

# Measuring the Performance of XML-Based Data Standards

Gino Crispieri

[gino.crispieri@ismi.sematech.org](mailto:gino.crispieri@ismi.sematech.org)

International SEMATECH  
Manufacturing Initiative

SEMATECH, the SEMATECH logo, AMRC, Advanced Materials Research Center, ATDF, the ATDF logo, Advanced Technology Development Facility, ISMI and International SEMATECH Manufacturing Initiative are servicemarks of SEMATECH, Inc. All other servicemarks and trademarks are the property of their respective owners.

INTERNATIONAL SEMATECH

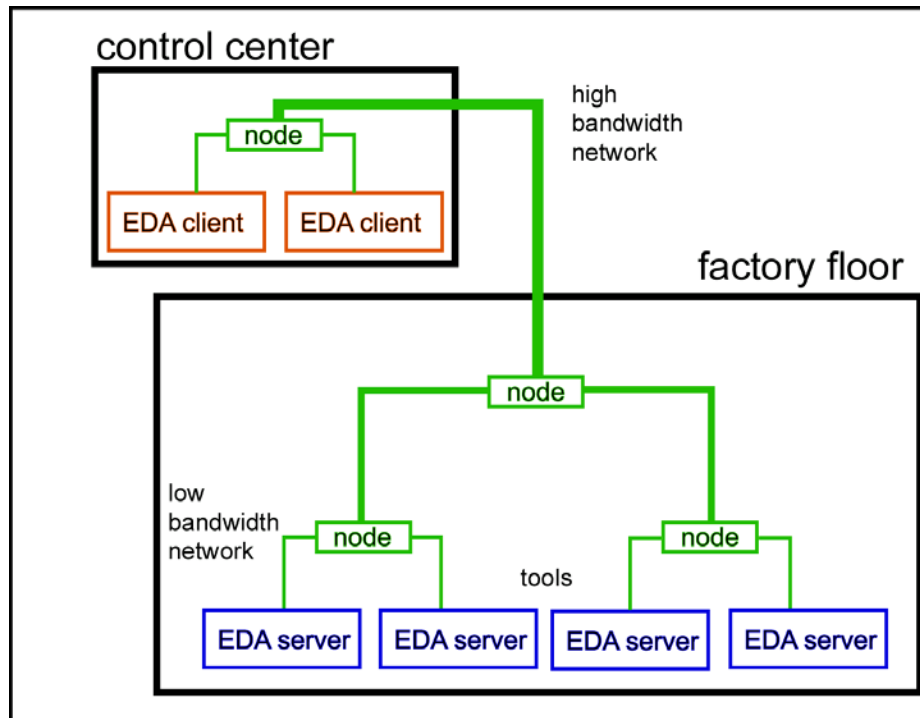


# Content

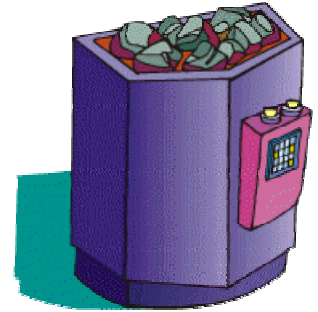
- **NIST Marshaling Performance Testing**
  - **Results based on Castor**
- **XML SOAP Package Performance Measurement**
  - **JavaSOAP**
  - **C++SOAP**
- **Lessons Learned**

# Common Manufacturing Environment

- More than 500 discrete instruments on the facility floor
- Typical factory networks equipment with 100 Mbps Ethernet with 1 Gbps backbone network to control room



# Equipment Requirements

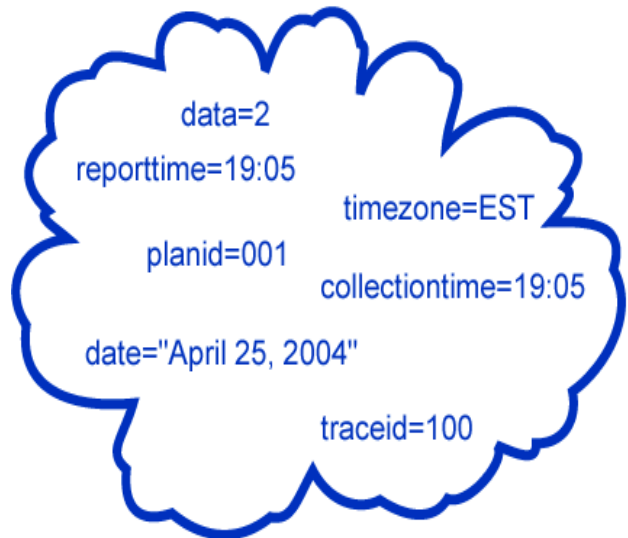


- The technology used for data transfer must support a maximum throughput of 10,000 data values per second or at least one trace message of 1000 variables at sample period of 100 milliseconds
- The typical tool should provide 50 variables per chamber at a maximum on-tool sample rate of 10Hz
- For some specialized processes such as rapid thermal and flash anneal, an additional smaller number of (up to, for example, a total of 30-40) critical variables may require a maximum on-tool sample rate of 200Hz
  - EEC High Level Requirements for APC
    - [ismi.sematech.org/emanufacturing/docs/EECReqs.pdf](http://ismi.sematech.org/emanufacturing/docs/EECReqs.pdf)

