On the Performance of Courier Remote Procedure Calls Under 4.1c BSD

James R. Larus

Computer Science Division
Department of Electrical Engineering and Computer Sciences
University of California, Berkeley
Berkeley, California 94720

ABSTRACT

Courier is a remote procedure call standard developed by Xerox. This paper reports measurements of Courier's performance under 4.1c BSD Unix running on VAX-11/780s and on Sun personel workstations.

The cost of a remote procedure call is many times that of a local call. However, Courier's performance could be greatly improved by using a simpler protocol and better Ethernet interfaces.

On the Performance of Courier Remote Procedure Calls Under 4.1c BSD

James R. Larus

Computer Science Division
Department of Electrical Engineering and Computer Sciences
University of California, Berkeley
Berkeley, California 94720

1. Introduction

This paper reports on the performance of an implementation of Courier remote procedure calls under 4.1c BSD Unix[1] [Cooper 83].

Courier is a Xerox network standard [Xerox 81], developed to facilitate procedure calls between computers connected by a network and between programs written in different languages. The Xerox standard prescribes the order in which arguments are to be passed, the way that different datatypes should be packaged and shipped, and the procedure for returning results or exceptions.

Eric Cooper implemented most of Courier for C programs running under 4.1c BSD Unix. He did not implement a few portions of Courier, such as exceptions and multiple results, that are closely tied to the Mesa language [Mitchell 79] and have no counterparts in C.

Writing a distributed program with Courier is relatively painless. From a file of Mesa-like datatype and procedure heading declarations, Courier produces C routines that look like the remote procedures, but package their arguments, ship them over the network, and receive and return a result value. The only substantive change necessary to use Courier is that the client (the program issuing the remote call) must open a connection with the server (the program receiving the remote call) by specifying the remote machine's and program's names.

Courier currently uses the reliable byte-stream protocol provided by TCP [Leffler 82] to pass control information, arguments, and results. This protocol is not optimal for Courier, which would be happier with a protocol that had packet boundaries and was cheaper, such as a reliable packet protocol. However, Unix does not yet provide a packet-stream protocol, so Courier had to make due with the available protocols.

Given a new facility like Courier, many people will rush to use it without understanding its limitations and costs. This paper describes some measurements of remote procedure calls' cost, both with the conventional byte-stream Courier and with a version of Courier that I modified to use datagrams. These measurements are not definitive because of the many problems in measuring any

[1] Unix is a trademark of Bell Laboratories.

program, particularly a distributed one, on Unix. However, the numbers show that a Courier remote procedure call is much more expensive than a local call and that the time required to complete a remote call depends heavily on the load of the computer and network.

2. The Experiment

This section describes the programs that I measured and the types of measurements I collected.

2.1. Courier and PCourier

Throughout the rest of this paper, the term *Courier* will mean Eric Cooper's implementation of Courier that uses TCP and *PCourier* (for Packet Courier) will mean my reimplementation that uses UDP (datagrams).

For a variety of reasons, PCourier is not a serious candidate to replace Courier. The datagrams underlying PCourier are unreliable and unordered. To cope with these shortcomings, PCourier cheats. Arguments and results must each fit into a single packet (1,500 bytes, maximum) so that packet sequencing is not a problem. To cope with undelivered packets, PCourier times out after 10 seconds, declares the packet lost, and returns an empty result. Neither restriction seriously impaired the measurements in this paper, but either restriction would make PCourier unusable in most real systems.

2.2. Measurements

The measurements in this paper are of two varieties. The first type recorded the elapsed time to execute some remote procedure calls and the second type measured the CPU cost of the calls.

The elapsed time measurements recorded the real time needed to complete 2,000 remote procedure calls. This measurement is extremely sensitive to variations in the local and remote computers' and the network's loads, so it was run between 8 and 16 times at random intervals between 1 and 8 AM. The collected numbers reflect this sensitivity in their high variance and only give a rough idea of the minimum and average costs of a remote procedure call.

Although the computers' loads in the early morning hours are atypical, Courier's performance can best be measured without interference from other users. The numbers in this paper are, therefore, lower bounds on Courier's performance and cannot be used to predict its behavior when the computer and network are loaded.

The other set of measurements recorded, with *gprof* [Graham 82], where the client program spent its time. These measurements had little variation and helped pinpoint the reasons why Courier calls cost so much.

