# Comparisons of Distributed Operating System Performance Using the WPI Benchmark Suite *

David Finkel Robert E. Kinicki Jonas A. Lehmann
Joseph CaraDonna

Department of Computer Science
Worcester Polytechnic Institute
Worcester, MA 01609
mach@cs.wpi.edu

**Abstract**

The Worcester Polytechnic Institute Mach Research Group has developed a series of benchmark programs, the WPI Benchmark Suite (WBS), designed to evaluate the performance of Unix-like operating systems. This paper presents performance results produced by running programs from the WBS on HP 386 PCs, HP 486 PCs, and Sun 3/60 workstations. The analysis of these benchmark runs includes comparisons of Mach 2.5, Mach 3.0, and SUNOS 4.1.1 running on identical hardware platforms. The focus of this paper is distributed benchmarks designed to evaluate the effectiveness of different operating system mechanisms for distributed applications. The results identify strengths and weaknesses in the Mach 2.5 and Mach 3.0 operating systems.

Worcester Polytechnic Institute Technical Report WPI-CS-TR-92-2

*This research was supported by a grant from the Research Institute of the Open Software Foundation.

# 1 Introduction

The Mach operating system [SIL91] currently exists in two principal forms: Mach 2.5 is a conventional macro-kernel design, and Mach 3.0 is micro-kernel based. The WPI Mach Research Group undertook a project to compare the performance of these two philosophically different versions of Mach with existing versions of Unix. To this end, a set of programs, the WPI Benchmark Suite (WBS), was developed. This concept of creating benchmarks designed to compare operating system performance differs from the intent of most other available benchmark programs and commercial suites which are designed to test hardware speed and involve few operating system services.

The design philosophy of our benchmark development is to have a two-tiered set of programs where the major programs are high-level synthetic benchmarks designed to reflect the usage of operating system services found in user application programs, and the low-level benchmarks consist of individual system functions which can be used to isolate and identify specific weaknesses in operating system designs.

This paper reports on the results of running the distributed application programs in WBS on HP 386 PCs, HP 486 PCs and Sun 3/60 workstations. The results are used to evaluate the differences in the behavior of Mach 2.5 and Mach 3.0 when handling user applications in a distributed environment.

Section 2 briefly puts the WBS in context with previous benchmarking efforts. Section 3 discusses methods used to develop the benchmarks and Section 4 describes the individual programs in the benchmark suite. The results from the distributed benchmarks are given and analyzed in Section 5 with conclusions presented in the last section.

# 2 Previous Mach Benchmarks

Benchmark results for the Mach Operating System have appeared in [BLA89], [FOR89], [GOL90], and [TEV87]. Generally these performance studies used low-level benchmarks, repeatedly exercising a single system call or system service. While such low-level benchmarks are important to system developers to test the efficiency of their implementations, such results do not lend themselves for interpretation by users concerned with the performance of high-level applications running on Mach.

We identified a large number of benchmarking programs and suites, for example [SPE90], [SPE91] [CUR76], [DON87], [SMI90], and [WEI84]. For the most part, these benchmarks emphasize CPU intensive applications, and did not specifically target operating system performance. An especially thorough set of benchmarks for Unix systems is given in [ACE89]. This report describes ten benchmark suites, and gives the results of running them on 47 Unix systems. Again, most of the benchmarks are either CPU-bound, or low-level tests of system functions, and do not significantly test fundamental operating system functions.

# 3 Developing User-Level Benchmarks

In order to specifically evaluate the performance of operating systems, we set out to create benchmarks that make extensive use of significant operating system services in a mix reflecting the usage by actual user applications. Two possible approaches to this task were collecting actual user code, as in the SPEC 1.0 benchmarks [SPE90], or writing synthetic programs with the desired properties.

We adopted the synthetic programs approach for several reasons. First, writing synthetic programs allowed us to understand in detail what system services were used in the benchmark programs. This in turn enabled us to create low-level benchmarks to test individual system functions to understand the reasons for the differences between results on different systems, and to provide guidance to system developers about areas of the system needing improvement. Second, synthetic programs permitted us to parametrize the benchmarks, to allow the same benchmark program to run on small scale and large scale systems (with different parameter settings). Third, by using a set of synthetic programs, a suite of benchmarks could collectively cover the entire range of important systems services.

We used several methods to ensure that our synthetic benchmark programs reflected the usage pattern of system services representative of actual user application programs. One approach was to run actual user applications under the control of a profiler, such as *gprof* [GRA82]. This allowed us to identify the particular system calls used by the program, and the number of times each call was made in the application. We also used statistical utilities, such as *vmstat* and *iostat*, to track the use of other system resources. This

then gave us some guidelines in constructing our programs and in tuning them to match the resource utilizations of the original programs.

We also examined the source code of user applications of interest, and identified the key system calls. This, together with the information provided by the statistical utilities, allowed us to construct representative benchmarks.

The first release of the WBS uses only Unix system calls, and no Mach-specific system calls.. This allows the benchmarks to be used directly on non-Mach Unix systems. We are currently working on re-writing some of our benchmarks to use Mach-specific system services, to understand the performance implications of using these services.

# 4 The WPI Benchmark Suite

The following is a brief explanation of the six high-level programs in the WPI Benchmark Suite and a short discussion of a set of low-level interprocess communication (IPC) tests. The five user-level programs with an S prefix are truly synthetic programs, while Jigsaw is a test program designed to utilize specific system services.

## 4.1 Scomp

This program creates a mix of Unix system calls which are designed to mimic system resource usage of gcc compiling gcc. Data was collected by using gprof to monitor the procedure calls used when gcc compiles itself. From the procedure call information, Scomp was synthesized to recreate the structure of gcc to some extent and to issue Unix system calls in a pattern similar to gcc.

## 4.2 Sdbase

This client-server database benchmark uses TCP/IP sockets to communicate between a single server and multiple clients. The system is composed of a concurrent database server, a number of client processes, a database generation program, a large database file and programs to analyze server and client performance.

The requested services include reading a random record from the database, modifying a record and appending new records. The client activity is based on a job mix used in the Byte Magazine benchmarks [SMI90], [SMI91].

## 4.3 Sdump

Modelled after the Unix dump program, this benchmark reads a set of one Mbyte files from a directory representing a file system and transfers the data to a process emulating a tape device. The transport of the data from the reading process to the writing process is done via Unix pipes. The writing process can either dump the merged file to a null device or to disk. The number of files dumped is a run-time parameter.

## 4.4 Sftp

By emulating an FTP transfer, Sftp is designed to show transmission rate performance when transferring large files with various buffer sizes. The host machine participating in the TCP/IP transfer runs a server background task which responds to remote client requests for file transfers.

## 4.5 SXipc

SXipc emulates network traffic between an X server and a set of X clients. Utilizing eight different X client types measured by Droms and Dyksen [DRO90], SXipc is a script-driven program which allows for a large number of local and remote clients to issue requests to the X server. This program currently characterizes the communication behavior of X. Efforts to include the I/O activity associated with X windows or to incorporate the CPU activity of servicing window requests are not included in the current version of the benchmark.

## 4.6 Jigsaw

Jigsaw solves a mathematical model of a jigsaw puzzle [GRE86] where the four sides of a puzzle tile have a recognizable relation with the sides of neighbor tiles in the solved puzzle. The benchmark builds a puzzle, scrambles tiles, and records the time required to solve the jumbled puzzle. Puzzle size

is variable. With tile sizes of 1 or 4 kbytes, this benchmark is targeted at studying memory allocation and paging behavior.