# What is Marshaling?

marshaling framework  
orders data according to  
mapping file

## E134 Message example



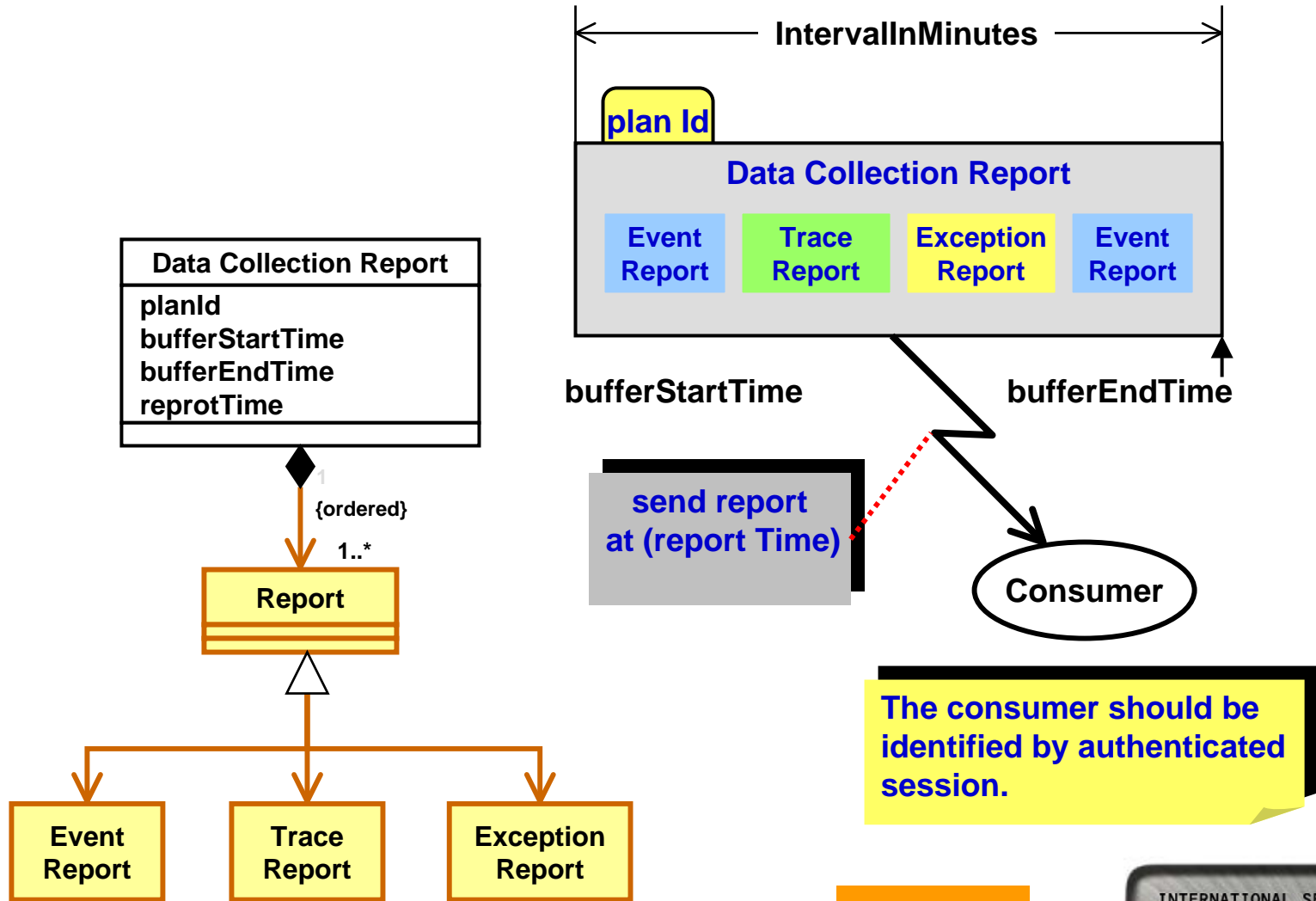
data resides inside  
computer program



```
<NewDataNotification>  
  <DataCollectionReport planID="001" reportTime="2004-04-25T19:05:27.456-04:00">  
    <Reports>  
      <Trace traceid="100" reportTime="2004-04-25T19:05:26.304-04:00">  
        <TraceResults>  
          <Data collectionTime="2004-04-25T19:05:25.273-04:00">  
            <ParameterValues>  
              <PV>  
                <EI>2</EI>  
              </PV>  
            </ParameterValues>  
          </Data>  
        </TraceResults>  
      </Trace>  
    </Reports>  
  </DataCollectionReport>  
</NewDataNotification>
```

formatted data  
is output

# Data Collection Report



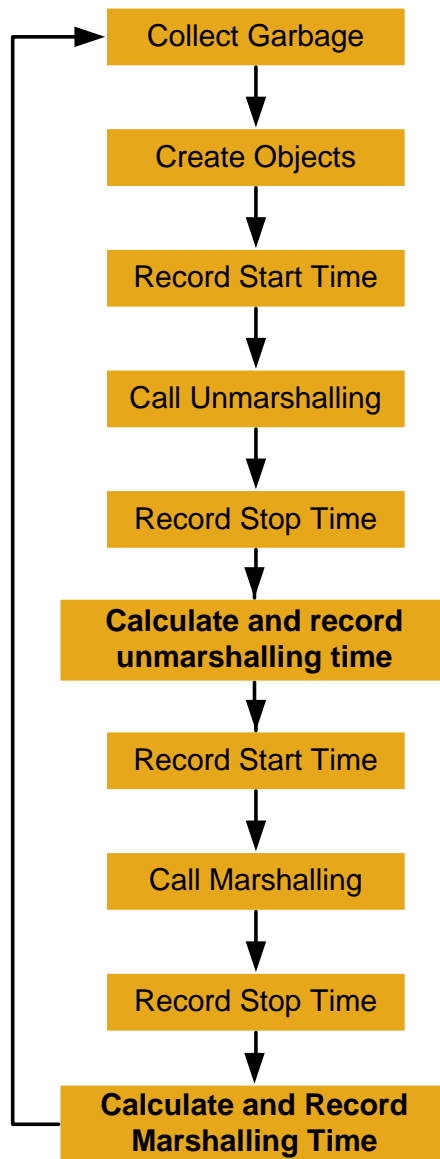
E134

# Hardware Setup

- **1.3 GHz Athlon Thunderbird CPU**
  - 100Mhz bus , 512M RAM, 7200 rpm IDE hard drive.
- **Linux Operating System**
- **Five levels of data were created containing**
  - 10, 50, 100, 500, 1000, 5000 and 10,000 data points
  - integer data values ranging from 0 to 100,000
  - double precision floating point data values ranging from 0.0 to 1.0



# Testing Flowchart



Java collect garbage before running the test

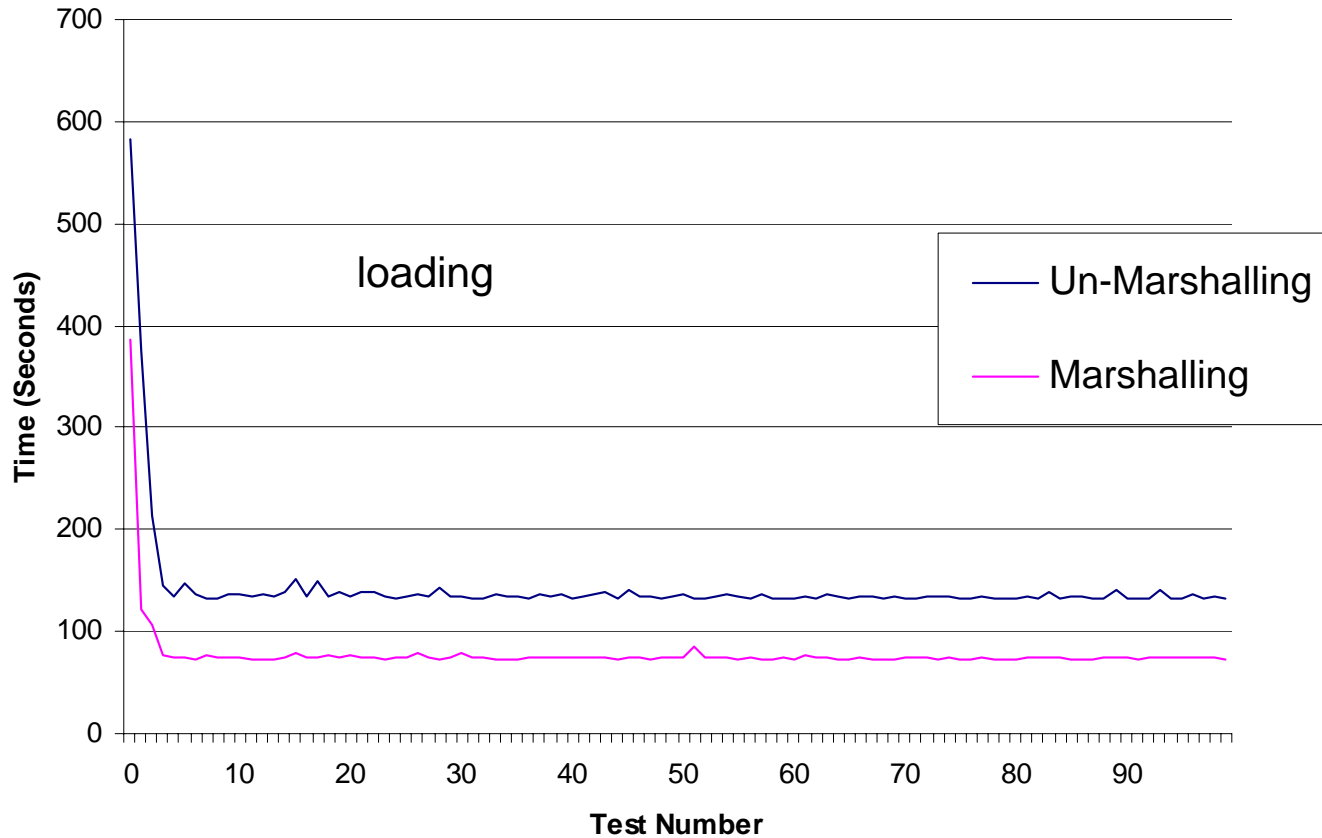
Create timer objects and object to store unmarshalled data