2.3. The Test Programs

The test programs were simple routines that took either no arguments or a single array argument and returned either nothing or a single integer. A single argument is enough to determine Courier's costs of packaging and shipping different amounts of data. Two or more arguments are equivalent, from Courier's point of view, to a larger single argument. The four test arguments were: 0 words, 1 word (4 bytes), 100 words (400 bytes), and 1000 words (4,000 bytes). Since Courier's behavior depends upon whether a function returns a value, each variety of function had a value-returning and a non-value-returning version.

2.4. The Hardware

The tests described above were run between the Berkeley computers *ucbkim* and *ucbarpa* and between *ucbkim* and *ucbrob* or *ucbchip*. The first two machines are VAX-11/780s connected by a large, heavily-loaded, 3 megabit per second ethernet. The other two machines are Sun workstations (based on the Motorola 68000) that are connected to *ucbkim* by a small, lightly-loaded, 10 megabit per second ethernet.

3. The Results

Table 1 contains elapsed time measurements for Courier.

Test		Elapsed time in seconds, for 2000 calls ± std. dev.					
		kim to arpa			kim to rob		
Arg. Size (bytes)	Result?	min.	max.	avg.	min.	max.	avg.
0	No	10.3	42.3	26.2± 9.3	13.3	115.8	33.4± 32.9
0	Yes	66.3	465.1	193.0± 142.9	118.4	160.4	129.2± 12.4
4	No	16.4	57.7	41.2± 12.4	25.1	104.6	47.7± 26.1
4	Yes	70.4	364.1	139.5± 93.8	104.6	148.9	142.4± 4.7
400	No	50.4	218.0	102.8± 53.4	65.6	172.4	88.0± 29.2
400	Yes	84.1	418.2	177.6± 106.8	154.2	182.5	166.1± 8.0
4000	No	213.6	633.8	376.7± 139.2	290.6	692.8	402.7± 121.0
4000	Yes	317.6	594.4	422.4± 94.4	396.2	776.5	479.4± 123.4

Table 1. Measurements of Courier on March 10, 1983

The table shows that the time required for a call increases greatly when Courier returns a result and as the size of the argument increases. The first increase is not surprising since Courier does not wait for a call to complete when a function does not return a value, but rather sends off the next call packet and relies on the network's queuing to keep the calls ordered. The reasons behind the second result are discussed in detail below. The time also varied widely, as might be expected for any elapsed time measurements in which outside interference is not controlled.

Table 2 shows the corresponding information for PCourier. The two tests programs that passed 1,000 word arrays could not run on PCourier because their argument was larger than a single packet. Also, the numbers in this table must be taken with a grain of salt, since they do not discount the time spent waiting

Test		Elapsed time in seconds, for 2000 calls ± std. dev.							
Arg. Size	Result?	kim to arpa				kim to rob			
(bytes)		min.	max.	avg.	*	min.	max.	avg.	*
0	No	7.1	22.5	8.5± 3.9		7.0	62.0	11.6± 11.2	
0	Yes	38.6	375.5	117.3± 80.6	170	43.6	93.8	51.6± 13.1	2
4	No	7.1	13.5	9.1± 1.4		7.0	23.1	10.1± 3.8	
4	Yes	37.4	281.3	107.4± 67.4	128	43.5	102.7	51.9± 11.4	5
400	No	22.0	76.6	28.6± 13.8		24.1	105.4	29.8± 16.1	
400	Yes	88.5	719.0	211.9± 163.8	322	68.3	149.7	87.6± 15.2	6

* Lost Packets

Table 2. Measurements of PCourier on March 19, 1983

for packets that never arrived. The figures, in the table, for lost packets count only the result packets not received by the client and do not directly record the packets not received by the server. However, since a server cannot return a result if a call packet never arrived, the figure for lost packets is approximately equal to the number of packets lost in both directions, for the function calls that return a value. I have no figures for the number of lost packets in the non-value-returning calls, but these packets do not affect the recorded speed of these calls.

