CALT-68-2200
SLAC-PUB-7988
November 1998

CORBA Evaluations for the *BABAR* Online System[a]

S. Yang[†] and T.Glanzman[††]

[†]Physics Department, Caltech, Pasadena, CA 91125
[††]Stanford Linear Accelerator Center, Stanford University, Stanford, CA 94309

For the *BABAR* Computing Group[b]

Abstract

The Common Object Request Broker Architecture (CORBA) is a software system to deal with distributed object computing. The release of CORBA version 2, and real implementations from numerous vendors (both freeware and payware) have made its use very attractive for interprocess and interprocessor communication within an object-oriented software system. A number of object request brokers (ORBs) were evaluated for possible use within the *BABAR* Online system. Given an expectation for a reasonable level of performance within the Online system, it was essential to characterize the behaviour and test the response of these products prior to their adoption. This paper summarizes the results of a systematic performance study of six ORB products. The products tested include: Visibroker, Orbix, DAIS, Omnibroker, OmniORB2, and TAO. Performance results of ORB products, including a test of TCP/IP sockets, are compared. These tests resulted in the adoption of the TAO ORB for use within the *BABAR* Online system.

[a]Contributed to the *International Conference on Computing in High Energy Physics (CHEP 98)*, Chicago, IL, August 31 - September 4, 1998.
[b]Work supported in part by Department of Energy grant DE-FG-03-92ER40701 and contract DE-AC03-76SF00515.

1 Introduction

The Common Object Request Broker Architecture (CORBA) is an industry standard way for distributed objects and clients to interact with each other. The driving force behind CORBA is the Object Management Group (OMG) [1], which was formed in 1989 to promote the interoperability of object-oriented software systems.

The CORBA was designed to allow intelligent software components to discover each other and interoperate on an object bus called Object Request Broker (ORB). The ORB is a server application that functions like a network switch. Any client object can make a request to a local or remote server object through an ORB. The server also responds through it. Being language-independent, CORBA introduces an Interface Definition Language (IDL) to describe the content and capabilities of objects without implementing their details. An IDL compiler then creates the *C++* or other language stub files.

The *BABAR* Online system [2] is a collection of cooperating processes. It is also object-oriented. Thus, the availability of CORBA made its use very attractive for interprocess and interprocessor communications. Six different CORBA products were tested which resulted in the adoption of TAO for use within the *BABAR* Online system.

2 Performance tests

We used six different *C++* ORBs to measure performance: `OMNIBROKER` 2.0.1 [3], `omniORB2` 2.2.0 [4], and `TAO` 0.0.10 and 0.2.3 [5], `VisiBroker` 3.0 [6], `orbix` 2.3c [7], and `DAIS` 3.0 [8]. These tests were geared toward addressing the types of communications anticipated should the CORBA technology be adopted.

We used two Solaris 2.5.1 UltraSPARC machines (`Percheron` is 246MHz CPU Enterprise 6000 and `Charger` is 168MHz CPU Ultra 2) connected by switched 100Mbps Fast Ethernet. A simple test was constructed: Client sends data and server receives data, then returns nothing (oneway), void, or the original input data. The oneway test is similar to a return of void except that the return of void requires a handshake, while oneway does not. With some effort all tested ORBs were interoperable with each other, *e.g.* An Orbix client could communicate with a VisiBroker server. We performed the following benchmark performance tests:

- Type dependence. Transmit various types of CORBA sequences (char, float, long, and double) and corresponding struct sequences (*e.g.* struct with only one char *etc.*).
- Structure element type dependence. Transmit structs of the same size and with same number of elements, but with different types of elements: four longs (homogeneous), and one double, one float, and two shorts (heterogeneous)

- Structure element number dependence. Transmit structs of the same size but with differing content, one struct with one double *vs.* and one struct with two floats

3 Results

The lessons we learned from the benchmark tests are summarized below. Results using `VisiBroker` 3.0 appear in figure 1.

- The sending of structs (vs. sequences) incurs an extra overhead.
- The relative struct sending overhead gets larger as the struct size gets smaller. For example, the overhead ratio between single char struct and single double struct is about 3.3.
- The struct overhead gets larger as the number of elements in the struct increases. For example, the overhead ratio between an eight char struct and a single double struct is about 5.
- There is no significant performance difference between different sequence types.
- There is no significant performance difference between structs containing identical vs dissimilar elements of the same size.
- The best performance is generally achieved when the data payload is greater than 1kB.

