webMethods Integration Server Socket Utility Package

SimpleSocket Package User Guide
Version 1.0

Prepared by
Igor Androsov.

February 15, 2005

Date	Author	Rev	Change Description
Feb 15, 2005	Igor Androsov	0.1	Created initial draft.
Mar 25 2005	Igor Androsov	0.2	Correction of errors and format
Mar 28 2005	Igor Androsov	0.3	Update sections with default protocol information

Overview

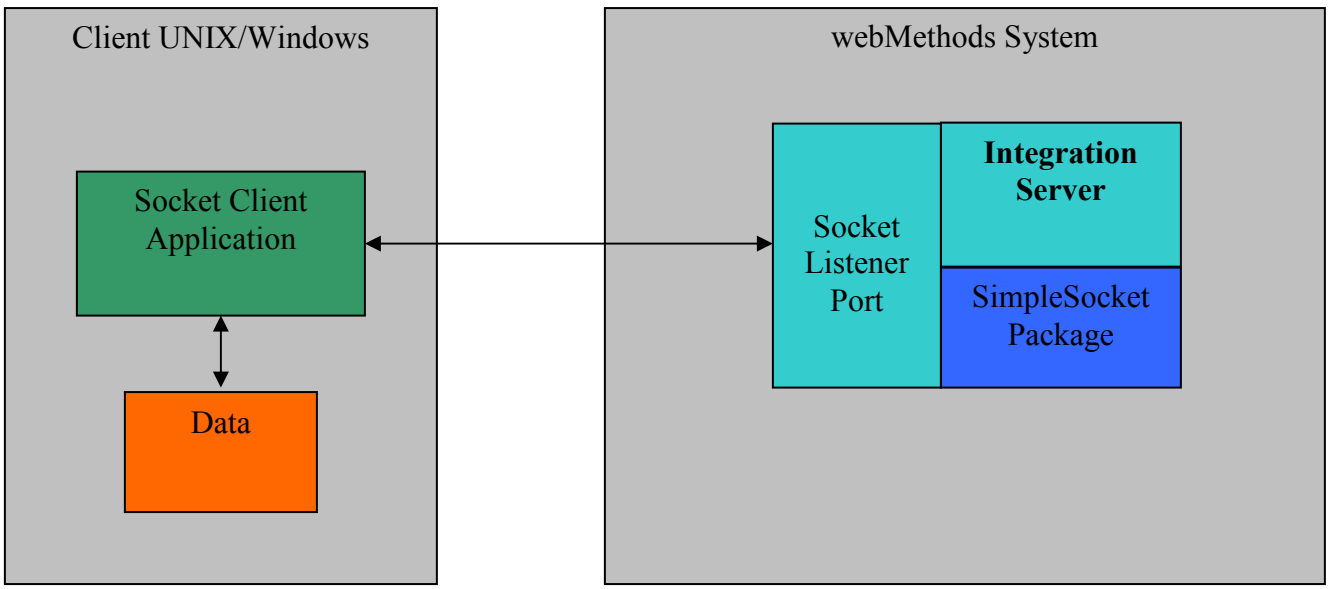
The SimpleSocket utility package for webMethods Integration Server has been designed to enable standard TCP server socket listener. SimpleSocket package integrates as component into webMethods Integration Server and provides additional port listener type with functionality of TCP socket interface.

Purpose

This document describes the technical information for webMethods SimpleSocket utility package, and its development and deployment details.

Architecture Overview

The SimpleSocket package is designed for webMethods Integration Server. The SimpleSocket package adds a new socket type listener to Integration Server standard listeners and ports. It provides TCP socket server that allows standard TCP socket clients to connect and invoke services on Integration server via custom protocol. SimpleSocket package enables new and existing legacy socket based client applications to deliver data to webMethods platform.



SimpleSocket Package Detailed Description

SimpleSocket Package Components

This section specifies the components of SimpleSocket package.

SimpleSocket.zip – Integration Server standard package.

System Requirements

This section specifies SimpleSocket Package's system requirements.