### 4.7 Low-Level IPC Benchmarks

While working on the SXipc and Sdbase application level benchmarks, we developed six low-level interprocess communication benchmarks, each of which focuses on a specific mechanism for interprocess communication in either Unix or Mach. Each of these benchmarks provides a functionally equivalent communications capability, but each is implemented using a different IPC mechanism. The results from the low-level IPC tests could then be used to focus on one aspect of the operating system services and help determine how the communication primitive performance impacted the larger benchmarks. The six IPC mechanisms are: pipes, message passing, sockets and shared memory in Unix, and message passing and shared memory (using threads) in Mach. The results include local and distributed uses of the IPC mechanisms over an Ethernet. The detailed results of these IPC benchmarks can be found in [RAO91]. Discussion of a few of these results which are germane to the analysis of the distributed benchmarks is given at the end of the next section.

## 5 Performance Results

We have previously reported preliminary results from Scomp, Sdump, and Jigsaw at the Usenix Mach Workshop [FIN90] and the OSF Micro-Kernel Design Review [KIN91]. Hence in this paper we focus on results from the three newer benchmarks Sftp, Sdbase, and SXipc. In addition, we present a discussion of the low level IPC benchmarks.

The principal results reported here were run on the following systems configurations:

- HP Vectra 486 PC, with
  - Intel 80486-25Mhz
  - 32-bit address and data busses
  - DMA data transfer rate of up to 33MB/sec

  - Extended Industry Standard Architecture (EISA)
  - 20 megabit/sec ESDI hard disk drive controller
  - 330MB ESDI Hard Disk
  - 16MB RAM
- Mach 2.5
- Mach 3.0 XMK 42

In addition, some of the tests reported were run on HP-Vectra 386 PCs running Mach 2.5 and Mach 3.0, and on Sun 3/60 workstations running the Mt. Xinu release of Mach 2.5, denoted Mach 2.6 MSD, and SunOS 4.1 For all the network tests, the machines involved in the test were running on a private network, eliminating the possibility of perturbing the test results because of extraneous network traffic. In all cases, the results shown are averages of 5 runs of the test.

A series of Sftp tests were run with a one Megabyte file being transferred between two 486 PCs. Figure 1 shows the results from these benchmark runs with the transport level buffer size varying from 64 bytes to 1 Megabyte. Note the performance measure shown is Kilobytes per second transferred, so that larger numbers indicate better performance. We see that Mach 2.5 does significantly better than Mach 3.0 at transferring 1MB files for all buffer sizes. The interesting result is that the Mach 3.0 performance is relatively constant as the buffer size changes, while Mach 2.5 varies considerably. It appears that one or more layers within the flow of data in Mach 3.0 has data structures that are fixed in size, where as in Mach 2.5 the data structures are dynamic. This difference allows Mach 2.5 to take advantage of the varying buffer size, while Mach 3.0 is unable to [JOH91]. The drop in the transfer rate at 4KB buffers is explained below in the discussion of the IPC tests.

Sdbase, a client/server benchmark, was run on all three hardware platforms (386, 486 and Sun3/60) both in local and remote modes. In the local mode, both the clients and the server run on the same machine; in remote mode, all the clients run on one machine, and the server runs on another. By measuring communications time as well as both the server and client performance, a variety of observations about these tests can be made [JOH91]. The figures for the Sdbase results show results only for the HP 486 machines

described above; the performance measure is elapsed time, so smaller measurements indicate better performance. Figure 2 shows the communications time for the Sdbase test running in remote mode. The results are consistent with the low level IPC tests and Sftp in showing that Mach 2.5 yields superior performance to Mach 3.0 in TCP/IP based communications. Server performance is shown in Figures 3 and 4. These figures show the average server time; i.e, the total elapsed time for the server divided by the number of clients. Here Mach 3.0 outperforms Mach 2.5 regardless of whether the clients are local or remote. We attribute this difference to the copy-on-write provided for shared memory in Mach 3.0, and its general lazy evaluation approach.

SXipc was run as a distributed benchmark under Mach 2.5 and Mach 3.0 on two HP486 PC's connected via Ethernet. Figure 5 presents the elapsed time for a series of tests where all the clients and the X server reside on the same machine. The clients are identical and their requests are driven by scripts characterizing X dvi client requests. The results are consistent with the other benchmarks in that Mach 2.5 does significantly better than Mach 3.0 when the primary activity is TCP/IP communication. With 20 local clients the Mach 2.5 elapsed time is less than half the Mach 3.0 elapsed time. Figure 6 shows elapsed time measurements when all clients are on one machine and the server is on a second 486 PC. On Mach 3.0 the elapsed time for 20 remote clients is more than three times the Mach 2.5 elapsed times. Note that in going from a workload of 20 local to 20 remote clients, Mach 2.5 takes advantage of the second machine and elapsed time decreases. However the inefficiencies in the Mach 3.0 communications services result in higher elapsed time for remote clients.

Figure 7 graphs the elapsed times from a series of tests where the client mix consists of an equal number of local and remote X dvi clients running under Mach 2.5 and Mach 3.0. Note an elapsed time for eight local clients is a measurement on the local machine when the server is dealing with a load of eight local and eight remote clients. Mach 2.5 continues to perform better. With this mixed workload, the Mach 3.0 remote clients get better service than the local clients because the local clients compete with the server for the CPU and the network message queues.

With Figure 8, attention is switched to the low-level IPC benchmark which employs sockets in Mach 2.6 MSD and SunOS 4.1 to send messages locally on a Sun 3/60 workstation. The graph shows that over a wide range of

message sizes in almost every case Mach takes more milliseconds per transaction than Sun's Unix. The transaction measurements are the results of averaging over 5000 transactions per test run. Because Sun's Unix allocates a limit of 4KB of memory for 32 mbuf structures, there is a significant jump in transaction time at 4097 bytes for both operating systems. This partially explains the performance drop seen in the Sftp results presented in Figure 1.

Figure 9 compares Unix and Mach using a message passing mechanism between local processes on the Sun workstation. At smaller message sizes Mach performs better, but there is a crossover point such that Unix does better above 1KB size messages. The graph stops at 2KB because that is a SunOS limit. Figure 10 presents results when shared memory is used for communication. Because Mach can use threads to accomplish this task, the time per transaction is about one third that of the Unix shared memory implementation.

Figure 11 shows a low-level comparison between Mach 2.5 and Mach 3.0 sending local messages using sockets on a 386 HP PC. This graph shows clearly that Mach 3.0 has a problem communicating through sockets. Furthermore, it shows part of the cause of the poor performance for Mach 3.0 on the Sdbase and SXipc benchmarks.

# 6 Conclusions

The WPI Benchmark Suite was developed to compare the performance of different Unix-based operating systems running on the same hardware. In this paper, we have given a brief description of the benchmark programs, and discussed performance results of the distributed benchmark programs.

The analysis of the results indicates that Mach 3.0 with XMK42 kernel from Carnegie Mellon does not perform as well as Mach 2.5 for most of the distributed PC tests. Our low-level results on Sun workstations provide some sense of comparison between SunOS and Mach. These results must be interpreted with the reminder that our objective was to run identical programs on Unix and Mach systems. Our plan is to convert some of the benchmarks to include Mach system calls which take advantage of some of the Mach mechanisms which should produce performance gains for some of the distributed benchmarks.

We believe that the major contribution of our work is the development of

the benchmarks focusing on operating system performance. Our benchmarks have been distributed to a number of researchers and system developers, and several have reported that they have found the benchmarks useful in tracking system performance.

The benchmarks are in the public domain, and are available via anonymous ftp from wpi.wpi.edu, in the benchmarks directory.

# Acknowledgements

In addition to the authors of this paper, the following individuals contributed to the development of the WPI Benchmark Suite: Aju John, Bradford B. Nichols, Somesh Rao, and Dhruve K. Shah. The authors wish to acknowledge their valuable contributions to this project.

# References

[ACE89] *Benchmarking UNIX Systems* , ACE (Associated Computer Experts bv, Van Eeghenstraat 100, 1071 GL Amsterdam, The Netherlands, 1989.

