Measurements of DCE RPC Performance in an OS/2 Environment

Ying Sun*
Rick Bunt
Greg Oster
Department of Computer Science
University of Saskatchewan

August 9, 1996

Abstract

Developing and managing applications for environments consisting of independently configured computing systems interoperating across network connections is of considerable interest in the commercial sector and presents many research challenges. The Open Software Foundation's Distributed Computing Environment (DCE) has evolved to address the need for a vendor-neutral platform to which distributed applications can be developed. Central to the design philosophy of DCE is its reliance on the Remote Procedure Call (RPC) to facilitate communication among the entities in the distributed environment. Since it profoundly affects the performance of both the DCE environment and applications running on top of it, RPC's performance is very much a concern of both application developers and system managers in a DCE installation. To shed light on some of these issues, this paper reports results from an ongoing investigation of DCE RPC and its performance in a commercially-available implementation, for IBM's OS/2 operating system. The results obtained to this point show the effects of protocol processing, of network load, and of flow control mechanisms, and suggest areas for further study.

1 Introduction

Distributed computing involves systems that operate across network connections, using the resources of multiple machines in a transparent fashion. Developing applications for such

---

*Presently at IBM Canada Laboratory, 1150 Eglinton Avenue East, North York, Ontario, Canada, M3C 1H7

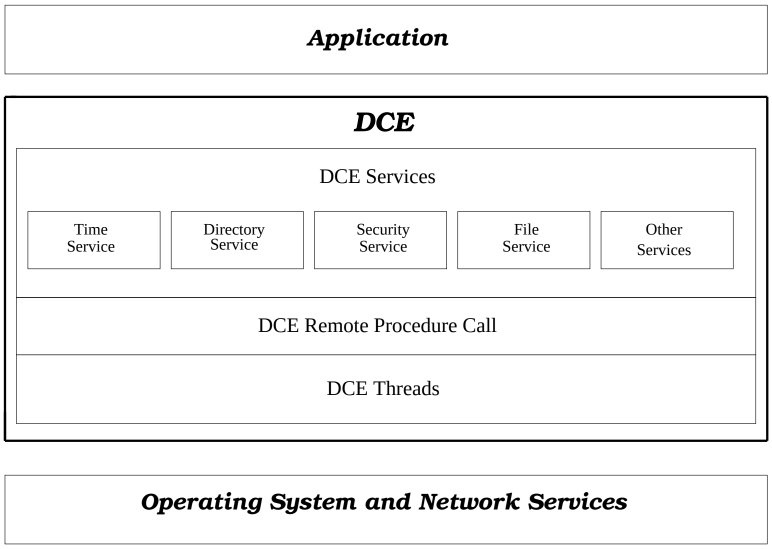


Figure 1: Distributed Computing with DCE

environments, and managing their execution, are problems of considerable interest in the commercial sector and present many research challenges. Among these are issues relating to the performance of distributed systems and the applications that run on them.

In response to a growing need for vendor-neutral platforms to which distributed applications can be developed, the Open Software Foundation (OSF)[1] developed the Distributed Computing Environment (DCE) midware platform. DCE sits on top of existing operating systems, providing a uniform set of services and a consistent interface to applications (as illustrated in Figure 1). Communication among the DCE entities is provided by means of remote procedure calls (RPCs) [1]. As the basis for all communication in DCE, RPC has a profound impact on performance in a DCE environment, not only system performance but application performance as well.

Our general interests in this project lie in issues affecting the performance of distributed applications, the performance of DCE running these applications, and the performance of systems (and networks) running DCE running these applications, all in the context of IBM's OS/2 operating system. OS/2 has been chosen as the basis for our research, in part, because it is a platform that has not been adequately studied (or at least reported on). The research reported in this paper is aimed at securing a better understanding of DCE RPC and issues affecting its performance. We begin with a brief overview of DCE, and then turn to a more detailed description of DCE RPC, in which issues affecting its performance are identified. Finally, we present results from a set of measurement experiments.

---

[1] OSF is a collaborative venture of several major vendors, including IBM, DEC, and Hewlett Packard, originally formed to develop and market UNIX-based technologies.