Platform and Operating System	Supported JREs
Microsoft Windows 2000 Professional, Server, and Advanced Server	JRE 1.3.1 and 1.4.2
Microsoft Windows XP Professional	JRE 1.3.1 and 1.4.2
Microsoft Windows Server 2003 Standard Edition, Enterprise Edition (32-bit), and Datacenter Edition (32-bit)	JRE 1.3.1 and 1.4.2
Sun Solaris 8 and 9 (both 64-bit)	JRE 1.4.2
Hewlett-Packard HP-UX 11i v1 for PA-RISC(64-bit) and HP-UX 11i v2 for Itanium2	JRE 1.4.1
IBM AIX 5.1 and 5.2 (both 64-bit)	JRE 1.4.1
Red Hat Enterprise Linux AS 2.1 and ES 2.1	JRE 1.4.2

SimpleSocket Configuration

This section describes the configuration available for SimpleSocket package. All configuration of SimpleSocket package is performed from Integration Server Administrator. User defines a socket port and listener similar to Integration Server's standard HTTP or FTP port configuration. For information on port definition refer to Integration Server Adminstrator's Guide.

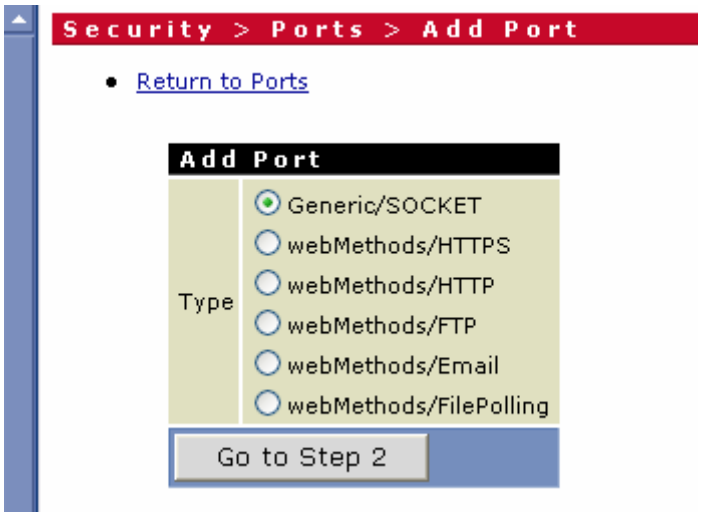
The webMethods Integration Server by default provides 4 types of ports:

- webMethods/HTTPS
- webMethods/HTTP
- webMethods/FTP
- webMethods/Email

With an addition of flat file processing (WmFlatFile package) it adds a special file polling port, webMethods/FilePolling.

The SimpleSocket package extends this further by providing an additional port/listener type: Generic/SOCKET. Once the package is installed on Integration Server the new port type is

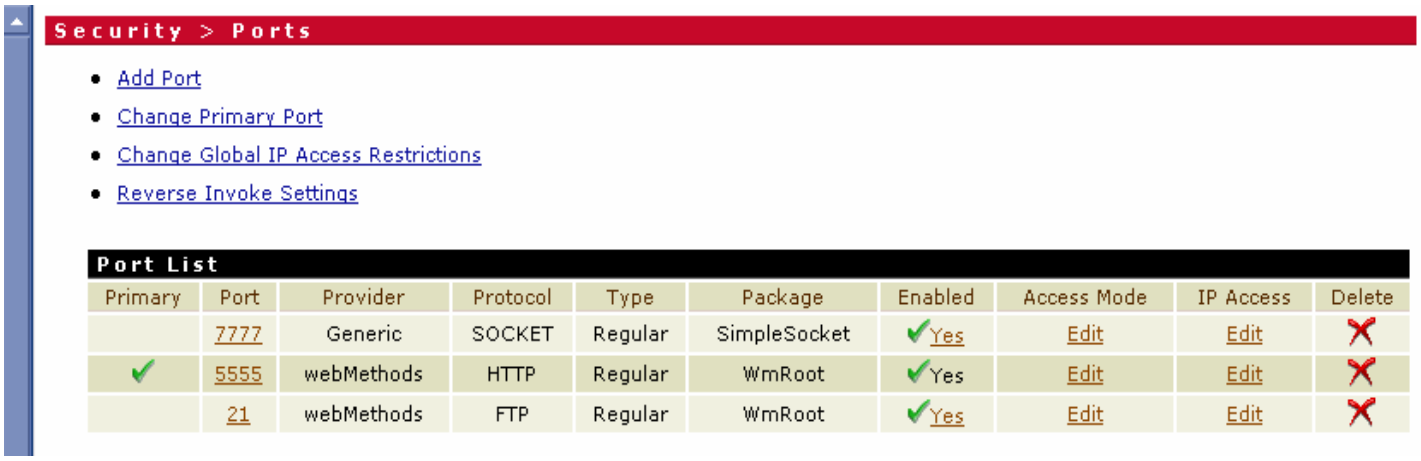
automatically added to the list. User can define a new Generic/SOCKET port like any other standard port types by selecting: Security > Ports > Add Port.