The comparison among different ORB vendors is summarized below:

- `Visibroker 3.0` versus `OMNIBROKER 2.0.1` and `omniORB2 2.2.0`: `Visibroker 3.0` gave the best results. The performance between `OMNIBROKER 2.0.1` and `omniORB2 2.2.0` is similar. Within a factor of two, the performance of all three vendors is similar and the throughput plateau is above 2Mbps as shown in figure 2.
- `Visibroker 3.0` versus `orbix 2.3c` and `DAIS 3.0`: `Visibroker 3.0` gave the best results. `DAIS 3.0` gave terrible performance at payload above 10kB as shown in figure 3.
- `Visibroker 3.0` versus `TAO`: `TAO 0.2.3` gave the best results for sending char sequence. For sending structs, `TAO 0.2.3` gave the best results at payload below 1kB and comparable to `Visibroker 3.0` at payload above 1kB as shown in figure 4.

We also compared ordinary TCP/IP socket communication versus `VisiBroker` 3.0 throughput performance as shown in figure 5. As indicated by the plots, the performance difference between sockets and CORBA is significant for small message sizes. This difference narrows considerably as the message sizes gets larger.

4 Conclusions

We have observed significant throughput differences between various ORB implementations and between CORBA and ordinary socket communications. A combination of throughput and latency issues have prevented us from adopting CORBA for event transport within the *BABAR* online system. However, CORBA was found to be sufficiently performant for use within other areas of the system. Coupled with its flexible higher level services and language independence, it has been adopted for appropriate applications. The TAO ORB was selected because of its performance and real-time orientation, vigorous developer group and, of course, it is free.