[BLA89] D.L. Black, R.F. Rashid, D.B. Golub, C.R. Hill, and R.V. Baron, "Translation Look-aside Buffer Consistency: A Software Approach", *Digest of Papers, COMPCON Spring '89, Thirty-Fourth IEEE Computer Society International Conference: Intellectual Leverage* , (1989), 184 - 190. [ Also Carnegie Mellon University School of Computer Science Technical Report, CMU-CS-88-201].

[CUR76] H.J. Curnow and B.A. Wichmann, "A Synthetic Benchmark", *The Computer Journal*, 19 (1976), 43 - 49.

[DON87] J. Dongarra, J.L. Martin and J. Worlton, "Computer Benchmarking: Paths and Pitfalls", *IEEE Spectrum*, July 1987, 38 - 43.

[DRO90] R.E. Droms, and W.E. Dyksen, "Performance Measurements of the X Window System Communication Protocol", Tech. Rep. 90-9, Department of Computer Science, Bucknell University, March 90.

[FIN90] D. Finkel, R.E. Kinicki, A. John, B. Nichols, and S. Rao, "Developing Benchmarks to Measure the Performance of the Mach Operating System", *Proceedings of the Usenix Mach Workshop*, Oct. 90, Burlington, Vt., 83 - 100.

[FOR89] A. Forin, J. Barrera, M. Young, and R.F. Rashid, "Design, Implementation and Performance Evaluation of a Distributed Shared Memory Server for Mach", Carnegie Mellon University School of Computer Science Technical Report, CMU-CS-88-165. [Also published as "The Shared Memory Server", USENIX Winter Conference, San Diego, 1989.]

[GOL90] D. Golub, R. Dean, A. Forin, and R. Rashid, "Unix as an Application Program", *Proceedings of the USENIX Summer Conference*, June, 1990, 87 - 95.

[GRA82] S.L. Graham , P.B. Kessler, and M.K. McKusick, "gprof: A Call Graph Execution Profiler", *Proceedings of the SIGPLAN '82 Symposium on Compiler Construction*, SIGPLAN Notices, Vol. 17, No. 6 (June 1982) 120 - 126.

[GRE86] P.E. Green and R.J. Juels, "The Jigsaw Puzzle - A Distributed Performance Test", *Proceedings of the 6th International Conf. on Distributed Computing Systems*, May 19-23, 1986, 288 - 295.

[JOH91] A. John, "Performance Evaluation of Virtual Memory Management and Interprocess Communication in the Mach Operating System", Master's Thesis, Worcester Polytechnic Institute, May 1991.

[KIN91] R. Kinicki, D. Finkel, A. John, B.B. Nichols, D. Shah, and S. Rao, "Comparative Performance Measurements", OSF Microkernel Design Review, Cambridge, Ma. Feb. 1991.

[RAO91] S. Rao, "Performance Comparison of Interprocess Communication in Mach and Unix", Master's Thesis, Worcester Polytechnic Institute, May 1991.

[SIL91] A. Silberschatz, J. Peterson, and P. Galvin, *Operating Systems Concepts*, Third Edition, Addison-Wesley, 1991.

[SPE90] "Benchmark Results ", *SPEC Newsletter*, Vol. 2, No. 2, Spring 1990.

[SPE91] "SPEC SDM: System Level Benchmark Suite", *Performance Evaluation Review*, Vol. 19, No. 2, (Aug., 1991) pg. 2.

[SMI90] B. Smith, "The Byte Unix Benchmarks", *Byte*, March 1990, 273 - 277.

[SMI91] B. Smith, private communication.

[TEV87] A. Tevanian, "Architecture-Independent Virtual Memory Management for Parallel and Distributed Environments: The Mach Approach", Ph.D. Thesis, Carnegie Mellon University School of Computer Science, Dec., 1987. [Also Carnegie Mellon University School of Computer Science Technical Report, CMU-CS-88-106].

[WEI84] R.P. Weicker, "Dhrystone: A Synthetic Systems Programming Benchmark", *Comm. of the ACM*, 27 (1984), 1013 - 1030.