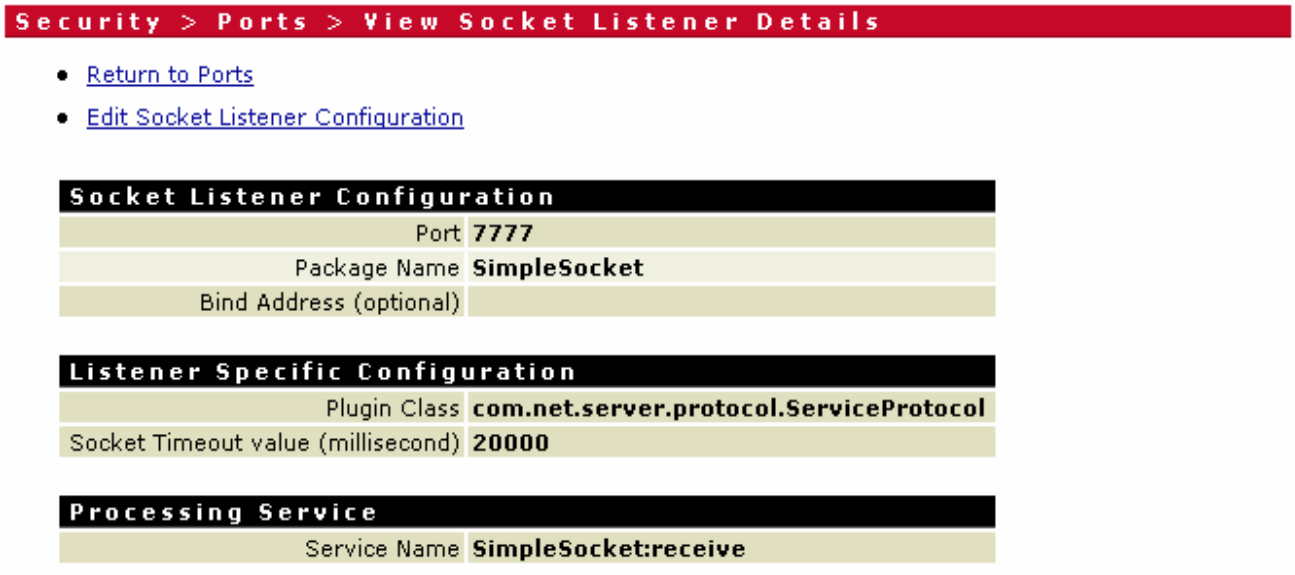
Define Socket Port

The SimpleSocket package provides a TCP socket server port listener that will be enabled as soon as user enables the port from administrator. The default package is delivered with default standard protocol implemented by following class: com.net.server.protocol.ServiceProtocol and predefined sample listener. The listener will be loaded but not activates after SimpleSocket package has been loaded successfully. The sample socket port is defined to use port 7777. The sample protocol designed to work with processing service provided by this package. User may choose to use this definition for reference or completely remove it. User can define any number of socket ports on a single Integration server. All user defined ports may use the same protocol or new user defined protocols.

To define new or manage existing socket port select: Security > Ports.



From this page user can manage Generic/SOCKET type ports. All new port definitions created as standard Integration Server ports. The difference is in providing protocol class and timeout values for socket port.



The following table describes settings available for socket port:

Option	Description
Port	Any available port number on the local system. If given port is already in use the port/listener will not be enabled and error will be thrown by the server.
Package Name	Name of package in local Integration Server where listener configuration will be stored and socket port will be associated with. Default value is WmRoot. If no value selected here the listener configuration will be saved in WmRoot package.
Bind Address (optional)	Optional value used to bind server socket to specified network address.
Plug-in Class	Java class that must extend ContentHandler to provide user defined protocol to interact with Integration server and manage data.
Socket Timeout Value	Value in milliseconds, time that server socket must wait for response from client until disconnecting and closing client socket with timeout error
Processing Service	Processing Service is optional parameter. The processing service supported by built in protocol class com.net.server.protocol.ServiceProtocol and will be functional only if port has specified this class as plug-in. Provide a full name of service to be invoked by socket server on data arrival. If other protocol is used as plug-in this parameter will be ignored. Default value set is SimpleSocket:receive. This service provided by the package as an example.

