Understanding the Performance of a Web Service
Due: Monday January 17. CS514 2003SP, HW3

The purpose of this assignment is to build basic intuition about performance for Web Services. The basic assignment is so easy that we have some extra credit options for people who want more of a challenge. But the basic one is all you are required to do.

You have our permission to ask for help from a friend or from someone else in the lab if you get stuck. However, try to actually "do" the assignment yourself. Your friend can show you exactly how to do things, but do the typing yourself and make sure you understand what you are doing at each step.

Step 1: Pick one of (a) or (b). For extra credit, do both parts! However, you should then do part b on the same PC where you did part (a) using C and the winsock interface. (This is so that you can compare the results from the two programs).

a) Using Visual Studio with C#, build a "hello world" Web Service application by following the demo instructions you can find online in the ASP.NET demos and examples documentation, under "help". Set it up so that the client and the server run on the same machine (``localhost''). If you prefer to use Java and J2EE or C and CORBA, that would also be acceptable – but you need to have access to the platform. We can't help on that.

b) On a machine of your choice, build a hand-crafted "hello world" application in C. The server will open a socket and then loop calling "rcvmsg" and then invoking the hello_world(arg) procedure, then create a message with the result it returns, then reply to the sender using sendmsg(). The client will be pretty similar. It creates a socket, uses sendmsg() to send a message to the server (on the same machine, hence its own IP address), then calls rcvmsg() to receive a reply.

Notes: If you want extra credit, you should do this on the same model of PC as you used for part a. Also, if you were to use Visual Studio for .NET from C, you could probably get it to generate a simple hello-world Web Setvice in C. But *don't do that* because our goal in the comparison used for this extra credit is to put a hand-coded ultra fast implementation next to a standard out of the box Web Service coded per the recommendations. If you don't care about extra credit, this is not such an issue.

Step 2: Evaluate the performance of your solution.

We want you to measure the amount of time it takes to do a procedure call to your hello-world service and to get a response back, using various sizes of arguments and various kinds of arguments. You will then put this data into a small spreadsheet, ask excel to make a graph, and hand in the graph.

Even a "slow" procedure call is actually pretty fast and the interface for measuring time on Windows or Linux isn't always very accurate – it can be trusted for seconds but not milliseconds. Accordingly you will probably need to measure the elapsed time needed to perform many hello-world calls (perhaps hundreds or even thousands – enough to run for perhaps 30 seconds). Then compute the average elapsed time per-call.

What to measure.
Measure the round-trip call time for hello-world with arguments consisting of byte-strings of length 0, 1000, 2000, … 10,000 bytes. Have the server reply by echoing the same string it received. Next measure the exact same thing, but now using arguments that are double-precision floating point numbers. Figure out how many bytes such a number requires inside the computer and make your vector just the right length so that the length, in bytes, is the same as for the first set of tests.

Finally, measure the exact same thing, but now using arguments that are vectors of "employee HR records". An "employee HR record" should contain the employee name, age, social-security number, work phone-number, home phone-number, address and a GIF file containing the person's picture. Obviously, you can use random junk for the actual contents of these fields, but you should declare them correctly so that C# (or whatever you use) is seeing the "right" data types! Experiment with sizes until your employee HR record apparently occupies exactly 1000 bytes in the sender's machine, then use a vector of 0, 1, … 10 HR records in your calls to hello-world.

In option b, the hand-coded version in C, you can use the sender's binary representation of the data as your "in the message" representation. Of course, for option a, the representation will be picked automatically by .NET and should be a SOAP XML representation.

What to hand in
Hand in a cover page with your name and ID number. Explain which option you evaluated and describe the machine on which you ran the test (e.g. Pentium P3 running at 300Mhz with 200MB of main memory) and a line-graph of performance obtained with each of the three cases shown on it (if this is just ugly or impossible, 3 graphs will be ok with us). The y axis should be "round-trip time, in milliseconds" and the x-axis should be "argument size, in bytes, as measured by the sender".

How to get extra credit
Compare the performance of your .NET solution with the performance of your hand-coded C version on the same platform.

For fun, but not for even more extra credit.
Run the exact same program but now put the client on a different machine than the server. You'll need the server's machine name to connect to it, of course, and there may be problems doing this if IIS isn't set up to allow remote access, so we can't promise that this is feasible right now in the CSUG lab, although Rimon is working on the issue. In the materials you hand in, tell us what network hardware is being used.