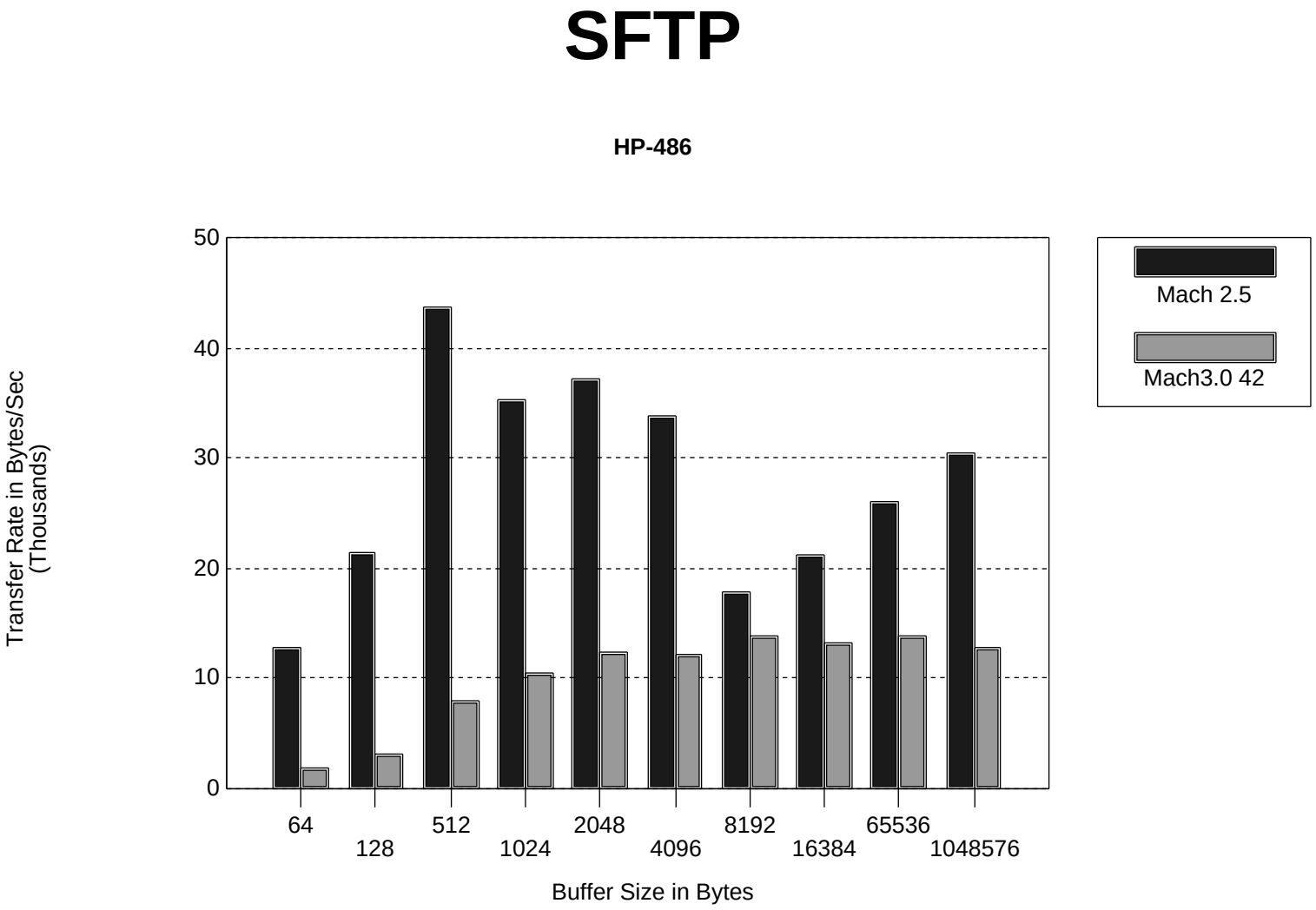


Figure 1: Performance of Sftp

# SDBASE

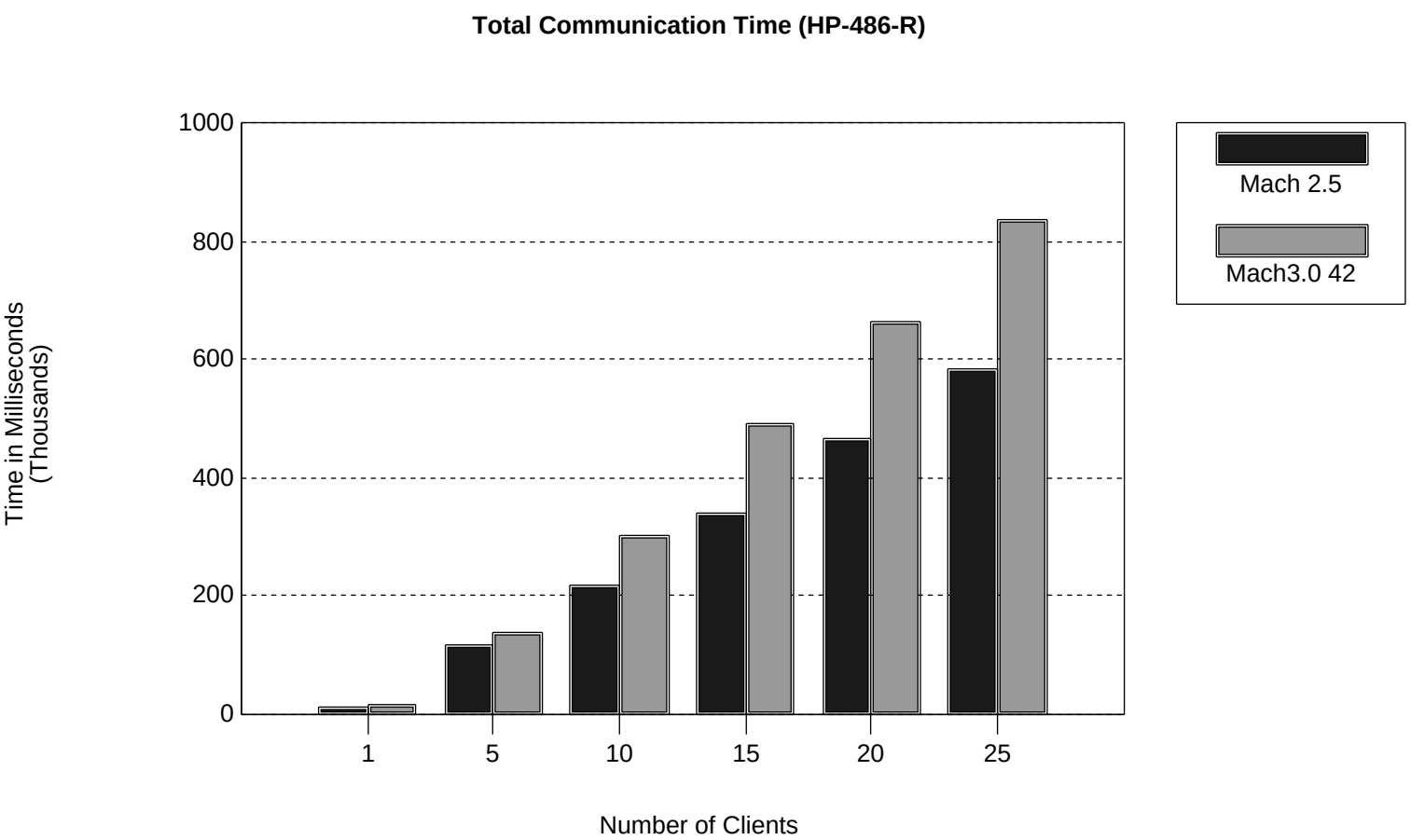


Figure 2: Total communication time for remote clients