Unmarshal from XML instance, using mapping file, into Java objects

Marshal data from Java objects into XML instance using mapping file

# Trace Report with 500 Parameter Values

500f8 no-X

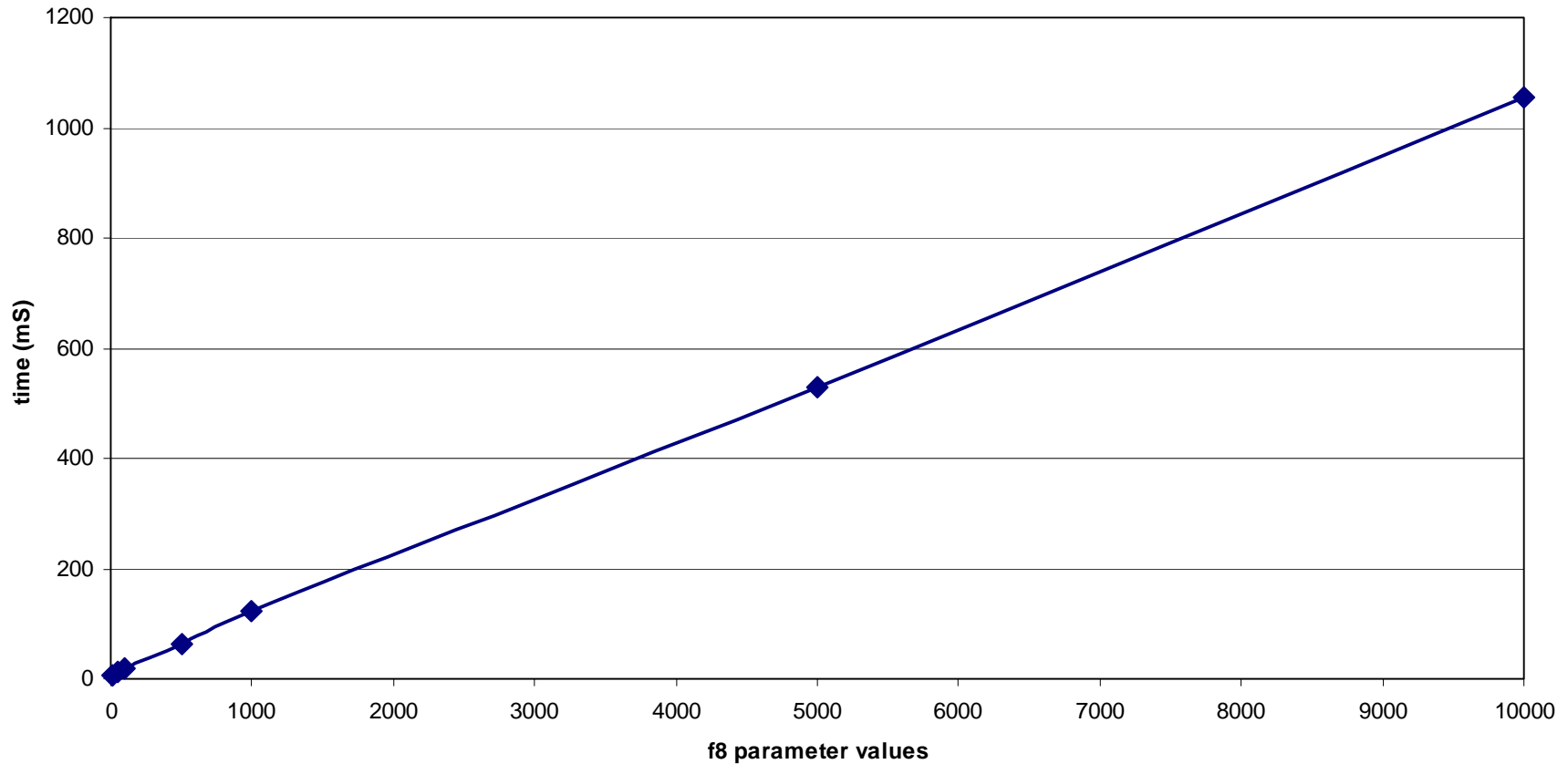


INTERNATIONAL SEMATECH



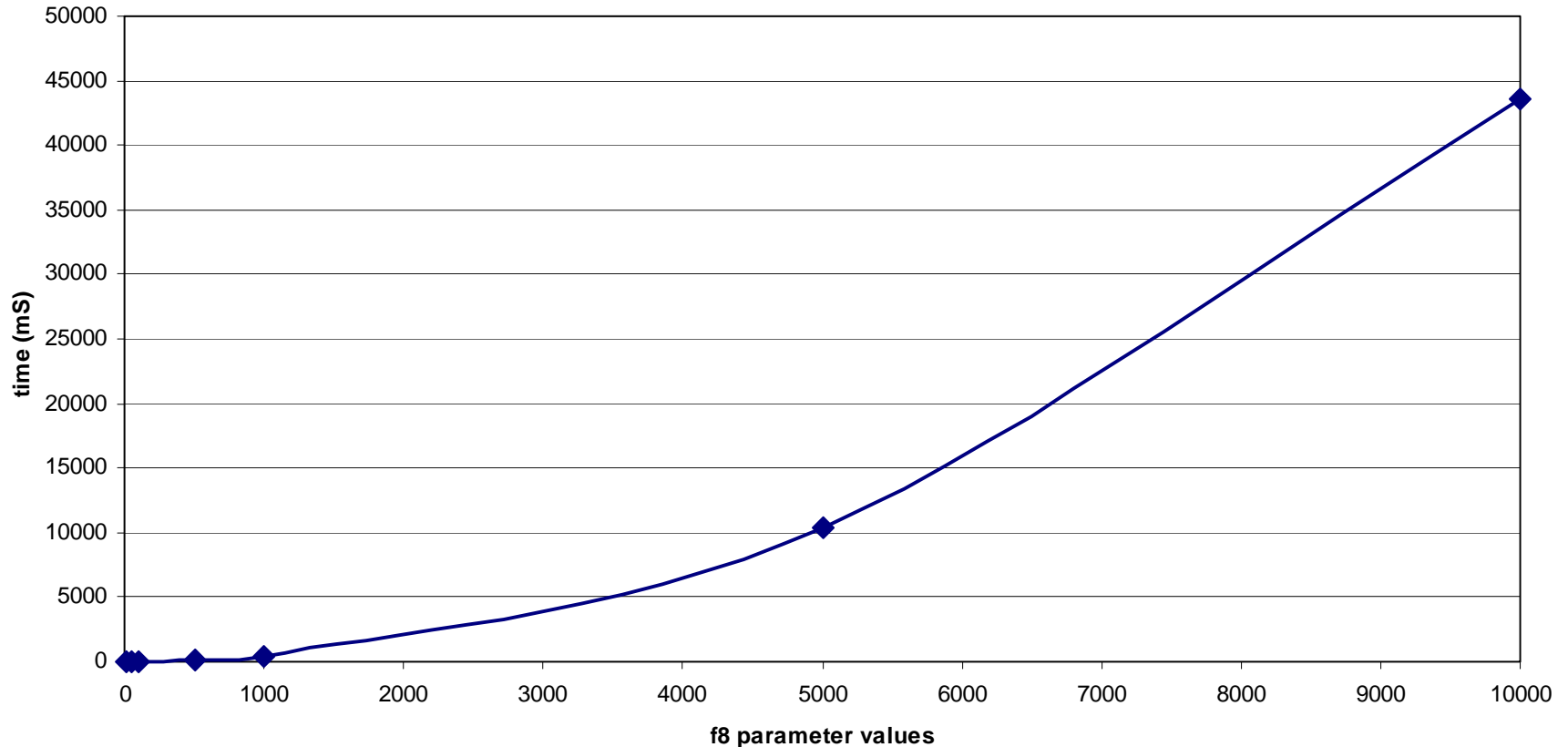
# Marshalling Speed with 1 to 10,000 Parameter Values