2 A Brief Introduction to DCE

The Distributed Computing Environment evolved from the efforts of the Open Software Foundation to provide vendor-independent tools and services to support the development, execution, and management of distributed applications in a heterogeneous computing environment. Such environments consist of clusters of machines, ranging from personal computers to mainframes, communicating with each other over a network to provide a set of resources for the applications to use. DCE provides a high-level, coherent environment for both distributed application developers and system administrators that is based on three popular distributed computing models: client/server, remote procedure call, and data sharing. Good general overviews of DCE can be found in [10], [4] and [11].

DCE provides tools and services to the application developer, along with an environment in which they can operate. These fall into two broad categories: tools to support the development of distributed applications, and services to support the distributed applications as they execute. Some of these services are illustrated in Figure 1. These include a mechanism for synchronizing the clocks on different machines to provide a global sense of time, and a directory service to provide system-wide access to resources. In addition, DCE provides two other facilities for programming distributed applications: threads and RPC. DCE provides a uniform threads interface, whether or not the operating system on the particular host machine provides a threads facility. Where it is able, DCE uses the threads already provided to implement DCE threads. In systems where there are no native threads, DCE provides its own package. An RPC facility is provided in the same way, and the other services of DCE are built on top of these two. More details of DCE RPC are given in Section 3.

3 Overview of DCE Remote Procedure Call

The Remote Procedure Call (RPC) underlies all functionality of DCE. Cell[2] servers communicate via RPCs, application clients use RPCs to locate application servers, and all subsequent communications between application clients and application servers take place using RPCs. Because of this high reliance on RPCs, RPC performance is critical to both cell performance and application performance. This paper focuses on RPC performance between application clients and application servers, performance that should be representative of all RPC performance in the cell. Our emphasis is on issues affecting DCE RPC performance in an environment of OS/2 machines connected by an Ethernet.

We begin with a brief overview of some important concepts relating to DCE RPC.

---

[2] A cell is a group of DCE machines that work together and are administered as a unit.

3.1 Types of RPC

Two forms of RPC are available to a DCE client or server: connection-oriented and connectionless RPC. From an application standpoint, servers will typically support both forms. A clients, on the other hand, may use either or both, depending on its needs. A client connecting with only a single server, for example, may use only connection-oriented RPC, since the connection protocol sequence works well for establishing and maintaining a connection. In particular, if a server is unavailable the error can be quickly detected. For clients which need to connect to many servers, however, the connectionless RPC may be preferable because it has low operating system overhead. Ultimately, the choice depends on the requirements of the application and the environment in which it will be run.

3.2 Run-time Processing

An RPC between an application client and an application server is processed by various system and network services, as illustrated in Figure 2. This processing takes place as follows:

- The RPC data in the RPC is first marshalled into a Network Data Representation, where it obtains immunity from byte ordering and other inter-operability problems of a potentially heterogeneous environment.
- The marshalled data is decomposed by the RPC run-time software into a series of Protocol Data Units (PDUs).
- Each PDU is then passed to either the UDP layer or the TCP layer. UDP is used for a connectionless RPC while TCP is used for a connection-oriented RPC. At the UDP or TCP layer, each PDU is further decomposed into UDP or TCP packets.
- The UDP or TCP packets are then converted into IP packets, and sent to the Ethernet.

Responses from the server follow a similar process, passing up through the IP and UDP or TCP layers, and then to the PDU and unmarshalling phases.

3.3 Decomposing the RPC Packets

As the various layers of the protocol stack process the RPC (with its data payload), it is broken down into units of specified size, and header fields are prepended where required by the protocol. This process is illustrated in Figure 3 for a connectionless RPC over UDP/IP, and in Figure 4 for a connection-oriented RPC over TCP/IP. In each case, both client and server are on OS/2 platforms. The *fragmentation* of the original RPC "packet" into a larger number of smaller units affects performance in at least two ways. First, the extra header information added at each level increases the amount of data that must be transmitted, and