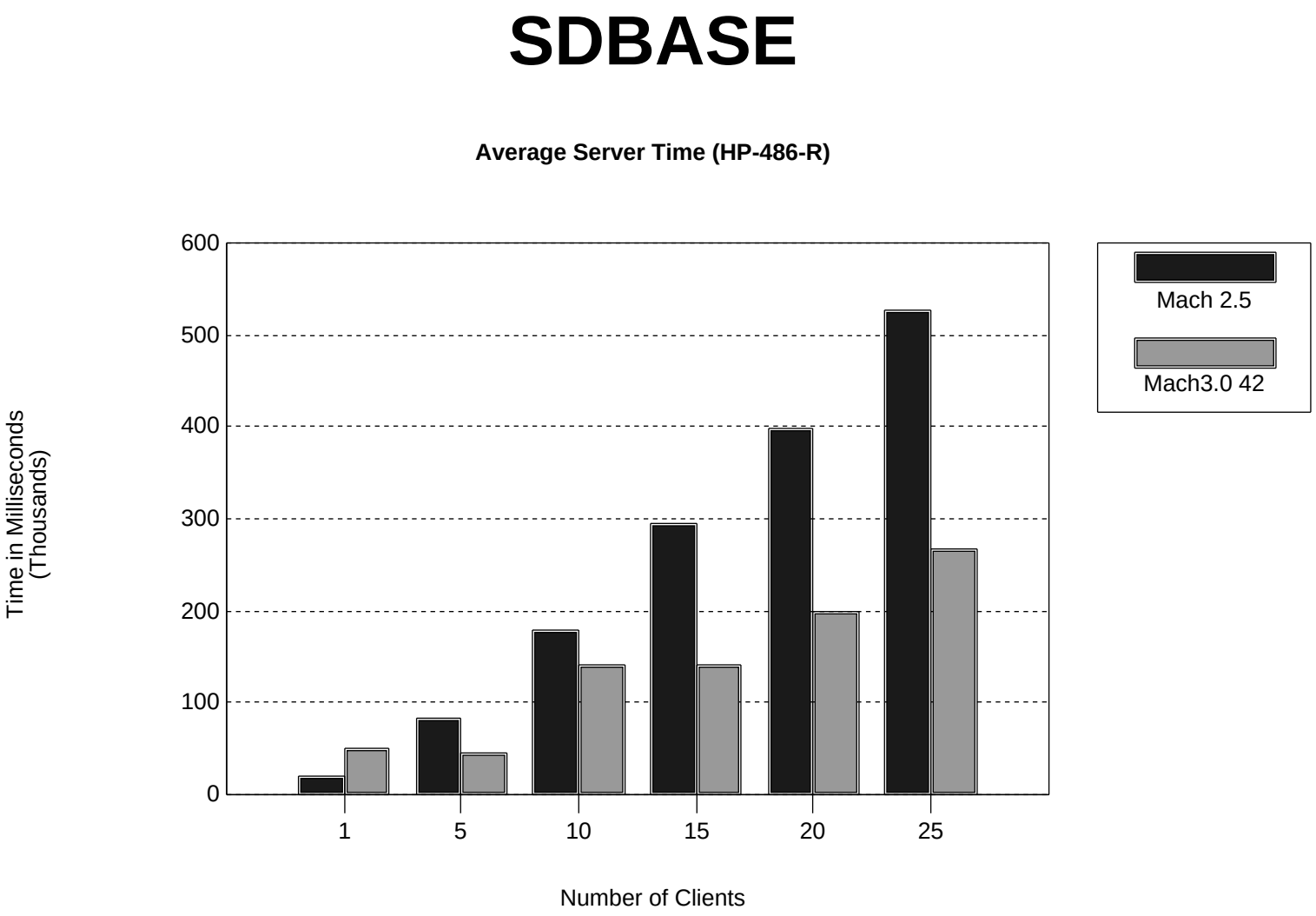


Figure 3: Average server time for remote clients

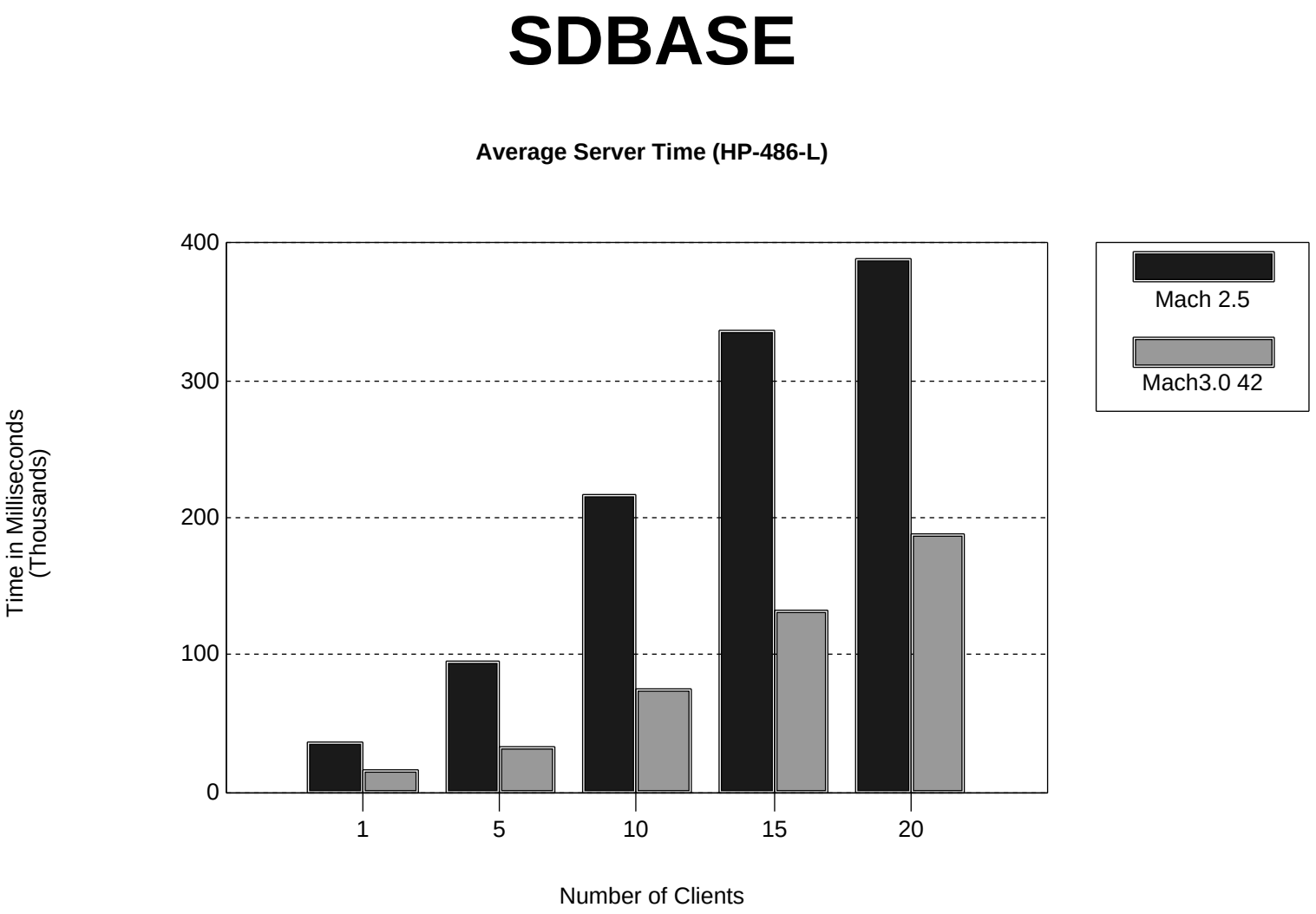


Figure 4: Average server time for local clients