Marshalling 10-10,000 F8 Parameter Values

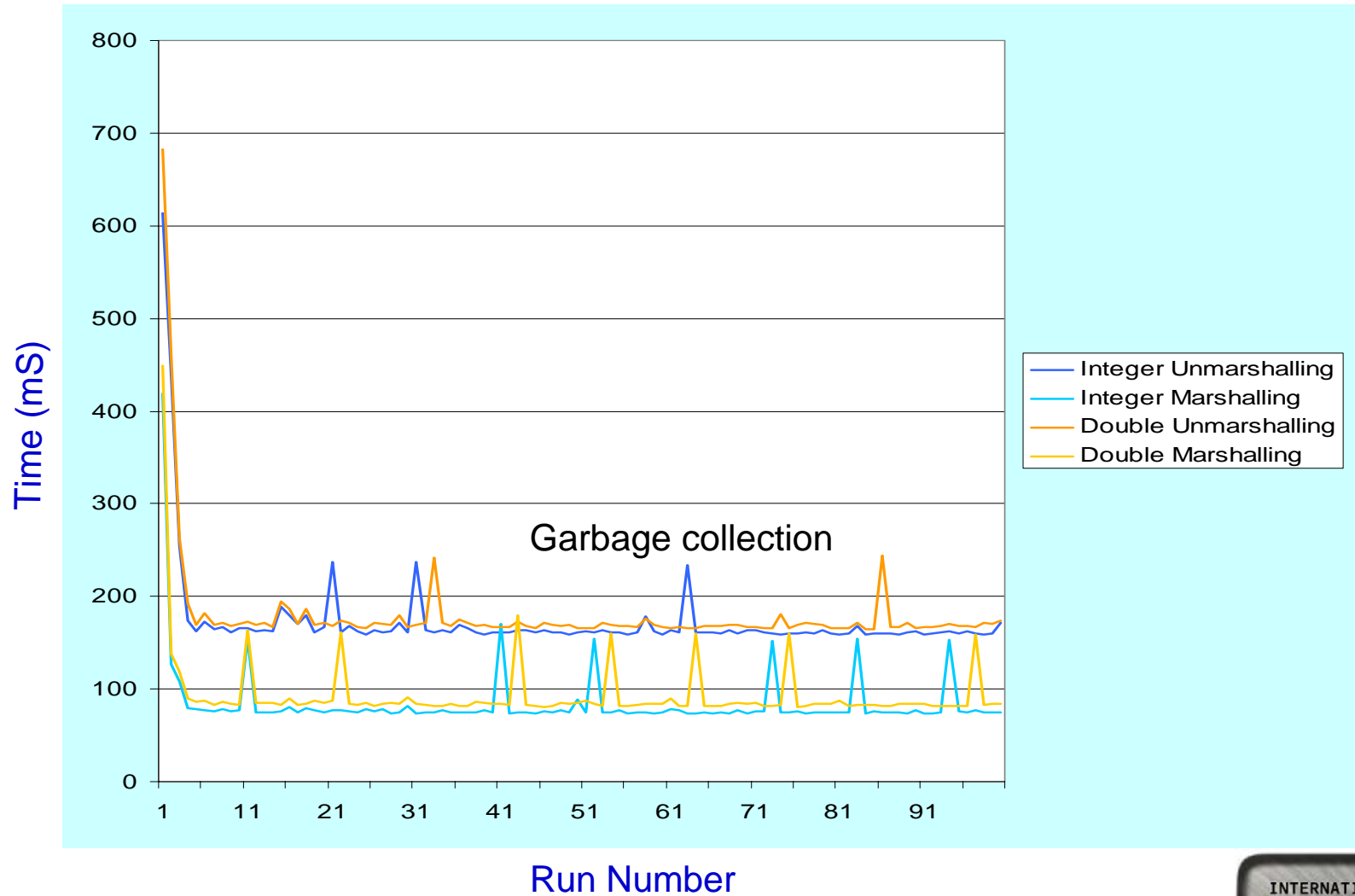


# Un-Marshalling Speed with 1 to 10,000 Parameter Values

Unmarshalling 10-10,000 F8 Parameter Values



# Integer vs Double Precision Parameter Value Performance



# NIST Summary of Results

- Initial loading of mapping file increases marshalling/unmarshalling time of first test
  - Object creation causes big performance hit
  - Other processes running concurrently while marshalling cause non-negligible delays
- There is little performance difference between an instance with integer values and an instance with double precision values
- Marshalling/Unmarshalling times scale approximately linearly with processor speed
- Unmarshalling scales exponentially with parameter values

# XML SOAP Package Performance

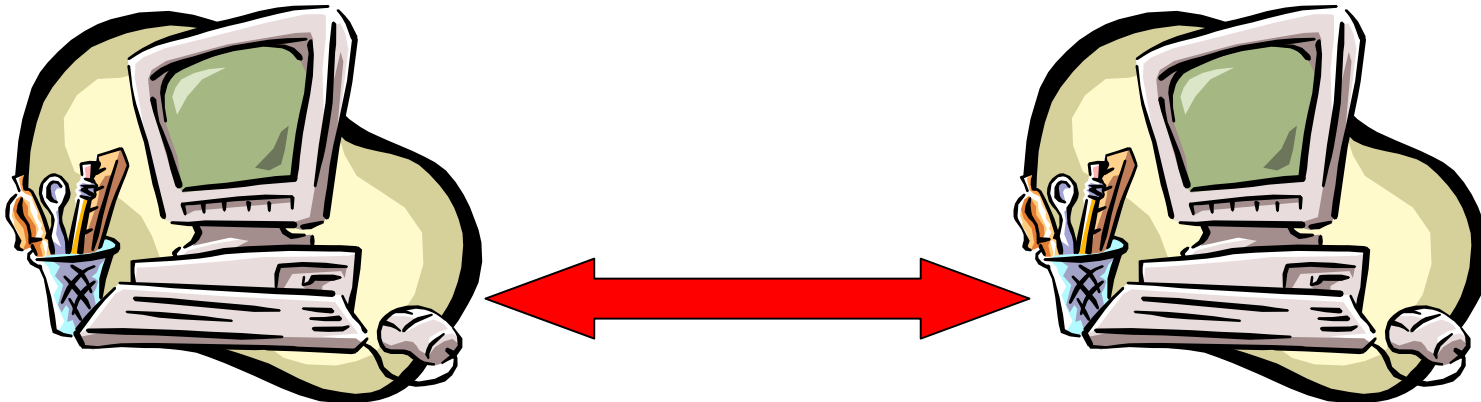
- Two SOAP packages were analyzed by CenterPoint GmbH
  - A Java language SOAP package (will call “JavaSOAP”)
  - A C++ language SOAP package (will call “C++SOAP”)
- Each implementation had Sender and Receiver
  - The sender was responsible for generating data, creating SOAP message from the data, and sending the SOAP message
  - The receiver was responsible for receiving the SOAP message, parsing the message, and processing the data
  - The time taken to perform these steps was measured
- Additionally an overall sender and receiver total time was measured
- CPU load and Page Fault Rate for the test scenarios were also measured