Test		Courier		PCourier		Corrected PCourier	
Arg. Size	Result?						
(bytes)		msec./call	call/sec.	msec./call	call/sec.	msec./call	call/sec.
0	No	13.1	76	4.2	235	4.2	235
0	Yes	96.5	10	58.6	17	21.6	46
4	No	20.6	49	4.5	219	4.5	219
4	Yes	69.7	14	53.7	19	25.9	39
400	No	51.4	19	14.3	70	14.3	70
400	Yes	88.8	11	106.0	9.4	35.9	28
4000	No	188.3	5				
4000	Yes	211.2	5				

Table 3. Calls per Second, VAX to VAX

Table 3 lists the cost, in milliseconds per call, and the maximum number of calls per second between the two VAXs (based on the average of the measurements). The column labeled "Corrected PCourier" discounts the effects of the time-outs due to lost packets. Table 4 shows the corresponding figures for VAX to Sun calls.

Test Arg. Size (bytes)	Result?	Courier msec./call	Courier call/sec.	PCourier msec./call	PCourier call/sec.	Corrected PCourier msec./call	Corrected PCourier call/sec.
0	No	16.7	60	5.8	172	5.8	172
0	Yes	64.6	15	25.8	39	25.4	39
4	No	23.8	42	5.0	198	5.0	198
4	Yes	71.2	14	25.9	39	24.9	40
400	No	44.0	23	14.9	67	14.9	67
400	Yes	83.1	12	43.8	23	42.5	23
4000	No	201.3	5				
4000	Yes	248.5	4				

Table 4. Calls per Second, VAX to Sun

3.1. Where is the Time Spent?

To see where Courier spends its time, I profiled the client program that invoked the remote procedures and includes Courier's network interface code. Table 5 shows the most costly procedures in the exchanges between *ucbkim* and *ucbarpa* and Table 6 shows the same procedures in the exchange between *ucbkim* and *ucbrob*. The corresponding information for PCourier is in Tables 7 and 8.

Arg. Size (bytes)	Result?	Time in CPU Seconds Total	*write*	*read*	swapping bytes
0	No	13.7	12.7 (93.0%)	0.05 (0.2%)	
0	Yes	25.5	13.8 (54.2%)	6.4 (25.1%)	
4	No	21.5	18.9 (88.1%)	0.02 (0.1%)	
4	Yes	30.4	19.6 (66.6%)	8.1 (26.7%)	
400	No	36.4	24.3 (66.8%)	0.03 (0.1%)	9.6 (24.4%)
400	Yes	43.2	22.3 (51.6%)	6.4 (14.8%)	9.9 (24.9%)
4000	No	159.6	35.9 (22.5%)	0.03 (0.0%)	120.7 (75.6%)
4000	Yes	140.5	35.2 (25.1%)	8.16 (5.8%)	92.5 (65.9%)

Table 5. Execution Profile of Courier (ucbkim to ucbarpa), March 1, 1983

Arg. Size (bytes)	Result?	Time in CPU Seconds Total	*write*	*read*	swapping bytes
0	No	15.3	14.2 (92.7%)	0.04 (0.3%)	
0	Yes	26.9	14.9 (55.6%)	7.9 (29.5%)	
4	No	23.3	20.7 (88.9%)	0.04 (0.2%)	
4	Yes	32.8	21.5 (65.5%)	7.0 (21.2%)	
400	No	41.1	27.2 (66.2%)	0.04 (0.1%)	10.8 (26.4%)
400	Yes	47.7	26.6 (55.9%)	7.3 (15.2%)	9.0 (18.9%)
4000	No	160.4	48.0 (29.9%)	0.03 (0.0%)	109.1 (68.0%)
4000	Yes	155.1	48.2 (31.1%)	7.3 (4.7%)	94.8 (61.1%)

Table 6. Execution Profile of Courier (**ucbkim** to ucbchip), March 1, 1983

Arg. Size (bytes)	Result?	Time in CPU Seconds Total	*sendto*	*recv + select*	swapping bytes
0	No	8.8	7.9 (89.5%)	0.03 (0.3%)	
0	Yes	20.1	9.6 (47.0%)	7.3 (36.3%)	
4	No	9.2	7.5 (81.0%)	0.03 (0.3%)	
4	Yes	20.2	8.57 (42.0%)	6.9 (33.8%)	
400	No	25.3	12.2 (48.0%)	0.02 (0.1%)	8.8 (35.0%)
400	Yes	37.8	12.3 (33.0%)	9.0 (24.0%)	9.8 (26.0%)