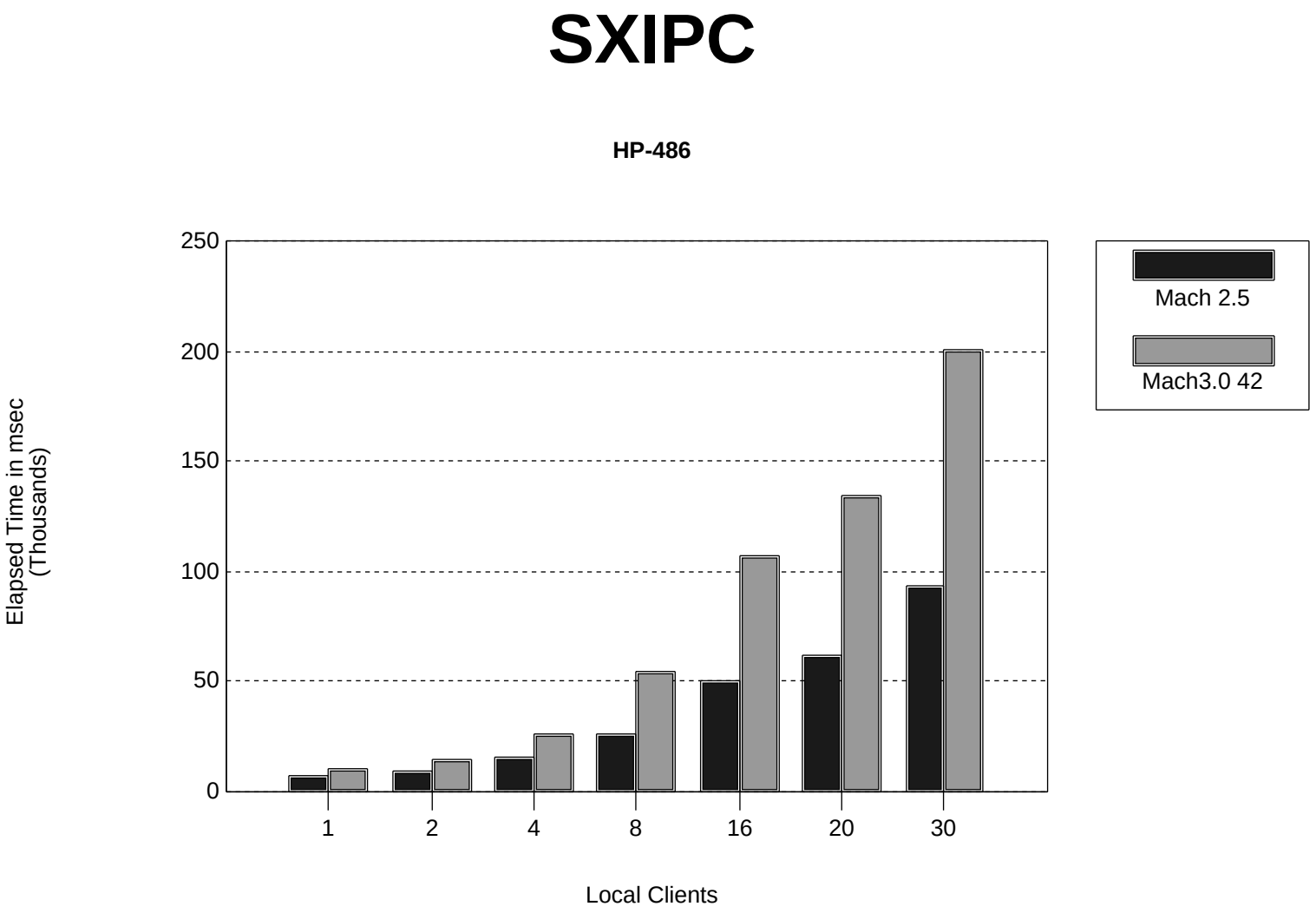


Figure 5: Local clients

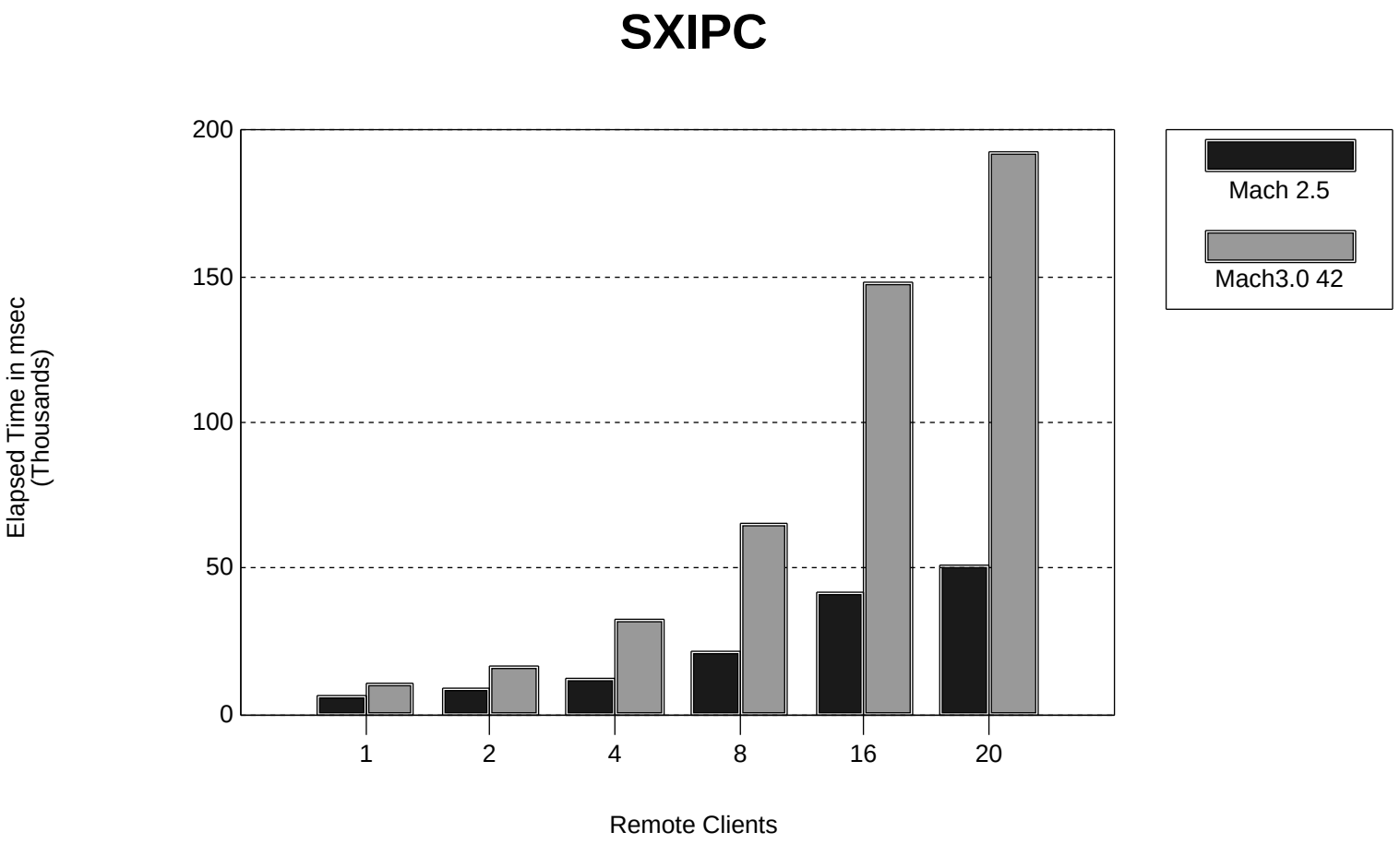


Figure 6: Remote clients

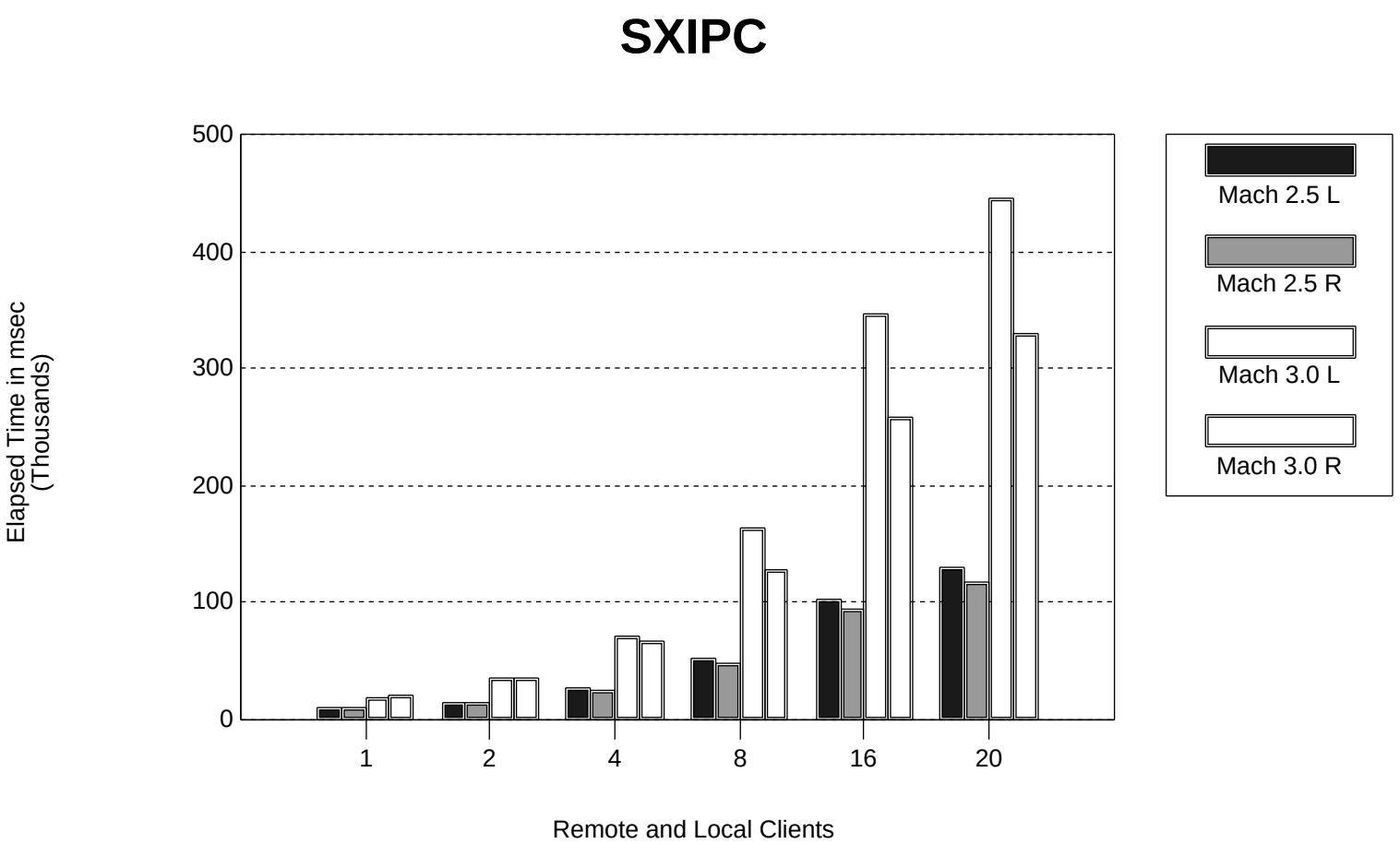


Figure 7: Local and remote clients