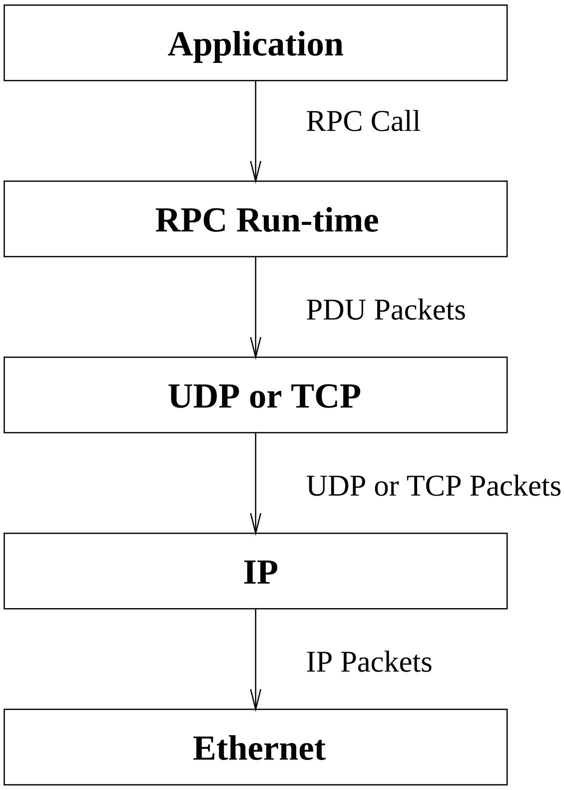


Figure 2: Processing RPCs for Ethernet Transport

thus increases the load on the network. Second, additional delays are introduced when these headers must be added and when packets are received and reassembled.

In the example shown in Figure 3, the original 6000-byte RPC packet results in five IP packets on the Ethernet. The RPC packet is first broken down by the RPC run-time layer into PDUs of at most 4312 bytes. For the first PDU, this includes an 80-byte RPC PDU header, an 8-byte RPC size field, and 4224 bytes of RPC data. For subsequent PDUs, this includes an 80-byte RPC PDU header, and at most 4232 bytes of RPC data. Each PDU is then passed to the UDP layer, where an 8-byte UDP header is added. Each resulting UDP packet is then split into packets of at most 1480 bytes, and handed off to the IP layer. From there, the 20-byte IP header is added, and the packets are sent out in 1500-byte Ethernet packets. As shown in Figure 3, a 6000-byte connectionless RPC will generate five packets on the network – the first three with size 1500, 1500, and 1380 bytes, and the last two with size 1500 and 404 bytes. A total of 6284 bytes are transmitted.

The RPC decomposition in TCP is similar to that in UDP in OS/2 DCE. The RPC run-time layer creates PDUs of at most 4352 bytes. For the first PDU, the contents are 24 bytes for the RPC PDU header, 8 bytes for the size of the RPC data, and 4320 bytes for the RPC data. Subsequent PDUs contain the 24-byte RPC PDU header, and at most 4328 bytes