Processing Service

This feature is implemented by this package and enables user to specify any service on Integration Server to be invoked when socket port receives data. This allows for easy integration without custom development. User can write a flow or Java services similar to those provided by this package SimpleSocket:receive and SimpleSocket.util:test to process incoming data from a socket input stream. These services will act as triggers on socket data and can rout processing to other services. The implementation plug-in class for processing service is provided by default with this package - com.net.server.protocol.ServiceProtocol. This class must be used as plug-in for the port in order to use processing service. The service invoke can only be preformed in synchronous mode. The Asynchronous execution is not available with this release.

Implementing Custom Socket Protocol

This section provides details on creating a custom socket protocol.
The package provides 2 default implementations of protocol defined to invoke default processing service and provide data stream that carry the services name as part if the data respectively, *com.net.server.protocol.ServiceProtocol* and *com.net.server.protocol.InvokeProtocol*. These 2 protocol classes provide case where client does not know anything about webMethods and just sends data to generic socket server. The second case the client explicitly controls what service to invoke to process data.

In many cases default socket protocol that provides ability to invoke a service and deliver data to Integration Server is enough for simple integration. However, there are also many situations where user may need to define a custom protocol that matches existing client application as an example. The default protocol provided with this package also has source code that serves as an example for implementation of any custom protocol.

To implement a new custom protocol user must create a new protocol class that extends the `com.net.server.protocol.ContentHandler` class and implements mandatory public method - `void receiveData()`. The protocol operates on IO streams that it gets from socket to read and write data.

Package Structure

It is recommended to implement new protocol class in same Java package structure as default: `package com.net.server.protocol`. However, it's not required to use same package, user may choose any package structure for his custom protocol.

Imports

The following list of imports typically is required to implement a custom protocol:

```
import java.io.IOException;
import java.io.*;
import java.util.*;
import com.wm.data.IData;
import com.wm.data.IDataCursor;
import com.wm.data.IDataFactory;
import com.wm.data.IDataUtil;
```

Protocol Class Example

Bellow is an example of a protocol class InvokeProtocol to illustrate how to design custom protocol. This class is incomplete it is missing the data processing code that is user specific. For complete example see the InvokeProtocol.java file under package documentation directory.

```
public class InvokeProtocol extends ContentHandler
{
	public InvokeProtocol()
	{
	}

	public void receiveData()
	{
		InputStream is    = null;
		OutputStream os   = null;
		BufferedReader in = null;
		PrintWriter out   = null;

		try
		{
			// Get the socket input/output streams
			is = getInputStream();
			os = getOutputStream();

			out = new PrintWriter(os, true);
				in = new BufferedReader(new
				InputStreamReader(getInputStream()));

			//String inputLine;
			StringBuffer inputLine = new StringBuffer(64000);
			// Loop to read lines of data while the socket is opened
			while ((inputLine.append(in.readLine()).length()) != 0)
			{
				// TO DO: Process custom data here!
				// Parse data and format it for service invoke
				// ……….

			} // End of WHILE LOOP
		}
		catch (IOException e)
		{
			// Handle I/O Exception here if any
```

```
            }
            catch (Exception e)
            {
                  Handle Protocol Exception here if any
                  // close the connection on error
                  try
                  {
                        close();
                  }
                  catch(IOException ex2) {}
            }
            finally
            {
                  // close the connection and all streams
                  try {
                        if (in != null)
                              in.close();

                        if (out != null)
                              out.close();

                        if (is != null)
                              is.close();
                        if (os != null)
                              os.close();

                        if (!isClosed())
                              close();
                  }
                  catch (Exception e) {}
            }
      }
```

Known Issues and Limitations

The SimpleSocket package extends the core Integration Server functionality by providing new port listener. Therefore, some of the classes are locked by IS and reloading this package does not completely refresh all related classes. In order to update or change any new protocols and reload this package the Integration Server must be restarted.

The SimpleSocket package is not cluster aware and does not provide load balancing port or other features of Integration Server software cluster. If clustering is required for solution then external load balancer must be used to direct traffic for multiple socket servers in a cluster.