References

[1] OMG: http://www.omg.org/

[2] *BABAR* : http://www.slac.stanford.edu:/BF/doc/www/bfHome.html

[3] OMNIBROKER: http://www.ooc.com/ob.html

[4] omniORB2: http://www.orl.co.uk/omniORB/omniORB.html

[5] TAO: http://siesta.cs.wustl.edu/ schmidt/TAO.html

[6] VisiBroker: http://www.inprise.com/visibroker/

[7] orbix: http://www.iona.com/products/orbix/

[8] DAIS: http://www.daisorb.com/sbs/daismenu.html

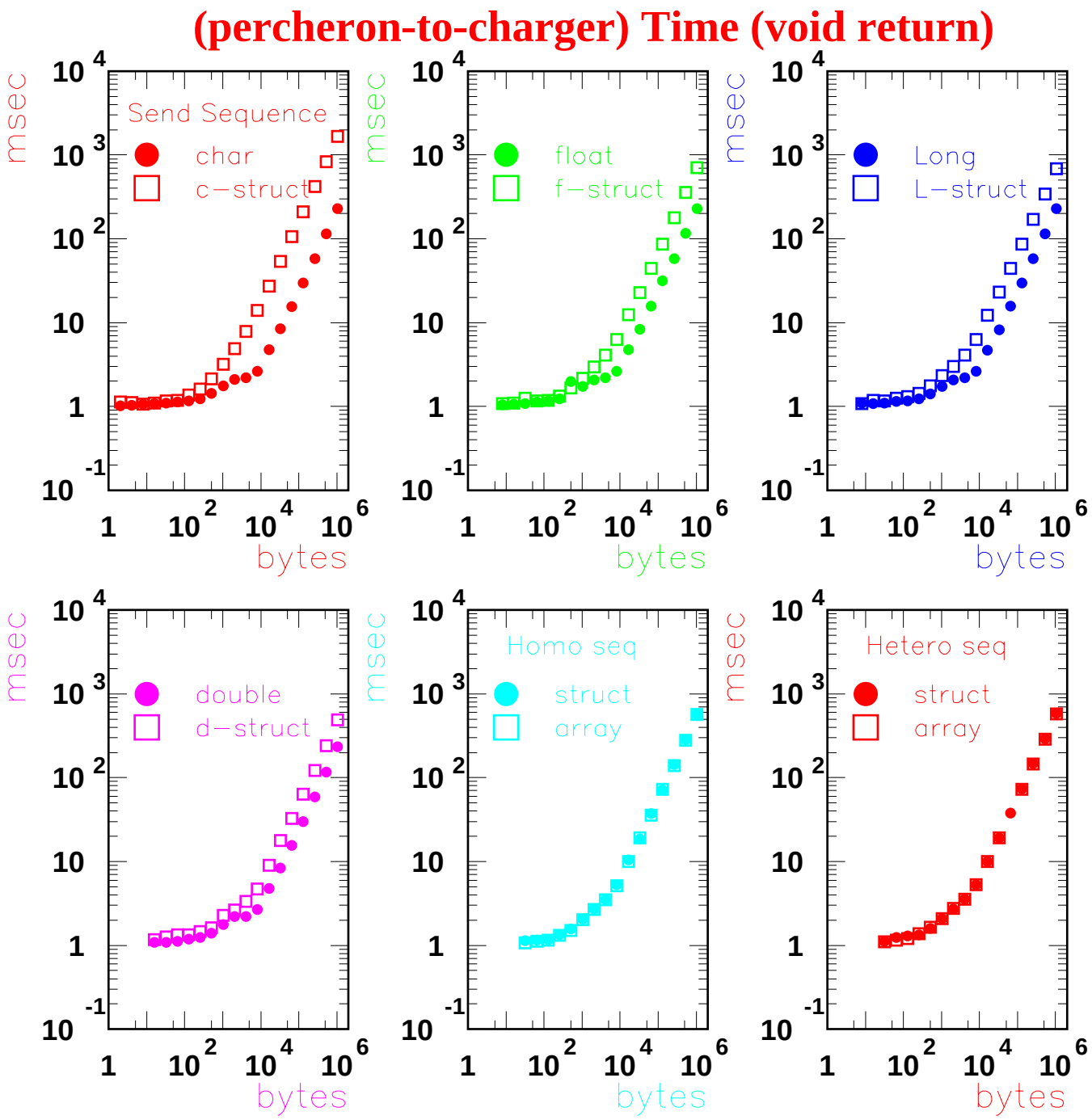

Figure 1: Transfer time in msecs as a function of payload in bytes

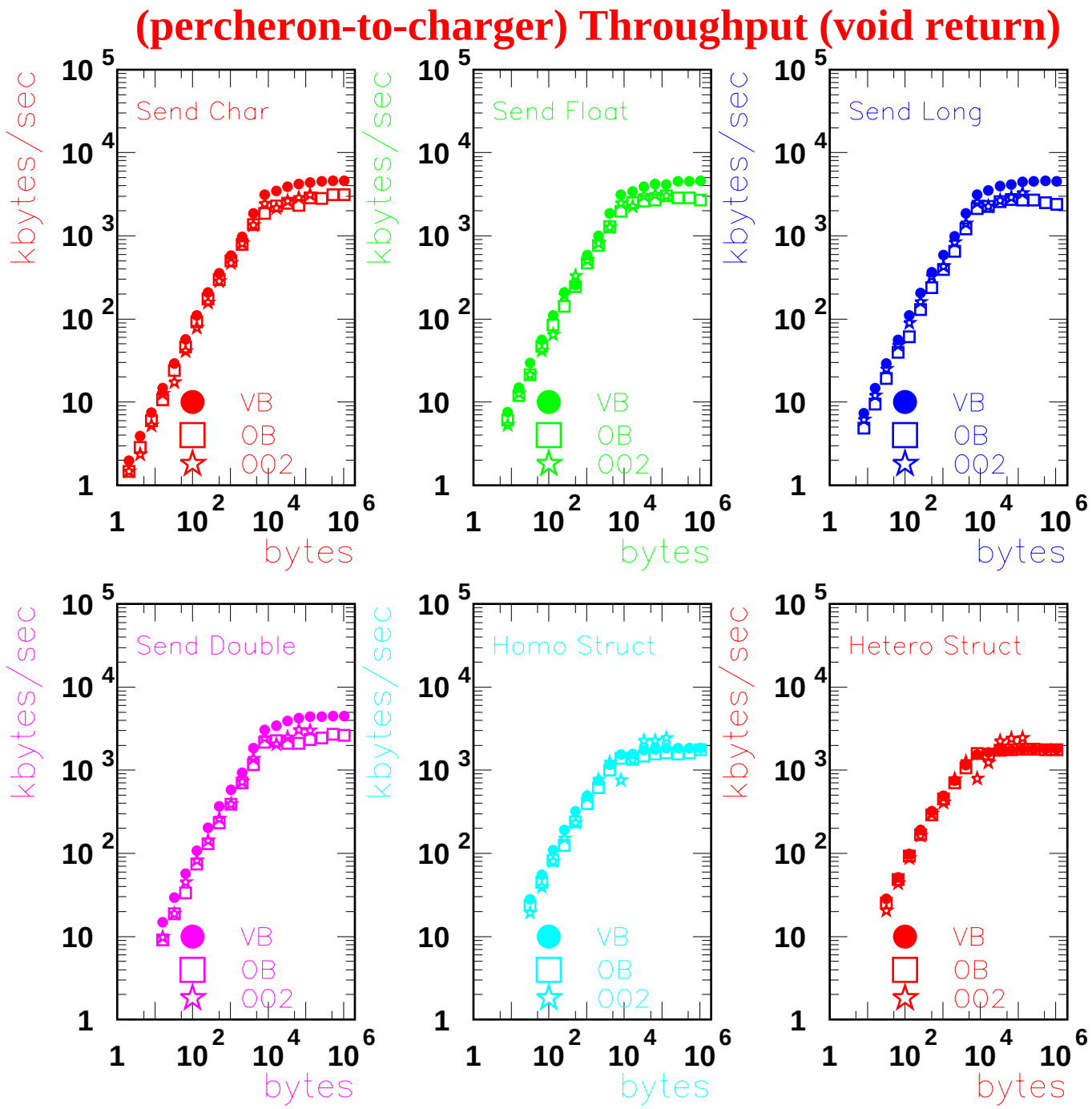

Figure 2: Throughput performance of three different ORB vendors: `VisiBroker` 3.0 (VB), `OMNIBROKER` 2.0.1 (OB), and `omniORB2` 2.2.0 (OO2).