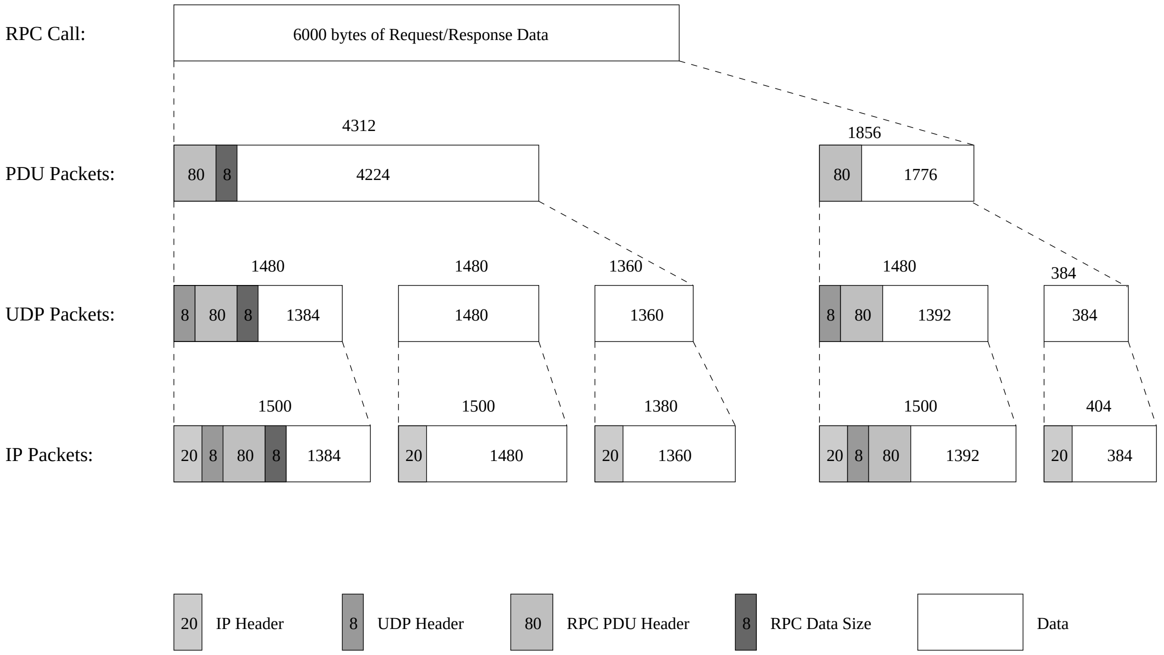


Figure 3: Decomposing a Connectionless RPC for Transport over UDP (OS/2 to OS/2)

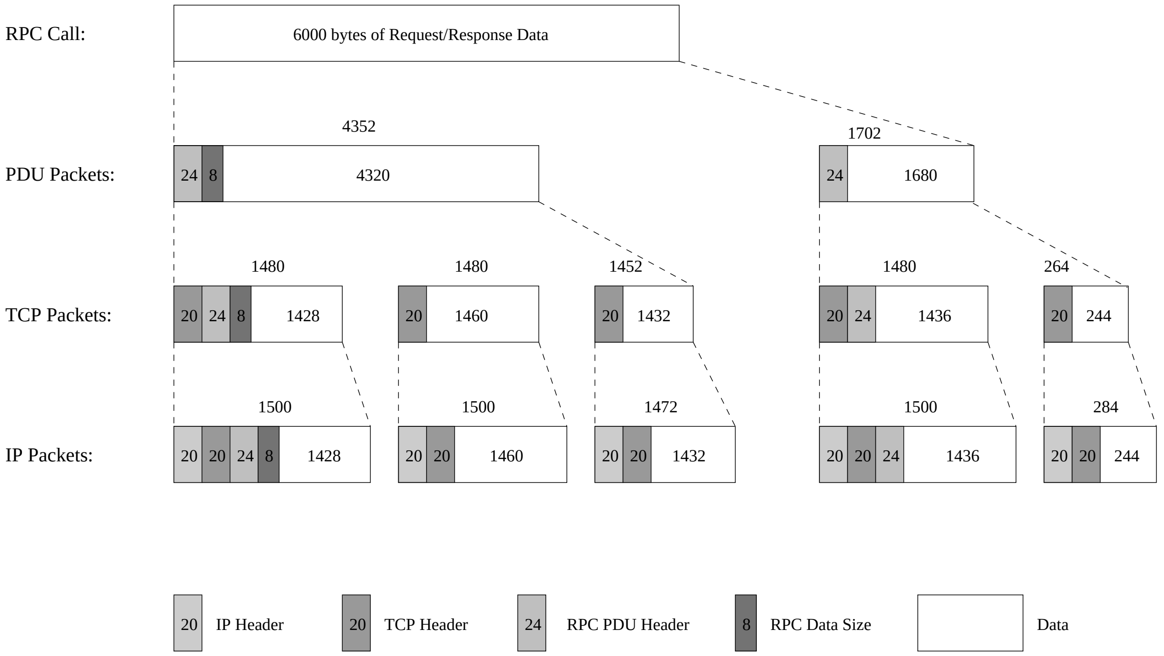


Figure 4: Decomposing a Connection-Oriented RPC for Transport over TCP (OS/2 to OS/2)

of RPC data. The TCP layer fragments the PDU into 1460-byte pieces, and adds a 20-byte TCP header. The resulting 1480-byte TCP packets then have a 20-byte IP header added, and the result is a 1500-byte packet ready for the Ethernet. Figure 4 shows the breakdown of 6000 bytes of data in an RPC call. The five resulting IP packets are of size 1500, 1500, 1472, 1500, and 284 bytes, respectively. A total of 6256 bytes are transmitted.