Table 7. Execution Profile of PCourier (ucbkim to ucbarpa), March 19, 1983

Arg. Size (bytes)	Result?	Time in CPU Seconds Total	*sendto*	*recv + select*	swapping bytes
0	No	9.2	8.3 (90.0%)	0.02 (0.2%)	
0	Yes	19.2	10.4 (54.0%)	6.0 (31.0%)	
4	No	9.7	7.8 (80.0%)	0.02 (0.2%)	
4	Yes	20.5	10.4 (51.0%)	6.7 (32.0%)	
400	No	25.2	12.7 (50.0%)	0.01 (0.0%)	8.1 (32.0%)
400	Yes	34.1	11.9 (35.0%)	7.5 (22.0%)	9.7 (27.0%)

Table 8. Execution Profile PCourier (ucbkim to ucbchip), March 19, 1983

4. Analysis

Because of the large variance in the measurements, they cannot accurately predict or compare Courier's or PCourier's performance. Nevertheless, some rough comparisons are possible.

4.1. Streams and Packets

PCourier, based on datagrams, is much faster than Courier, which uses byte streams. Table 9 compares the relative speeds of Courier and PCourier. The column headed by "Corrected PC" is based on the reported PCourier times minus the timeout delays. These times are a rough approximation of the time required by a version of Courier that had a reliable datagram facility.

Arg. Size (bytes)	Result?	Kim to Arpa Courier/PC	Kim to Arpa Courier/Corrected PC	Kim to Rob Courier/PC	Kim to Rob Courier/Corrected PC
0	No	3.1	3.1	2.9	2.9
0	Yes	1.6	4.5	2.5	2.5
4	No	4.5	4.5	4.7	4.7
4	Yes	1.3	2.7	2.7	2.9
400	No	3.6	3.6	2.9	2.9
400	Yes	0.84	5.9	1.9	2.0
Avg.		2.49± 1.3	4.05± 1.06	2.9± 0.96	3.0± 0.83

Table 9. Ratio of Courier and PCourier (PC) Performance

Based on the uncorrected figures, PCourier is 2.5 to 3 times faster than Courier. Discounting PCourier's excessive time-out delays, it is 3 to 4 times faster than Courier.

4.2. Does the Ethernet's Speed Matter?

The VAXs are connected by a 3 megabit per second ethernet and the Suns are connected by a 10 megabit per second ethernet. Presumably, the latter network would be faster. Table 10 shows the ratio of elapsed time the tests between two VAXs and between a VAX and a Sun. Considering the difference in speed

Arg. Size (bytes)	Result?	Courier 3MB/10MB	PCourier Uncorrected 3MB/10MB	PCourier Corrected 3MB/10MB
0	No	0.78	0.73	0.73
0	Yes	1.49	2.3	0.85
4	No	0.86	0.9	0.9
4	Yes	0.98	2.1	1.04
400	No	1.17	0.96	0.96
400	Yes	1.17	2.4	0.84
4000	No	0.94		
4000	Yes	0.88		
Avg.		1.02	1.6	0.89

Table 10. Ratio of Elapsed Time on Different Speed Networks

between a VAX and a 68000 and the heavier traffic on the departmental ethernet, the underlying speed of the ethernet appears to make little difference to Courier's performance.

However, the previous statement must be qualified. The departmental ethernet had many more lost packets than the Suns' small network. The effect of the waiting time caused by these lost packets is evident in the uncorrected figures for PCourier in Table 10 (middle column). A reliable packet facility would not have a time-out as long as the one used in this experiment, so that these figures set an upper bound on the differences in speeds caused by lost packets.

4.3. Where Does the Time Go?

Tables 5 and 6 and 7 and 8 show that a client program running on *ucbkim* spends about the same amount of time whether the server is on *ucbarpa* or a Sun. So, in the following discussion, we only consider the figures for VAX to VAX traffic.

One point that is immediately obvious is that Courier takes far less CPU time to send and receive messages with packets rather than streams. Table 11 shows the ratio of the CPU time required for both protocols.