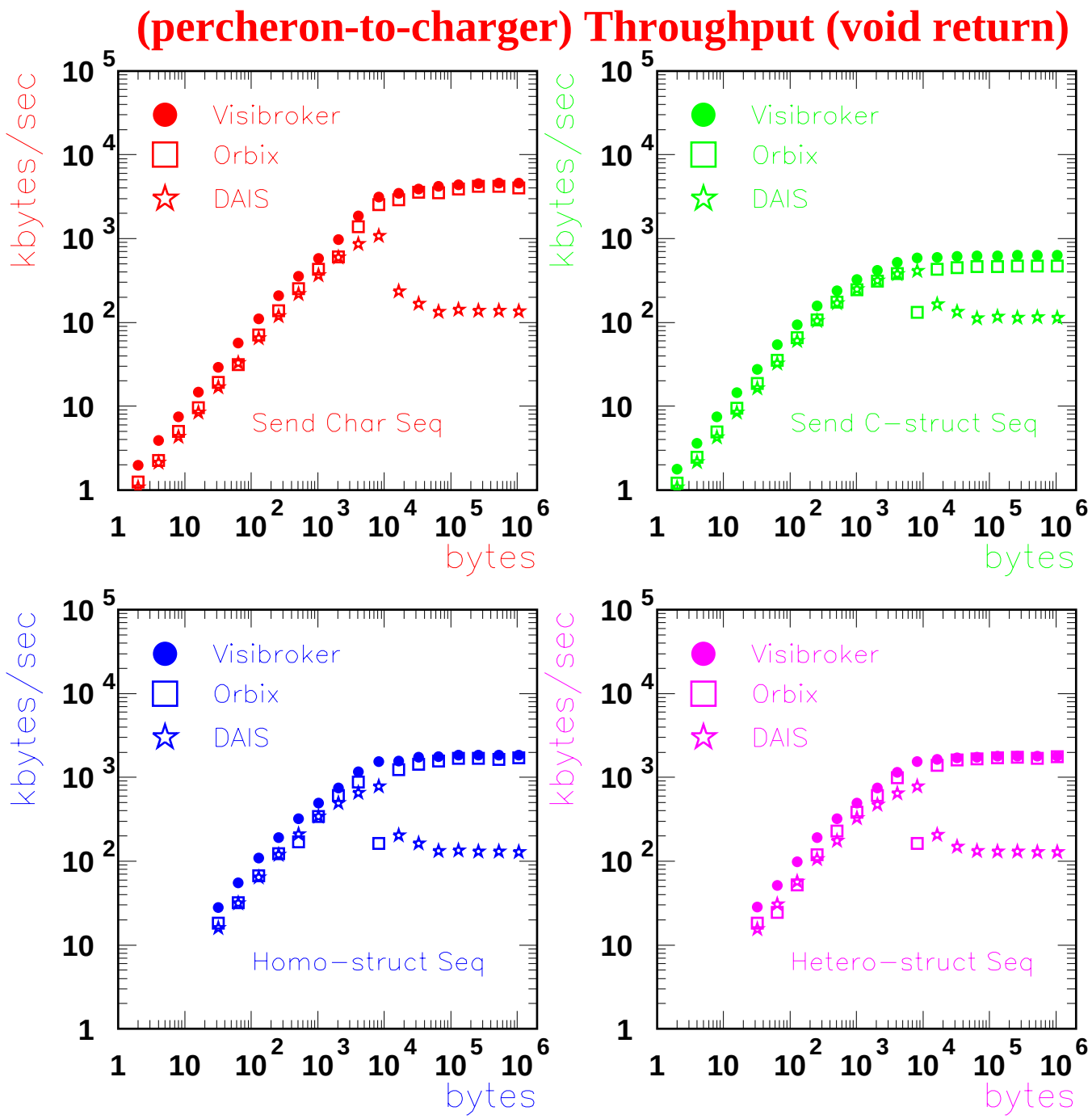

Figure 3: Throughput performance of three different ORB vendors: `VisiBroker` 3.0, `orbix` 2.3c, and `DAIS` 3.0.

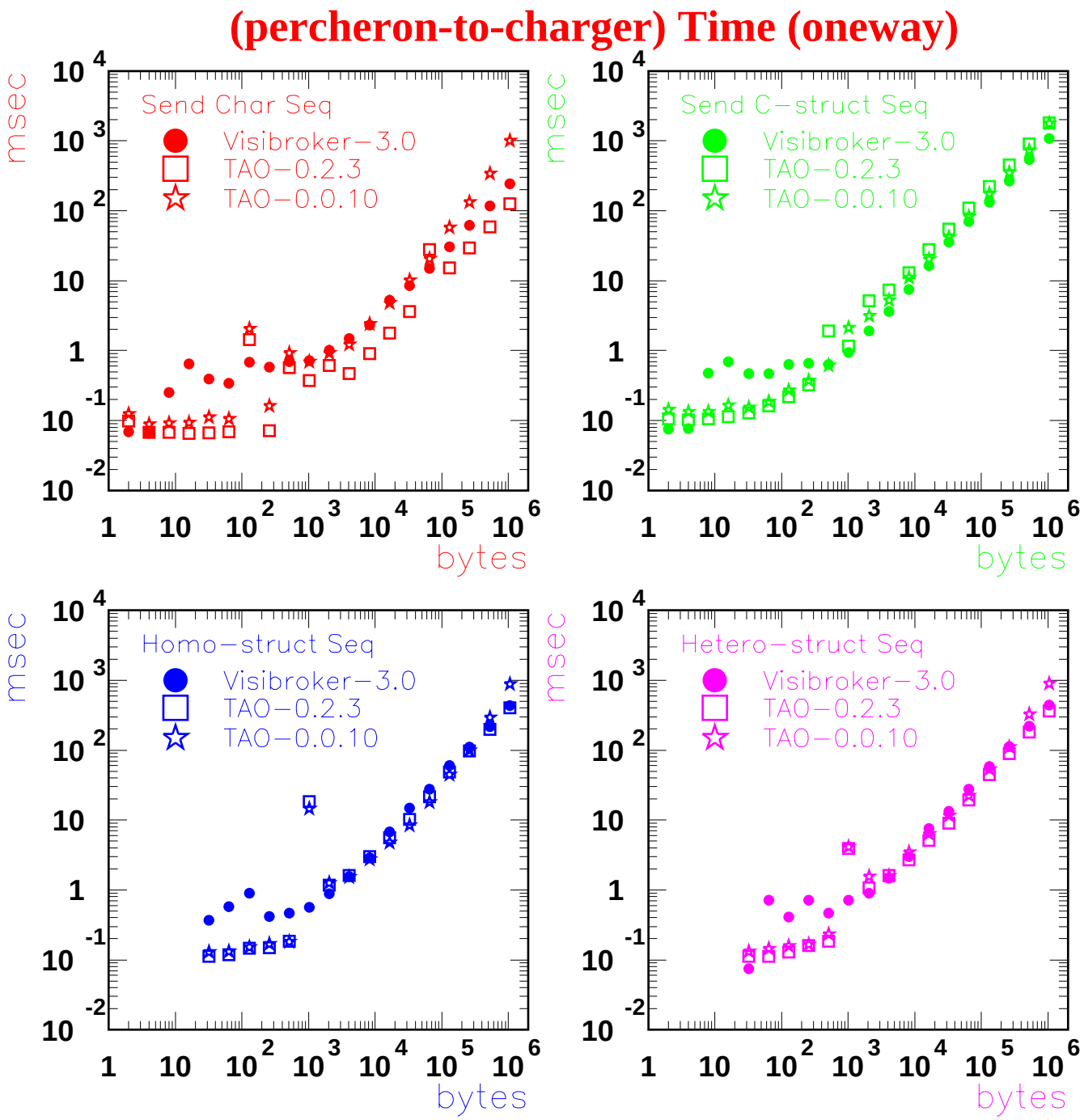

Figure 4: Throughput performance of two different ORB vendors: `VisiBroker` and two different `TAO` version. `TAO` improves the performance over time.