# Measurements

<i>Field</i>	<i>Defined</i>
<b><i>DATATIMERECEIVER</i></b>	The time required by the receiver application to process the data. In the case of our testing, the data is written to a file.
<b><i>XMLTIMERECEIVER</i></b>	The time required by the receiver to parse the XML document and to receive the message.
<b><i>DATATIMESENDER</i></b>	The time required by the sender to generate the values for transmission.
<b><i>XMLTIMESENDER</i></b>	The time required by the sender to construct the XML document and send the message.
<b><i>CPULOAD</i></b>	CPU Load as a percentage. Obtained from the operating system.
<b><i>PAGEFAULTRATE</i></b>	Page Faults / second. Obtained from the operating system.

# Hardware Specifications

- The two computers used had the following specifications:
  - **Computer** Reymont PX8
  - **CPU** Pentium IV 2.8GHZ S478 FSB800 (1MB) Hyper Threading
  - **RAM** Dimm Module 256MB DDR
  - **Hard Disk** 36.6 GB Seagate Cheetah U32 4.7ms 10.000rpm, 4MB Cache ST336607W
  - **O/S** Windows 2000 SP 4 and Linux

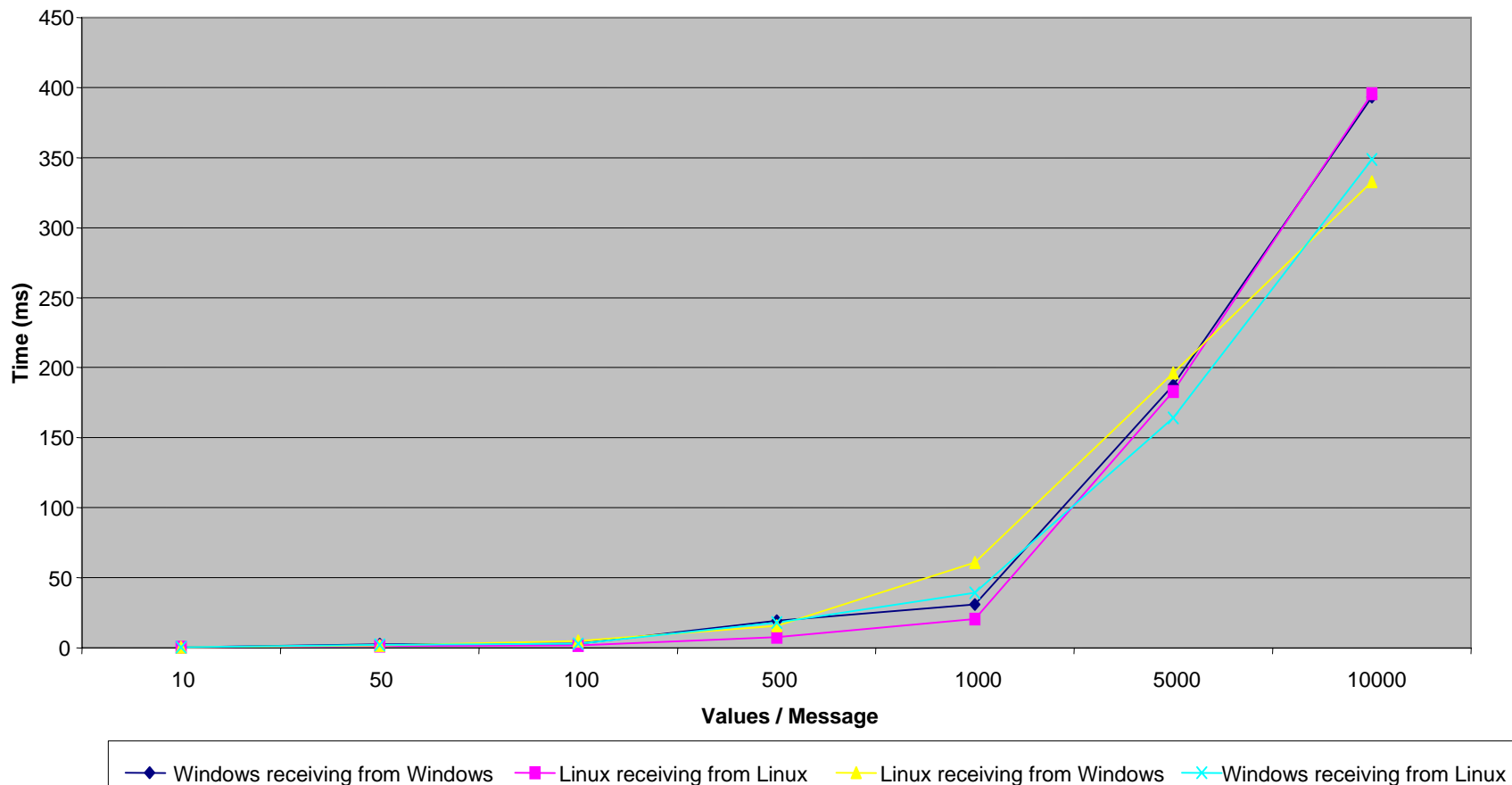


# Statistics

- Each test case has more than 100 numerical results that were averaged and plotted
- The standard deviations are used to generate confidence intervals using a Student's T distribution, which assumes a Gaussian distribution for the response variables.
- A 95% confidence interval of +/- 1.5 for a mean value 15 means that, while we report 15 as the average, the true running is within 13.5 to 16.5, 19 times out of 20
- All confidence intervals are valid 19 times out of 20, or 95% of the time.