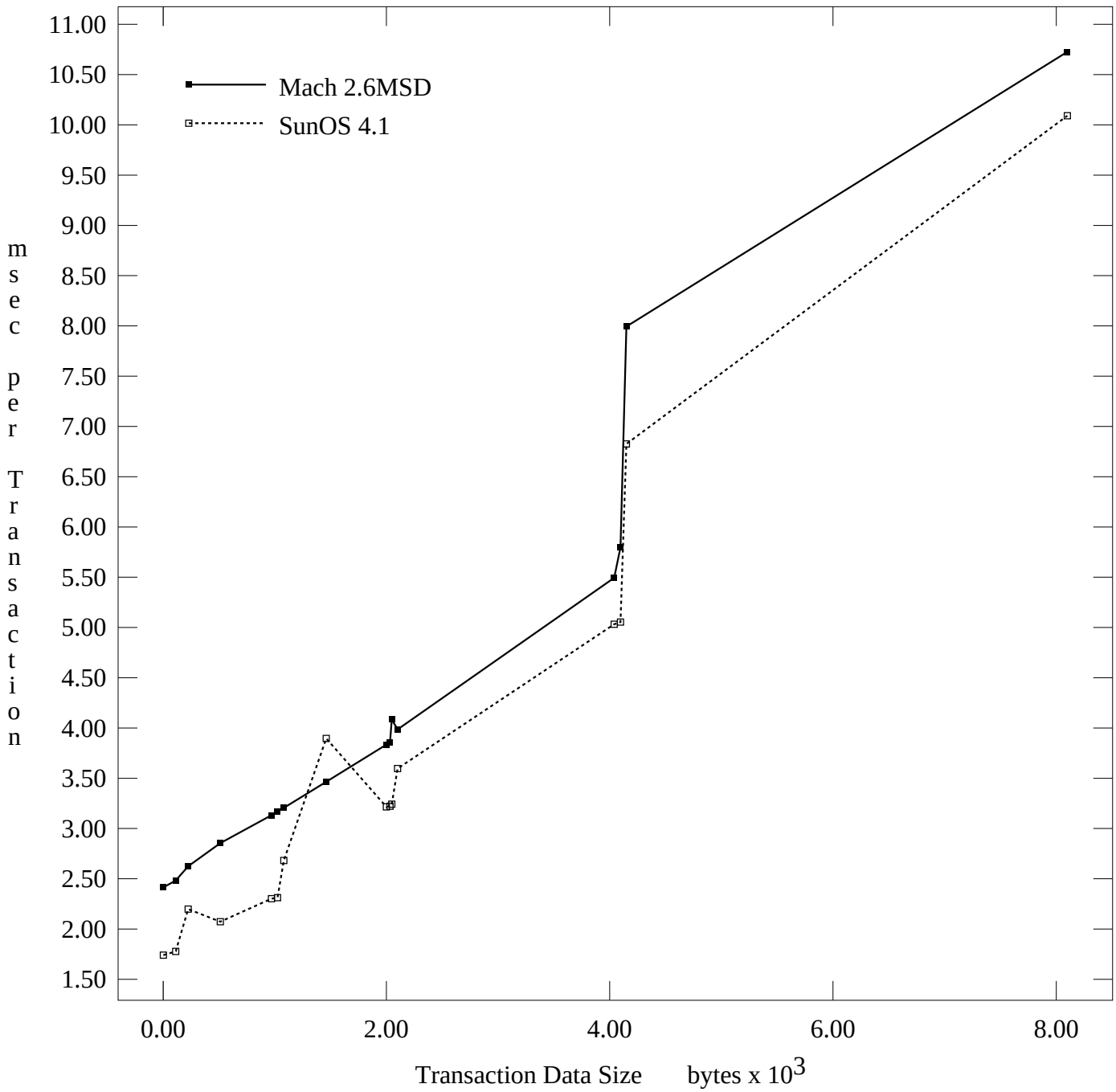


Figure 8: Socket (local) benchmark (Sun 3/60)

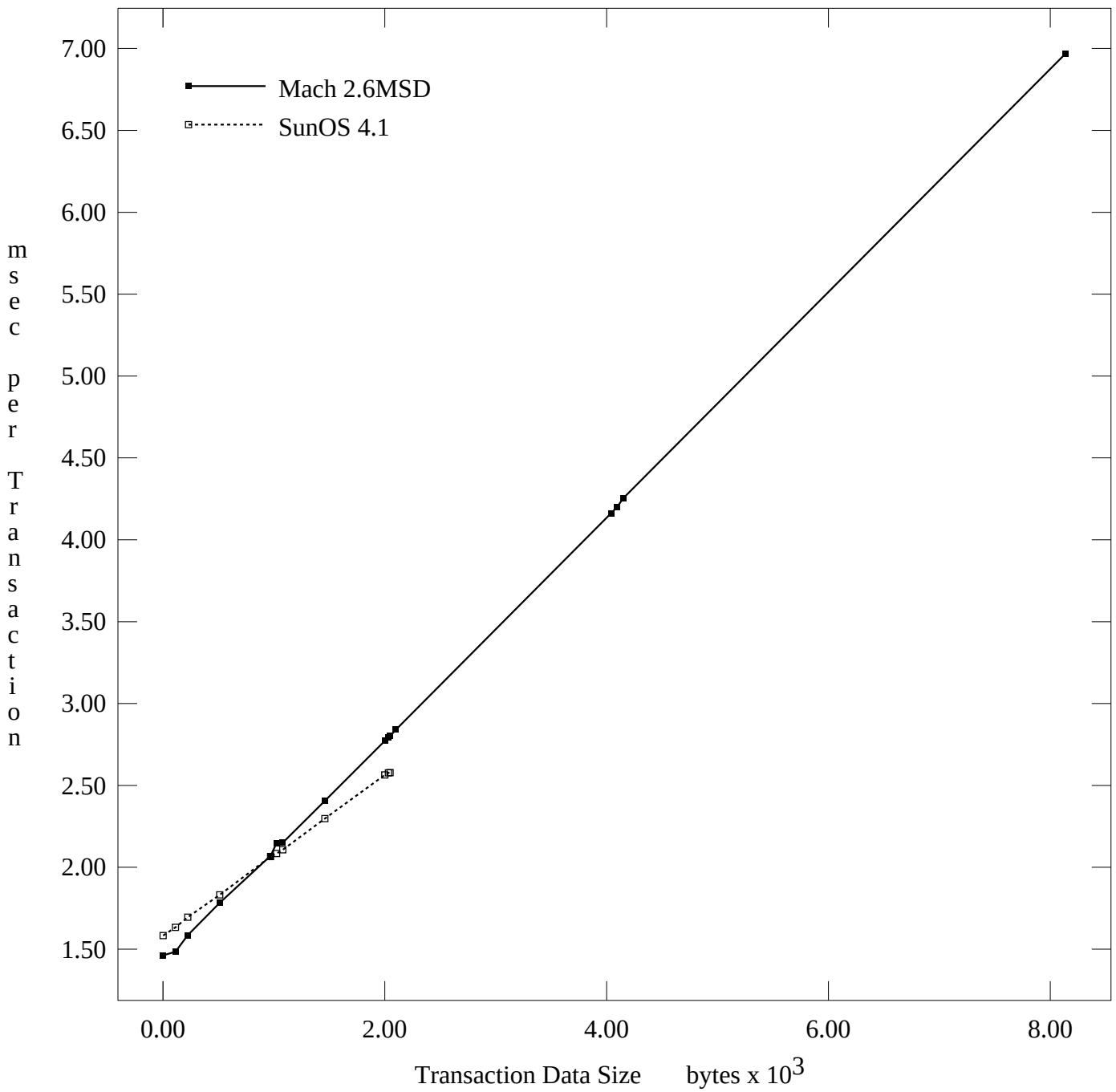


Figure 9: Message passing benchmark (Sun 3/60)