SimpleSocket Performance

This section provides some information on socket performance and scalability on different platforms. Socket performance is strongly dependent on byte size transferred, network speed and

somewhat on general system configuration and load. This relationship is shown details by the tests results presented below.

Several test were performed to validate and test socket server performance on SUN Solaris and Windows platforms

System Description

Hardware environment

The hardware description that was used during testing is described below:

Sun Solaris Test System:

Model	**SUNW Sun-Fire-480R** ▪ SunOS 5.8 Generic_108528-19 ▪ 64-bit sparcv9 ▪ 4 UltraSPARC-III CPU 900MHz ▪ Physical Memory 8192 MB	**SUNW Sun-Fire-480R** ▪ SunOS 5.8 Generic_108528-19 ▪ 64-bit sparcv9 ▪ 4 UltraSPARC-III CPU 900MHz ▪ Physical Memory 8192 MB
Role	▪ Host all webMethods server components IS/Broker. ▪ JDK and development environment	▪ Host test java Socket client, IS libraries and all dependencies

Windows 2000 Server Test System:

Model	**Compaq-Proliant** ▪ Win2000 Server SP4 ▪ 2 Intel Pentium III CPU 1400MHz ▪ Physical Memory 2GB	**Compaq-Proliant** ▪ Win2000 Server SP4 ▪ 2 Intel Pentium III CPU 1400MHz ▪ Physical Memory 2GB
Role	▪ Host all webMethods server components IS/Broker. ▪ JDK and development environment	▪ Host test java Socket client, IS libraries and all dependencies

Software environment

Components

The following software components were used on the development and test systems:

- webMethods Integration Server, version 6.1.
    - JVM used is 1.4.2 Sun

- Test Clients, Custom development multithreaded socket client program using Java. .
- The JVM used is SUN JVM version 1.4.2_04.
- Test Flow service. A simple IS flow service was developed to perform the following tasks:
    - Accept set of input parameters.
    - Mapping of input parameters (or documents) to the output parameters /documents.
    - Return output data to the client

Environment Changes

The environment settings for testing were largely left as default. The latest JDK was added to all systems. On the test environment used for client SUN JDK 1.4.2_04 has been installed. The Integration Server settings were changed as described below.

Integration Server

Parameter	Default value	New Value(s)	Explanation
-ms	128M	512M	Sets the initial amount of Java heap space for the JVM hosting IS. Given the number of concurrent processes, the default setting would not be enough.
-mx	256M	1024M	Same as above, but for the maximum heap space.
Threads	75	100-400	Controls the initial and maximum value of threads that IS will allocate to run Flow services.

Test Plan (multithreading)

Test plan included a multithreading test case for a sample data. This test shows the usability and stability of the Socket Server in multithreaded program as well as some performance benchmarks. All test start with single process executing constant number of invokes. With each test case the number of threads created per process is increasing and number of invokes per thread stay constant.