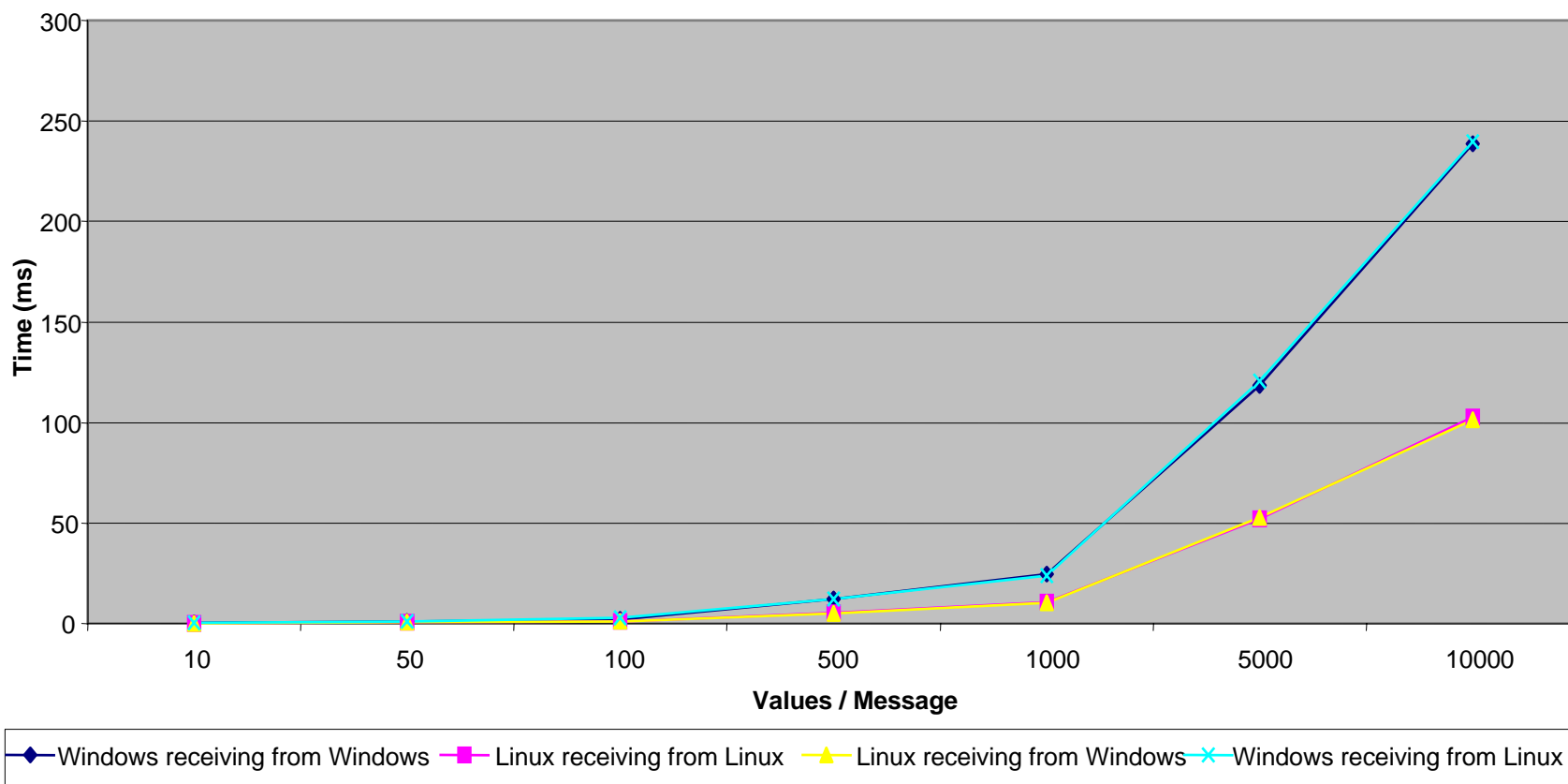
# XML Time Receiver JavaSOAP

XMLTIMERECEIVER: JavaSoap to JavaSoap



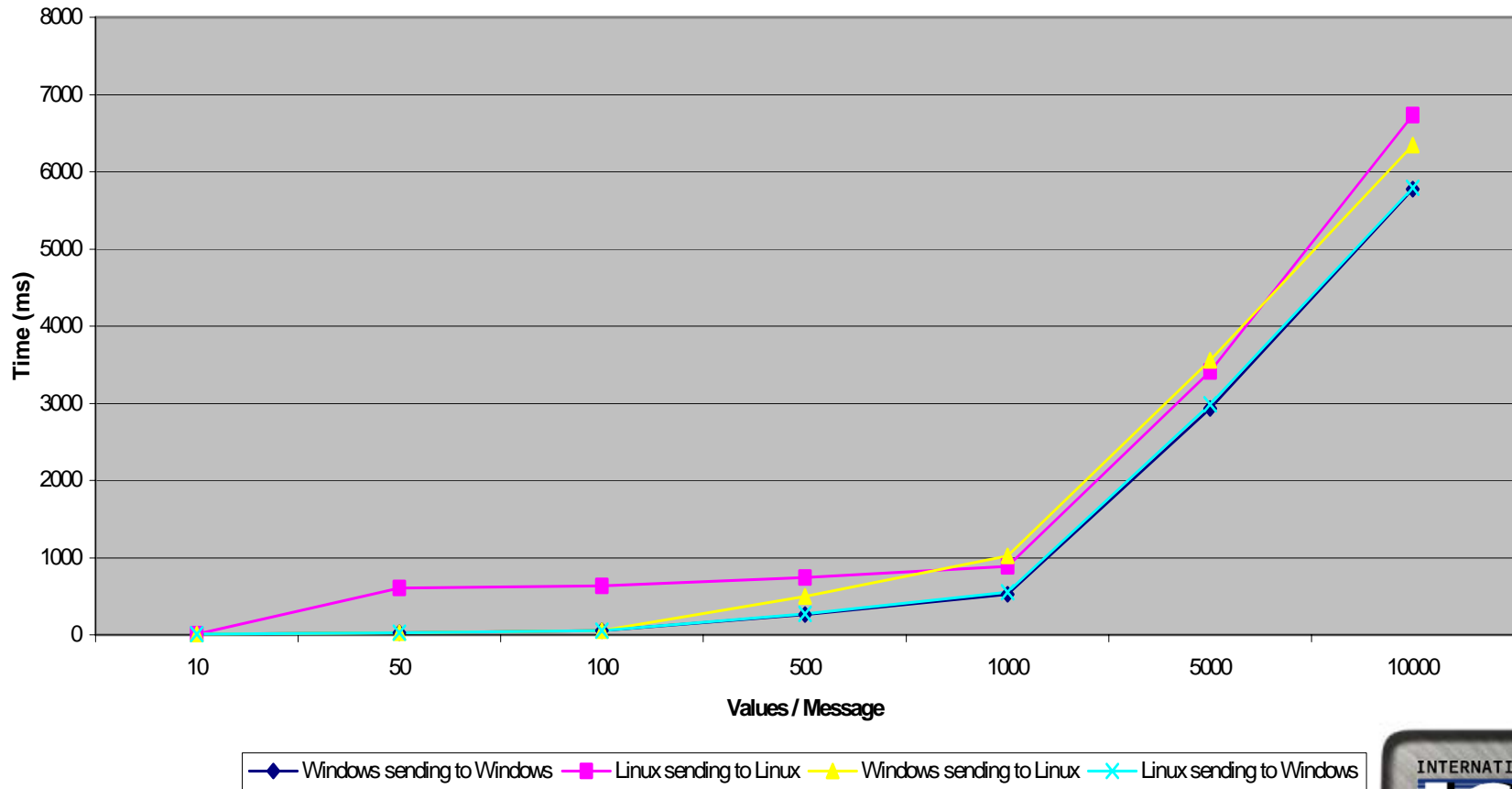
# XML Time Receiver C++SOAP

XMLTIMERECEIVER: C++Soap to c++Soap



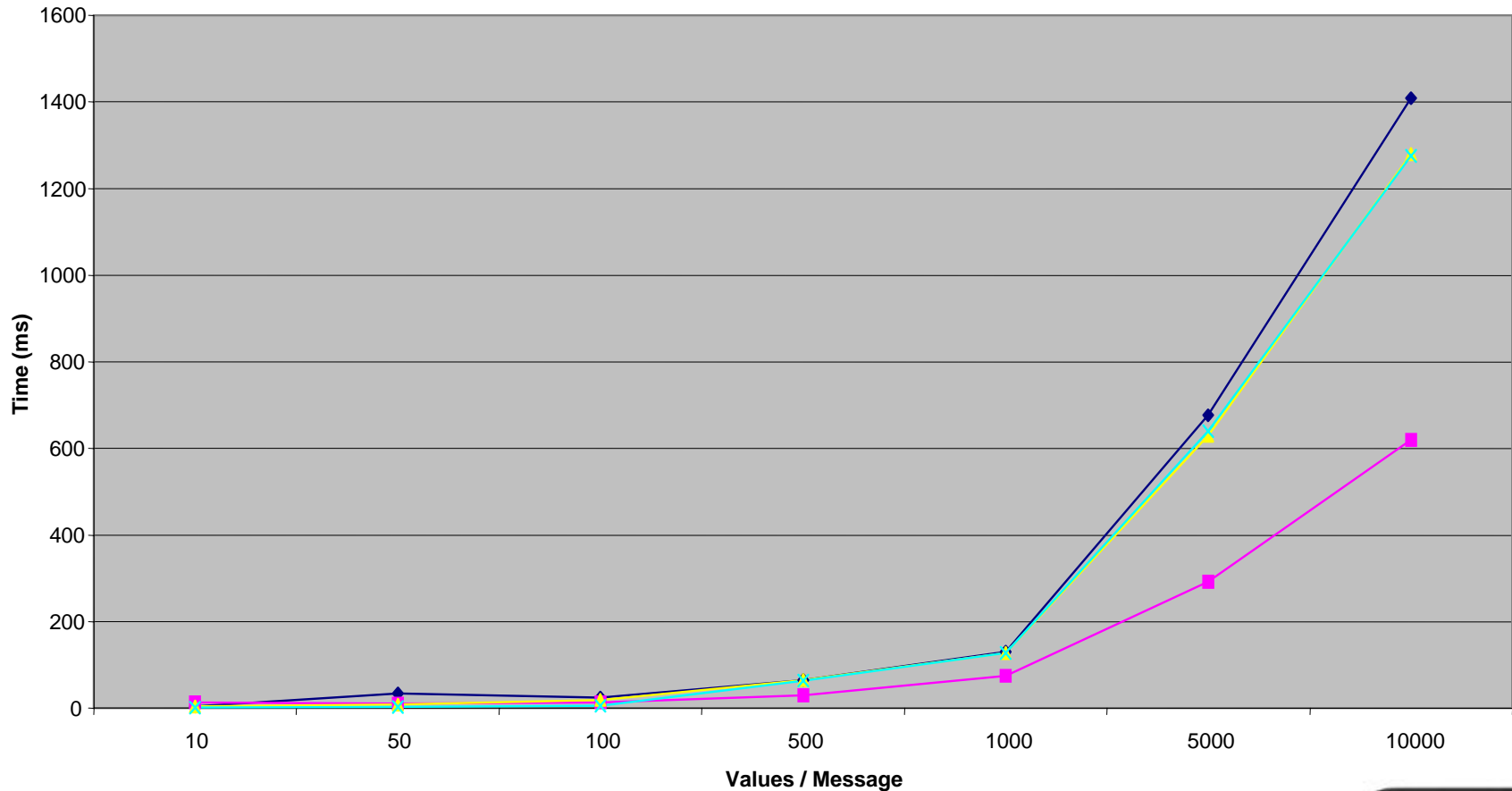