Arg. Size (bytes)	Result?	Total Courier/PCourier	Sending Courier/PCourier	Receiving Courier/PCourier
0	No	1.6	1.6	n/a
0	Yes	1.3	1.4	0.87
4	No	2.3	2.5	n/a
4	Yes	1.5	2.3	1.2
400	No	1.4	2.0	n/a
400	Yes	1.1	1.8	0.71
Avg.		1.5	1.9	0.93

Table 11. Ratio of CPU Time in Courier and PCourier

When the amount of data shipped over the network grows, the time to put the data in network order dominates. The VAX, unfortunately, lays out its bytes in an order different from most other machines and different from the Courier standard. Rearranging these bytes is very expensive, particularly for large quantities of data.[2] On the other hand, the 68000's byte arrangement agrees with the network standard, so the Suns do not have to rearrange their bytes. Hence, sending large amounts of data from the Suns should be much cheaper than sending it from the VAX. Unfortunately, the Suns were missing the profiling code necessary to verify this conjecture.

[2] The time to rearrange the bytes for small amounts of data (e.g. 1 word) was not measured since the rearrangement was done by a macro that gprof could not time.

4.4. Relative Performance

To compare the cost of remote and local procedure calls, I also measured the time to execute 2,000 local procedure calls that copied their argument (a normal C call would pass an array by reference and would have negligible cost). Passing a 4,000 byte array 2,000 times takes 13.7 CPU seconds locally and about 160 CPU seconds with Courier. A 400 byte array takes 1.4 CPU seconds locally, about 45 CPU seconds with Courier, and about 25 CPU seconds with PCourier.

Bruce Nelson, in his thesis on remote procedure calls [Nelson 81], measured the speed of a variety of implementations of remote procedure calls on Xerox computers. Using a software implementation on a Dolphin similar to Courier, he obtained a round-trip call time of about 25 milliseconds for a call without arguments and about 29 milliseconds for a single word (2 bytes) call. These numbers are a bit higher that the corrected PCourier cost (Table 3) and much lower than the Courier cost.

Nelson was able to reduce the cost of a remote procedure call down to the range of a normal procedure call by microcoding the call handler and dispensing with operating system support. Such an approach is unlikely to be popular or possible under Unix.

5. Other Results

Robert Hagmann measured the number of messages per second that could be sent over a stream between two single-user VAXs [Hagmann 83]. The largest number of round-trip exchanges that he obtained was about 60 per second, which means that 34 seconds is the minimum time to make 2,000 procedure calls that return results. The measurements in Table 1 are much larger than this bound, which implies that Courier is not limited by the operating system's or network's performance.

6. Conclusion

Although a Courier call is many times more expensive than a local call, Courier spends most of its time packing, sending, or receiving data. Any program that sent the same data between two remote computers would have to do an equivalent amount of work. With the exception of byte swapping, which is pointless when the communicating computers are VAXs, Courier does not have any blatant inefficiencies. Byte swapping on a VAX is probably best done in hardware by the Ethernet interface since this operation is extremely costly to perform in software.

This is not to say that Courier's performance cannot be improved. Although the cost of a reliable packet protocol is higher than that of a datagram protocol, like the one measured in this paper, the packet protocol is certainly cheaper than the stream protocol that Courier currently uses. Speeding up the underlying protocol would have a noticeable effect on Courier's performance.

The real advantage of Courier is that it is quick and simple to use. After mucking with the network primitives to modify Courier [Leffler 83], I have a deep appreciation for the enormous amount of effort and difficulty that Courier subsumes.

Bibliography

[Cooper 83]
Cooper, E., "Writing Distributed Programs with Courier," UCB, undated.

[Graham 82]
Graham, S. L., Kessler, P. D., and McKusick, M. K., "gprof: A Call Graph Execution Profiler," in *Proceedings of the Sigplan '82 Symposium on Compiler Construction,* June 1982.

[Hagmann 83]
Hagmann, R., Personal Communications, February 24, 1983.

[Leffler 82]
Leffler, S. J., Joy, W., and Fabry, R., *4.2BSD Networking Implementation Notes - Draft of September 6, 1982,* CSRG, UCB, Sept. 1982.

[Leffler 83]
Leffler, S. J., *A 4.2BSD Interprocess Communication Primer, Draft of February 10, 1983,* CSRG, UCB Feb. 1983.

[Mitchell 79]
Mitchell, J. G., Maybury, W., and Sweet, R., *Mesa Language Manual,* Xerox PARC report CSL-79-3, April 1979.

[Nelson 81]
Nelson, B. J., *Remote Procedure Call,* Xerox PARC report CSL-81-9, May 1981.

[Xerox 81]
Xerox Corp., "Courier: The Remote Procedure Call Protocol," XSIS 038112, Dec. 1981.