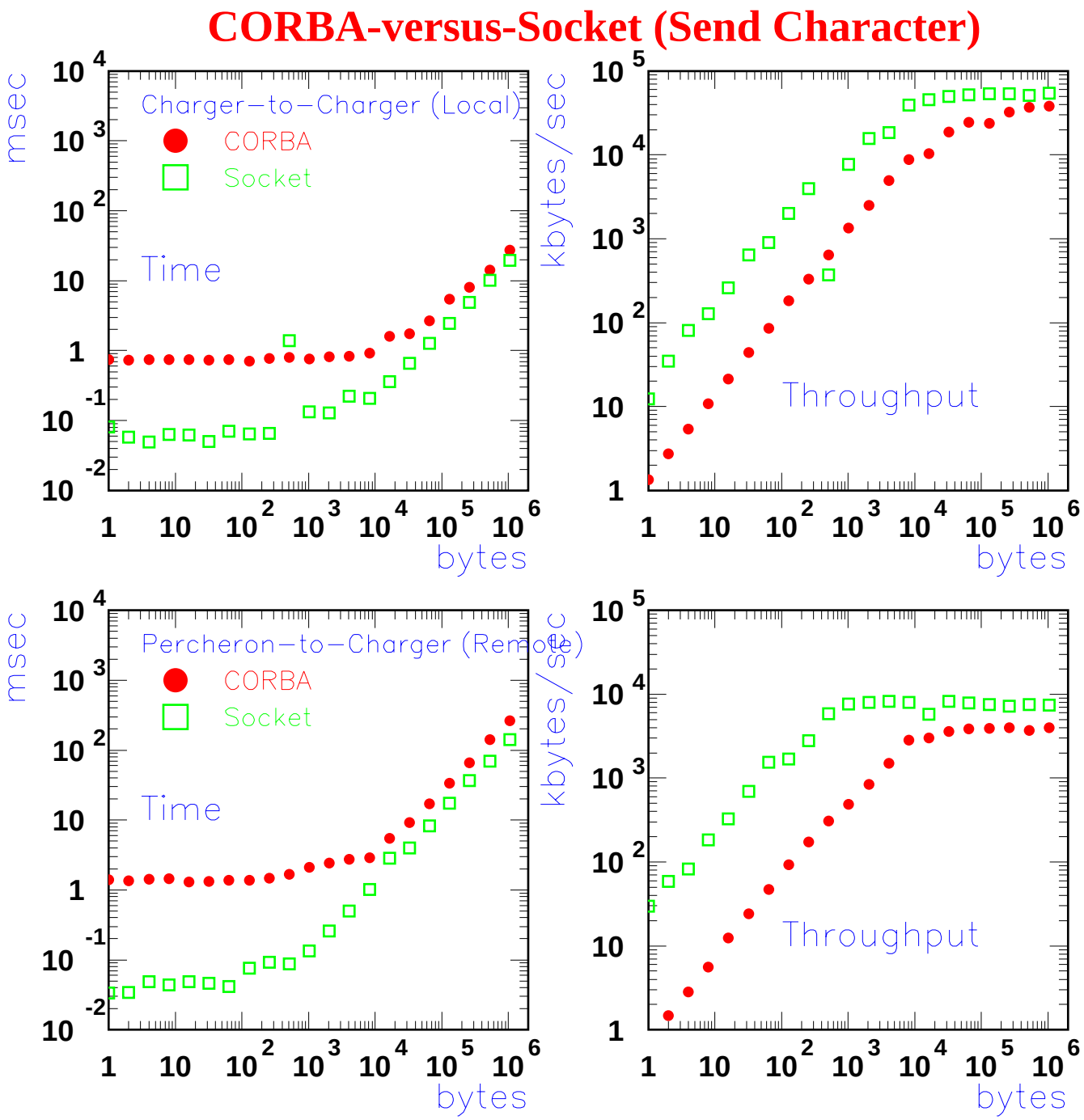

Figure 5: TCP/IP socket versus `VisiBroker` 3.0 performance. The top two plots are for local transfer and the bottom two plots are for remote transaction.