# XML Time Sender Java-SOAP

XMLTIMESENDER: JavaSoap to JavaSoap



# XML Time Sender C++SOAP

XMLTIMESENDER: C++Soap to C++Soap

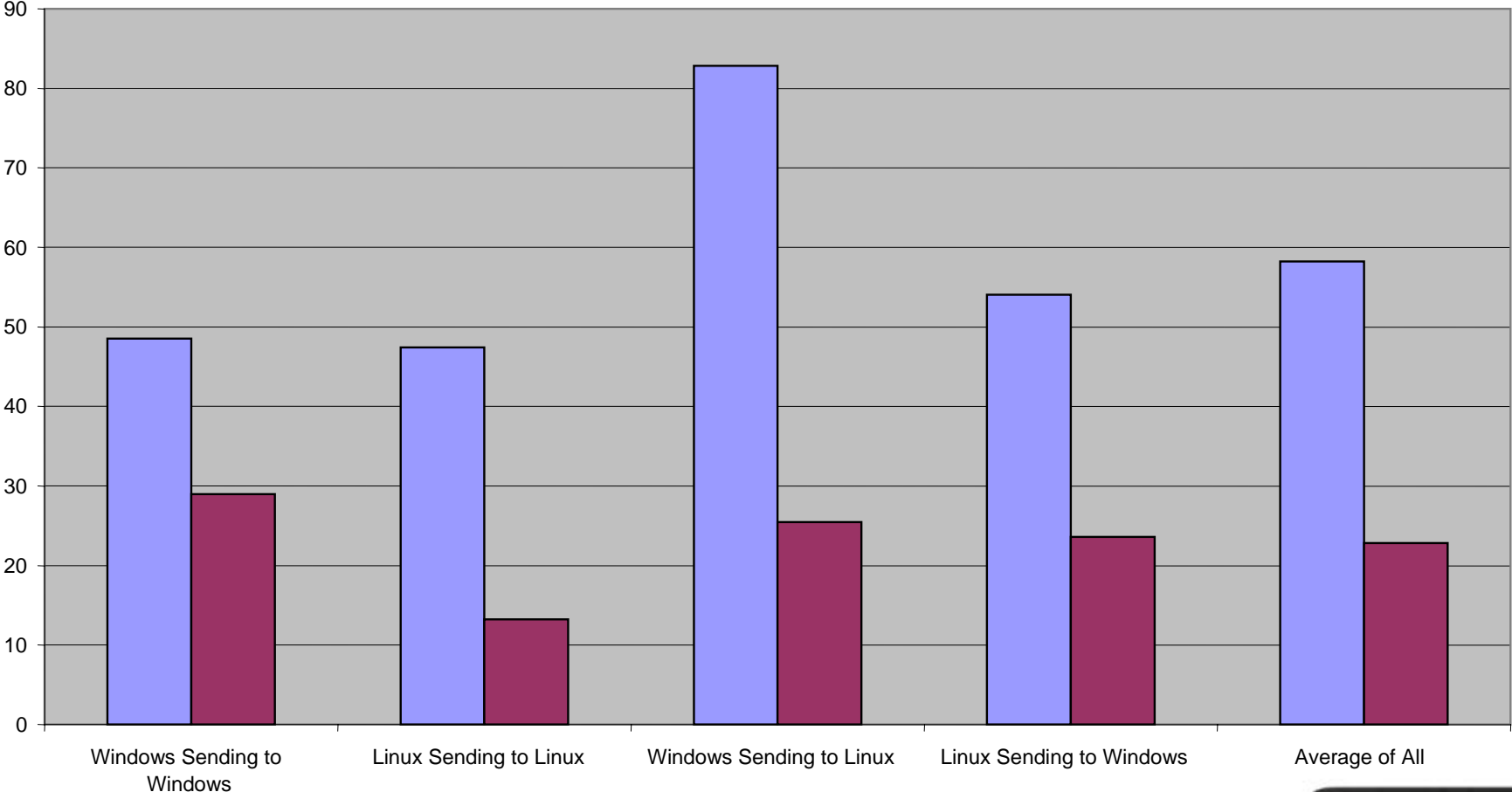


—◆— Windows sending to Windows —■— Linux sending to Linux —▲— Windows sending to Linux —×— Linux sending to Windows