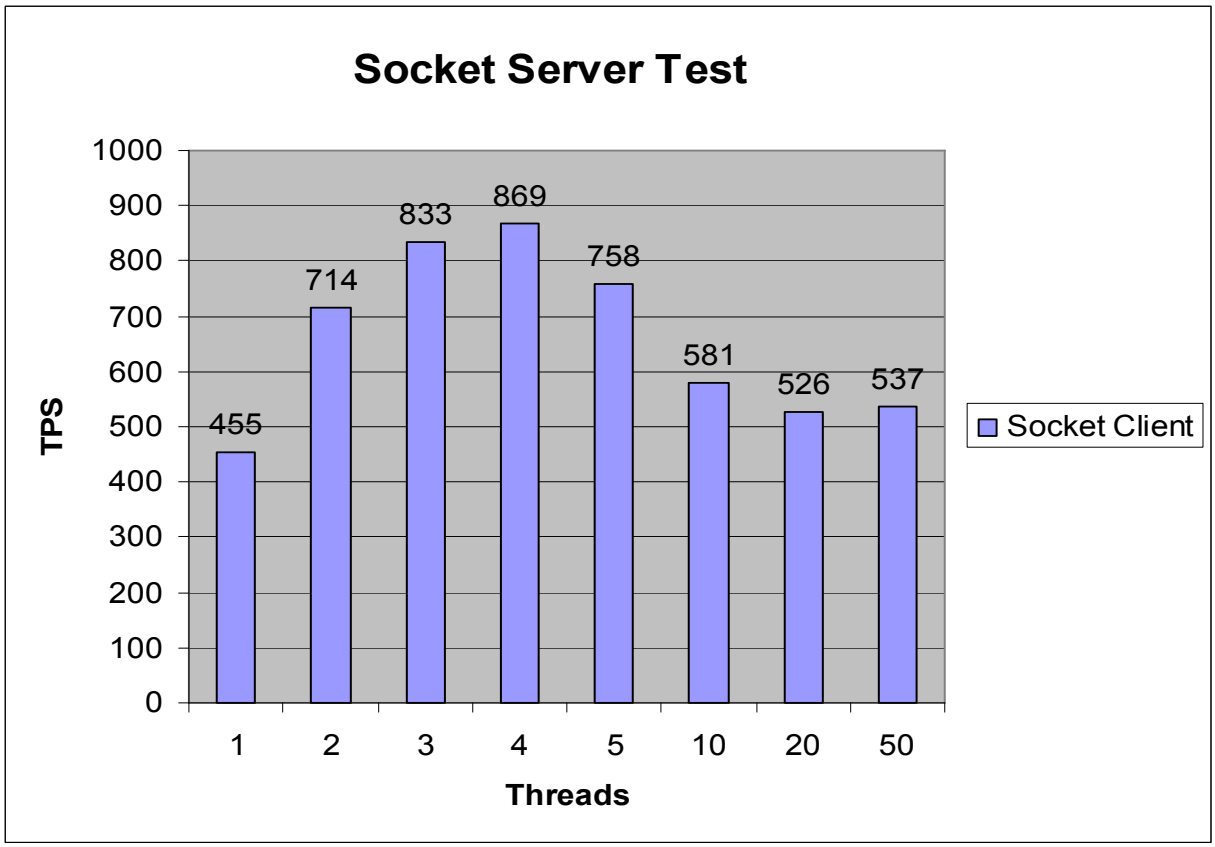
Test 1 – 1 process 1 thread and 5000 invokes
Test 2 – 1 process 2 threads and 5000 invokes
Test 3 – 1 process 3 threads and 5000 invokes
Test 4 – 1 process 4 threads and 5000 invokes
Test 5 – 1 process 5 threads and 5000 invokes
Test 6 – 1 process 10 threads and 5000 invokes
Test 7 – 1 process 20 threads and 5000 invokes

These tests were performed with small and large data samples of approximately 352 and 54519 bytes for comparison. The socket performance is dependent on network bandwidth and speed. Therefore the amount for data transported for each invoke will directly affect socket performance as shown in tests below.

