173 | 2000   |         |        |         |        |         |        |          |
| #6       | pub    | .xmlkpath  | 1174 | 2000   |         |        |         |        |         |        |          |

## 10.6 Experiment 6

### 10.6.1 Hardware JBI – Object Size 2KB

| Computer | Action | ObjectType | Seed | Object | ObjRecv | Time   | ObjRecv | Time   | ObjRecv | Time   | AvgTime  |
|----------|--------|------------|------|--------|---------|--------|---------|--------|---------|--------|----------|
| #1       | pub    | .xmlkpath  | 1171 | 250    |         |        |         |        |         |        |          |
| #2       | pub    | .xmlkpath  | 1172 | 250    |         |        |         |        |         |        |          |
| #3       | pub    | .xmlkpath  | 1173 | 250    |         |        |         |        |         |        |          |
| #5       | sub    | .xmlkpath  | 171  | 250    | 2843    | 25577  | 2843    | 25768  | 2843    | 24215  | 25186.67 |
| #6       | pub    | .xmlkpath  | 1174 | 250    |         |        |         |        |         |        |          |
| #1       | pub    | .xmlkpath  | 1171 | 500    |         |        |         |        |         |        |          |
| #2       | pub    | .xmlkpath  | 1172 | 500    |         |        |         |        |         |        |          |
| #3       | pub    | .xmlkpath  | 1173 | 500    |         |        |         |        |         |        |          |
| #5       | sub    | .xmlkpath  | 171  | 500    | 10209   | 62800  | 10209   | 62540  | 10209   | 63712  | 63017.33 |
| #6       | pub    | .xmlkpath  | 1174 | 500    |         |        |         |        |         |        |          |
| #1       | pub    | .xmlkpath  | 1171 | 1000   |         |        |         |        |         |        |          |
| #2       | pub    | .xmlkpath  | 1172 | 1000   |         |        |         |        |         |        |          |
| #3       | pub    | .xmlkpath  | 1173 | 1000   |         |        |         |        |         |        |          |
| #5       | sub    | .xmlkpath  | 171  | 1000   | 39535   | 206608 | 39535   | 209101 | 39535   | 208340 | 208016.3 |
| #6       | pub    | .xmlkpath  | 1174 | 1000   |         |        |         |        |         |        |          |
| #1       | pub    | .xmlkpath  | 1171 | 2000   |         |        |         |        |         |        |          |
| #2       | pub    | .xmlkpath  | 1172 | 2000   |         |        |         |        |         |        |          |
| #3       | pub    | .xmlkpath  | 1173 | 2000   |         |        |         |        |         |        |          |
| #5       | sub    | .xmlkpath  | 171  | 2000   | 151111  | 769136 | 151111  | 664865 | 151111  | 768405 | 734135.3 |
| #6       | pub    | .xmlkpath  | 1174 | 2000   |         |        |         |        |         |        |          |

## 10.6.2 Software JBI – Object Size 2KB

| Computer | Action | ObjectType | Seed | Object | ObjRecv | Time   | ObjRecv | Time   | ObjRecv | Time   | AvgTime  |
|----------|--------|------------|------|--------|---------|--------|---------|--------|---------|--------|----------|
| #1       | pub    | .xmlkpath  | 1171 | 250    |         |        |         |        |         |        |          |
| #2       | pub    | .xmlkpath  | 1172 | 250    |         |        |         |        |         |        |          |
| #3       | pub    | .xmlkpath  | 1173 | 250    |         |        |         |        |         |        |          |
| #5       | sub    | .xmlkpath  | 171  | 250    | 2843    | 43543  | 2843    | 35541  | 2843    | 35231  | 38105    |
| #6       | pub    | .xmlkpath  | 1174 | 250    |         |        |         |        |         |        |          |
| #1       | pub    | .xmlkpath  | 1171 | 500    |         |        |         |        |         |        |          |
| #2       | pub    | .xmlkpath  | 1172 | 500    |         |        |         |        |         |        |          |
| #3       | pub    | .xmlkpath  | 1173 | 500    |         |        |         |        |         |        |          |
| #5       | sub    | .xmlkpath  | 171  | 500    | 10209   | 100545 | 10209   | 99733  | 10209   | 92053  | 97443.67 |
| #6       | pub    | .xmlkpath  | 1174 | 500    |         |        |         |        |         |        |          |
| #1       | pub    | .xmlkpath  | 1171 | 1000   |         |        |         |        |         |        |          |
| #2       | pub    | .xmlkpath  | 1172 | 1000   |         |        |         |        |         |        |          |
| #3       | pub    | .xmlkpath  | 1173 | 1000   |         |        |         |        |         |        |          |
| #5       | sub    | .xmlkpath  | 171  | 1000   | 39535   | 646190 | 39535   | 640551 | 39535   | 666378 | 651039.7 |
| #6       | pub    | .xmlkpath  | 1174 | 1000   |         |        |         |        |         |        |          |
| #1       | pub    | .xmlkpath  | 1171 | 2000   |         |        |         |        |         |        |          |
| #2       | pub    | .xmlkpath  | 1172 | 2000   |         |        |         |        |         |        |          |
| #3       | pub    | .xmlkpath  | 1173 | 2000   |         |        |         |        |         |        |          |
| #5       | sub    | .xmlkpath  | 171  | 2000   |         | crash  |         | crash  |         | crash  | #DIV/0!  |
| #6       | pub    | .xmlkpath  | 1174 | 2000   |         |        |         |        |         |        |          |

## 11 List of Acronyms

|       |                                          |
|-------|------------------------------------------|
| B     | 1 byte                                   |
| CAPI  | Common Application Programming Interface |
| CPU   | Central Processing Unit                  |
| HTTP  | Hyper-Text Transfer Protocol             |
| IP    | Internet Protocol                        |
| JB1   | Joint Battlespace Infosphere             |
| JMS   | Java Messaging System                    |
| KB    | Kilobyte (1024 B)                        |
| MB    | Megabyte (1024 KB)                       |
| RMI   | Remote Method Invocation                 |
| SDK   | Software Development Kit                 |
| TCP   | Transmission Control Protocol            |
| UDP   | User Datagram Protocol                   |
| XML   | eXtensible Markup Language               |
| XPATh | XML Path Language                        |