# CPU Load Sender

CPU Load Sender  
JavaSoap, C++soap

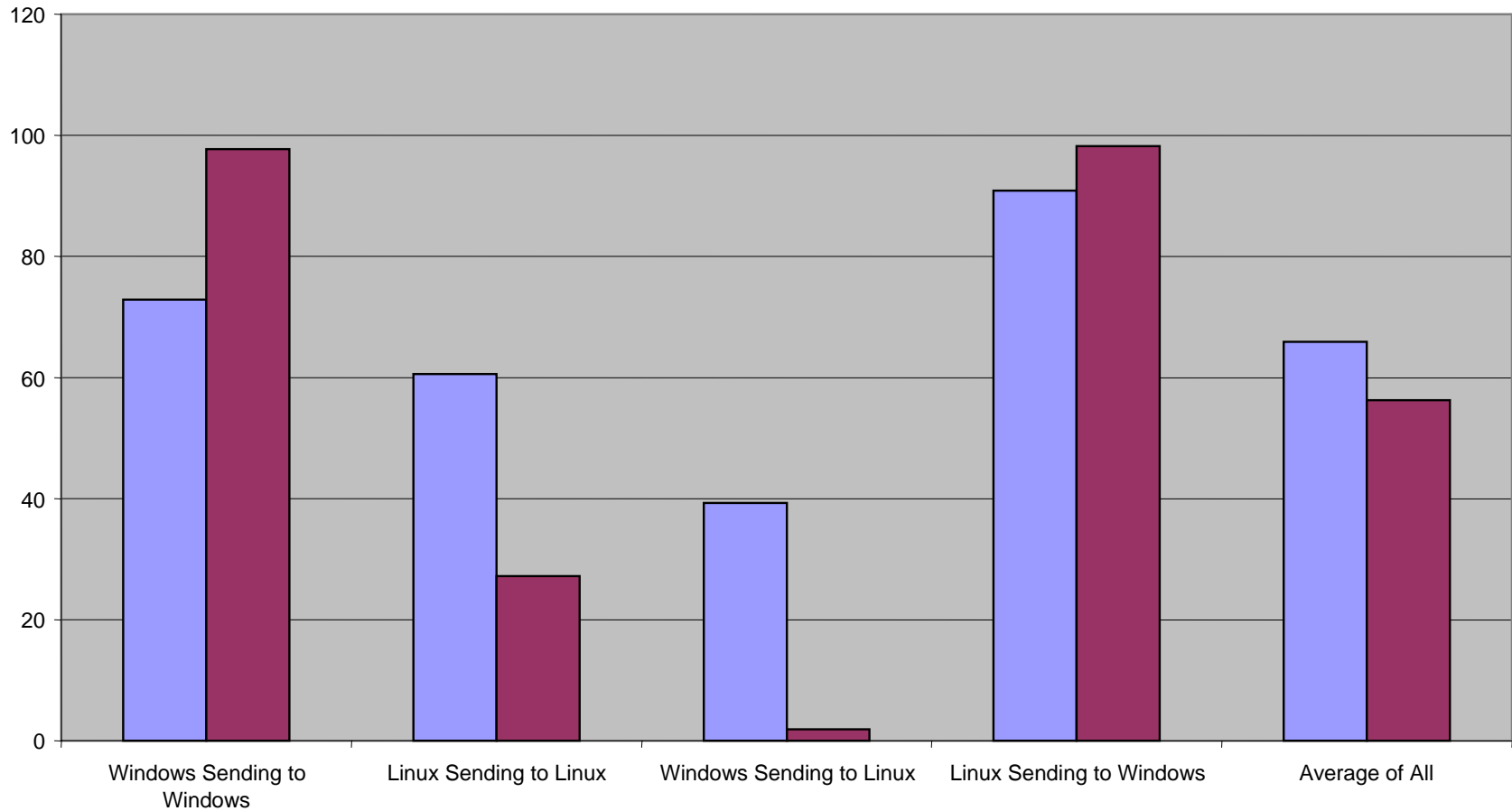
■ C++Soap to C++Soap  
■ JavaSoap to JavaSoap



# CPU Load Receiver

CPU Load Receiver  
JavaSoap, C++soap

■ C++Soap to C++Soap  
■ JavaSoap to JavaSoap



# CPU usage Results

- C++Soap outperformed JavaSOAP with respect to speed, on every test.
- JavaSOAP outperformed C++Soap [in terms of resource consumption] in our hardware tests.
  - JavaSOAP always used less CPU than C++Soap.
  - Page faults rates were similar for the receiver regardless of SOAP package, but the JavaSOAP sender had significantly less page faults than the C++Soap sender.
  - Page fault rates for the sender and receiver were less under Linux than for Windows. Soap package did not affect this.
- CPU Time was much less when using JavaSOAP than when using C++Soap. Linux used marginally less CPU time than windows did.

# Interoperability

- Both packages provide code generation utilities that take a WSDL and schema documents and creates **sending/receiving** code for their SOAP framework
- JavaSOAP implementation generated invalid (non conformant) messages
  - Own mechanism to optimize message exchange
- C++SOAP implementation generated valid (conformant) messages
  - Qualified each tag with a namespace prefix (unnecessary)
  - SOAP Message from JavaSOAP:

```
<TR soapenc:arrayType='n3:CollectedDataType[1]'\>  
  <i><PV soapenc:arrayType='n3:ParameterValueType[2]'\><i>  
    <Arr xsi:nil='1'\></Arr>  
    <F8>884.27734375</F8>
```

- Soap Message from C++Soap:

```
<ns2:TR collectionTime="1970-01-02T04:46:39+02:00"\>  
  <ns2:PV>  
    <ns2:F8>884.27734375</ns2:F8>
```

# Lessons Learned

- Using generic Parsers for marshaling and un-marshaling can affect performance
  - Castor, JIBX, Gnome, JABX, XBinder, Liquid XML, LMX, .net & others
  - Knowledge a must for xml, wsdl, SOAP specs, WSI Basic Profile Rules
- There are many SOAP commercial packages for the computer language of your choice
  - GSOAP, .Net Framework, Websphere, IONA XML, TCLSoap, SoapRMI, GLUE, Apache Axis, and many others
- Operating system can have impact on performance
  - Be careful of background processes (GUI, other applications)
  - Windows 2000 vs XP, Unix vs Linux, Solaris, VMX, others
- Interoperability can be a problem if you do not ensure your toolkit supports the W3C and WS-I specifications
  - EDA specs use “document/literal encoding”
  - Make sure the soap package supports this encoding and conforms to the Web Services Interoperability (WS-I) organization’s Basic Profile
    - See <http://www.ws-i.org>
  - It is possible to create map files to fix this problem but performance can be affected
- Do a complete performance and resource consumption characterization of your candidate toolkits before selecting one for implementation