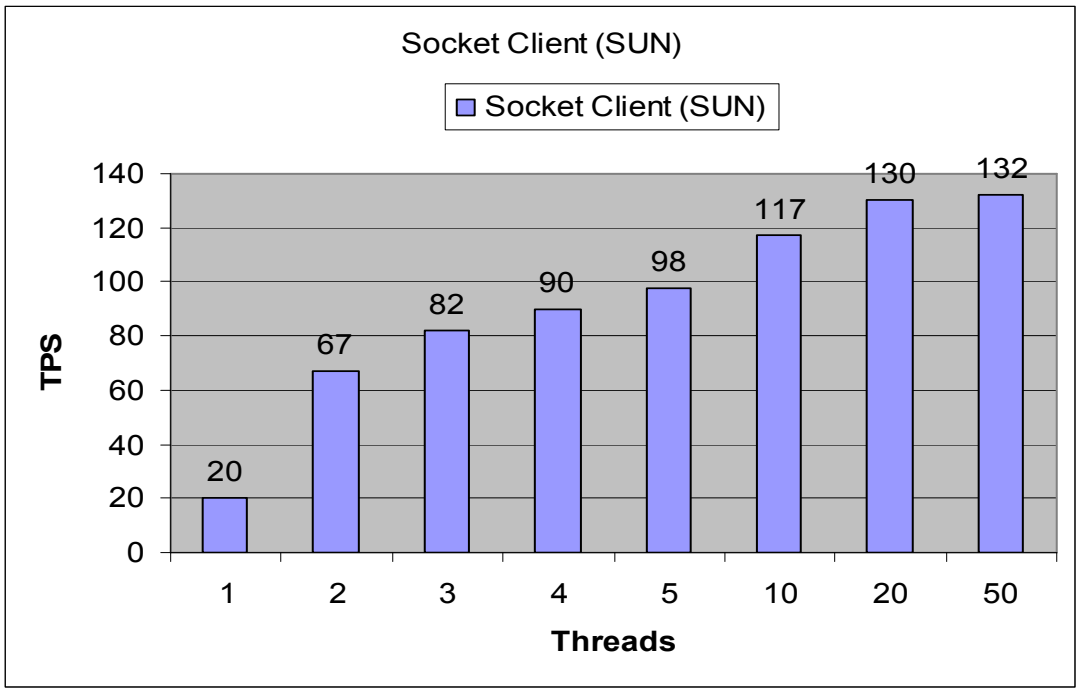
Test Results Sun Solaris platform

These results are based on testing Socket server and multithreaded client using JVM 1.4.2_04 on SUN Solaris SunFire 480 test system. The performance was measured in units of Transactions Per second (TPS) – how fast can a client invoke a service on IS and get response back.

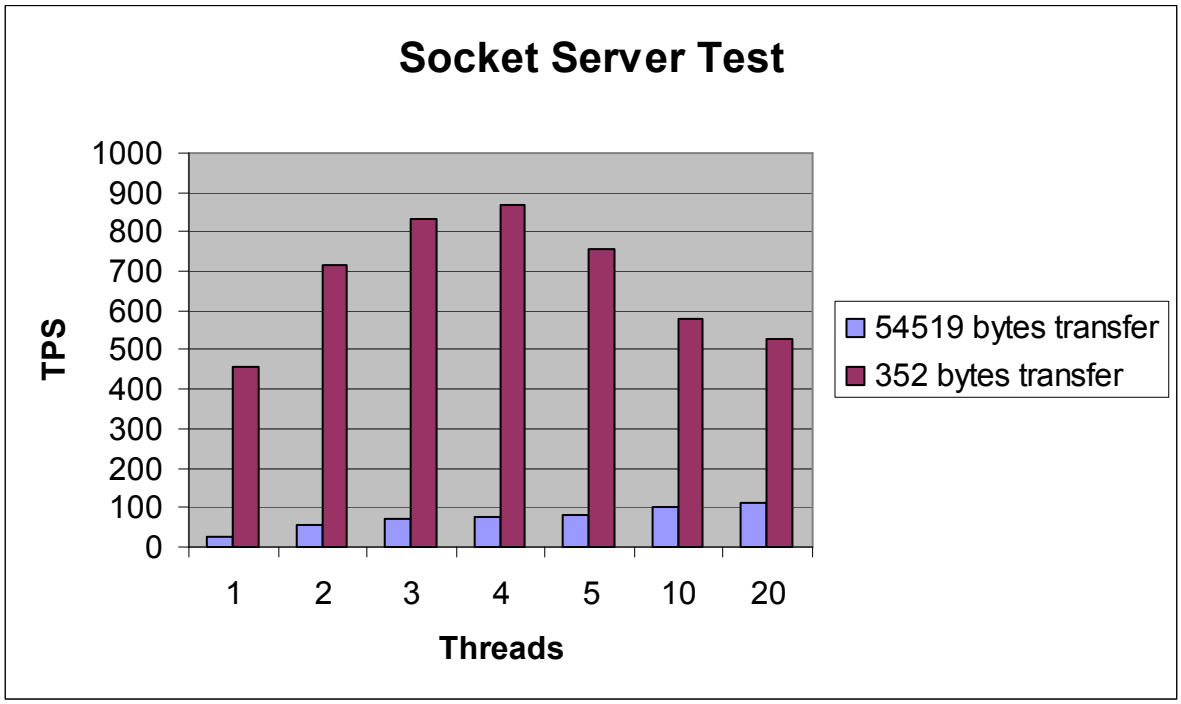
Small Data Sample 352 bytes



Large Data Sample 54519 bytes



Comparison of Large and Small Data Sample



Test Results Windows 2000 platform

These results are based on testing Socket server and multithreaded client using JVM 1.4.2 on Intel Pentium III Windows 2000 test system. The performance was measured in units of Transactions Per second (TPS) – how fast can a client invoke a service on IS and get response back.

Large Data Sample 54519 bytes