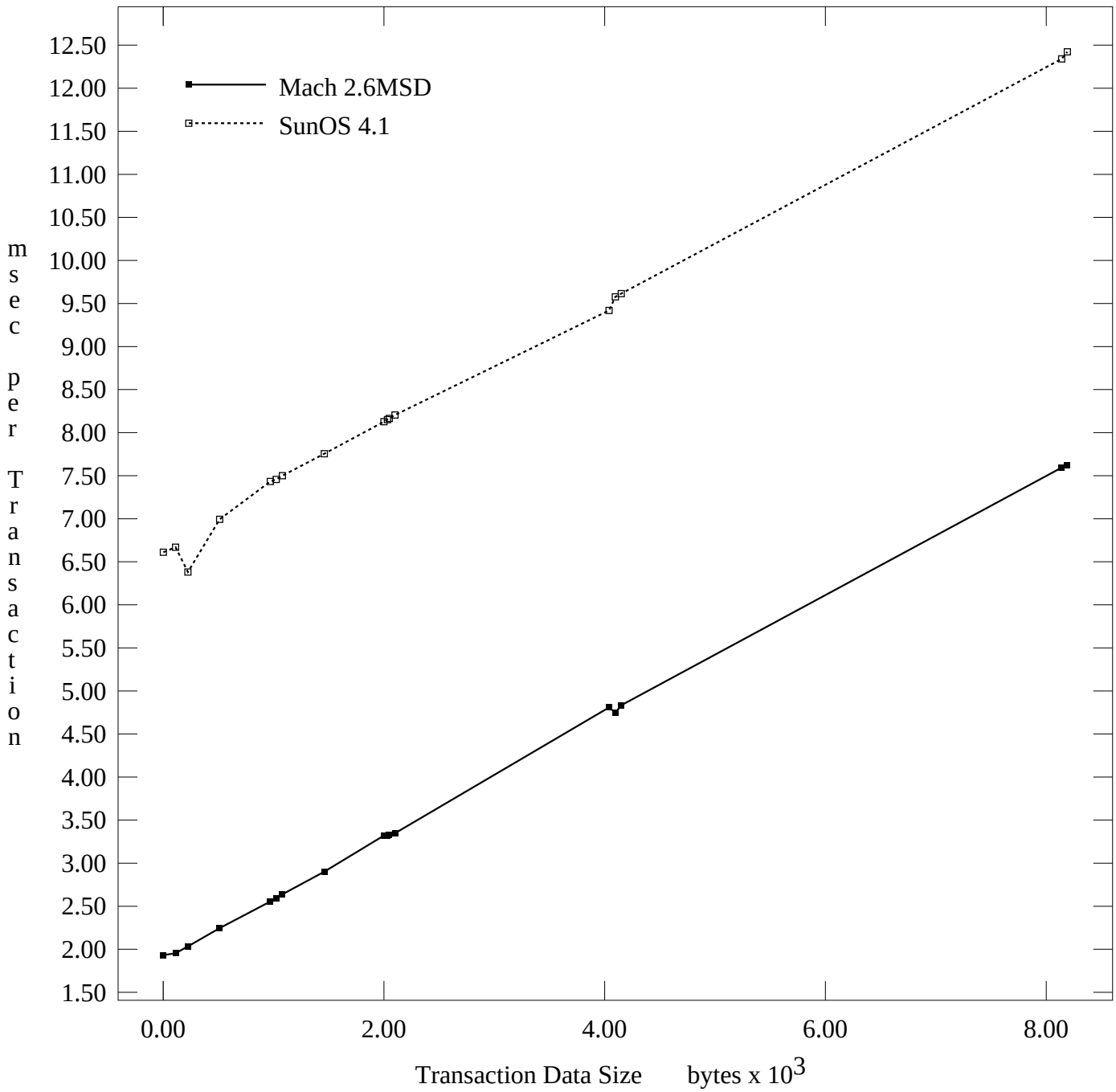


Figure 10: Shared memory benchmark (Sun 3/60)

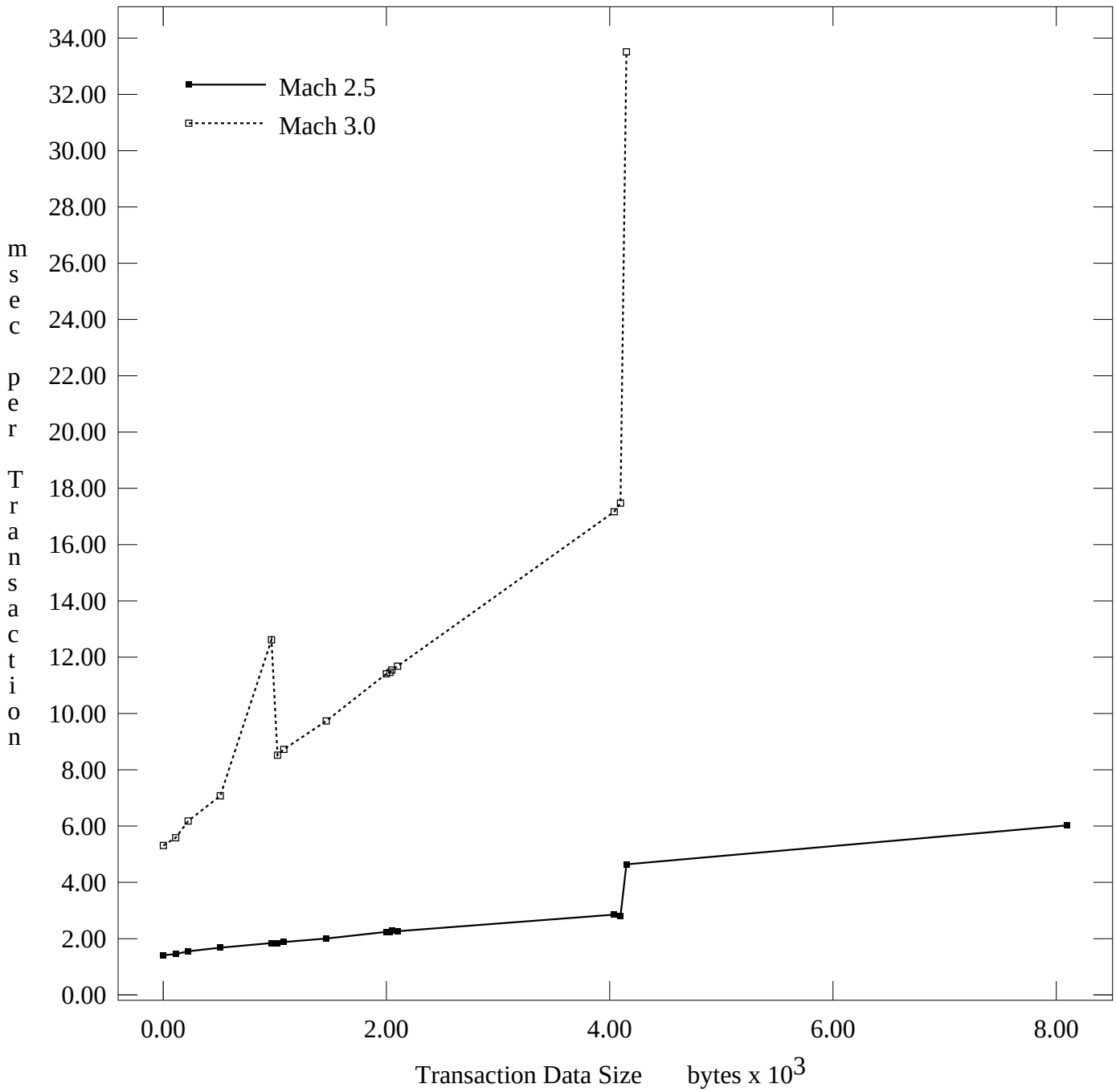


Figure 11: Socket (local) benchmark (HP RS/25c i386)