3.4 Other Issues

3.4.1 The First RPC

The first RPC plays a special role in the communication between the application client and the application server. The *binding* of client to server occurs during the first RPC. As well, it is during the first RPC that both the application client and the application server must negotiate a mutually agreeable PDU size. For the connectionless RPC in OS/2, the initial PDU size is 1024 bytes. In OS/2 to OS/2 RPC communications, the negotiated PDU size is 4312 bytes. For the connection-oriented RPC in OS/2, the initial PDU size is 4352 bytes, but the PDUs (and their respective headers) from the first RPC are concatenated together before being split into TCP packets. This means that the second PDU header (if there is one) will be at the end of the third IP packet. The negotiated size (again, in OS/2 to OS/2 communication) is 4352 bytes. In a heterogenous environment (such as an application client running under DCE on Digital Unix, and an application server running under DCE for OS/2), the negotiated PDU sizes may be different from the ones given here.

3.4.2 Flow Control

Flow control is an important issue in any network-based environment since overloading a communication channel can have unfortunate performance consequences. Dropped packets must be retransmitted and this increases both delay and network load. Since connection-oriented RPC is built on the TCP transport protocol, which already provides network-level flow control, no additional work needs to be done by DCE for this type of communication. Connectionless RPC, however, is another matter. Because connectionless RPC is built on UDP, which does nothing to control packet flow on the network, DCE must manage flow control on its own when it transfers connectionless multi-PDU requests or responses. In this case, DCE applies a version of the familiar *slow-start* flow-control algorithm [5] to control the transmission of data. The introduction of flow control to connectionless RPC affects the pattern of network traffic generated by multi-PDU RPCs, and this, in turn, affects the performance of connectionless RPC, especially when large amounts of data are transferred.

The slow-start approach relies on the receiver's telling the sender when it can send more data units (PDUs) through the network. The sender begins in the "slow-start" phase, by sending one PDU to the receiver. Upon receiving the acknowledgement for this PDU from the data receiver, the sender increments the size of a *window* parameter. This parameter controls the number of PDUs that can be sent out before requiring acknowledgement from the receiver. The window increases (usually exponentially) until it reaches a prescribed upper limit. When sending many PDUs with slow start, the sender does not begin sending them all at once, but rather starts "slowly", sending them out 1, 2, 4, and so on, at a time. How quickly the window grows depends on the speed with which the sender receives acknowledgements for sent PDUs, which, in turn, depends on the amount of congestion in the network. If the the sender fails to get an acknowledgement for a sent PDU, the window stops growing. The sender backs off and begins sending once again at the slow rate. This simple mechanism provides both flow control and congestion control.

3.5 Issues Affecting RPC Performance

RPC performance is affected by a number of factors, including the characteristics of the hardware and software environment, and the nature of the DCE application (for example, the frequency of RPCs and the choice of RPC type to use). Some of these factors have already been mentioned, such as the impact of packet fragmentation, but there are other issues as well. Because of the reliance on a network for communications, the type, quality, and utilization of the network connection have a significant impact on performance. If the network is unreliable, RPC performance is degraded because of the need to retransmit unreceived packets. If network congestion is detected, the rate of packet transmissions is reduced, which increases the time to complete an RPC. Heterogeneity in the environment is also a factor. For example, an architecture having a byte ordering that is very similar to that of the Network Data Representation requires little data manipulation to achieve the machine-independent data form required. On other machines, however, additional CPU cycles must be spent on marshalling and unmarshalling the data into an appropriate form. The number of threads in the client or server is another factor, since a multi-threaded server typically performs better than a single-threaded one. And the nature of the data sent or received by the application is yet another factor, since small amounts of data sent infrequently place a much smaller load on the network than do large continuous streams of data. Some of these factors have been explored by other researchers, whose work is reviewed briefly in the next section.

4 Related Performance Studies

Performance studies of DCE are becoming more common as DCE becomes more widely available. In particular, RPC has been the focus of several recent studies. The most common measures of performance have been RPC response time (usually referred to as round trip time, or RTT) [3, 2, 6, 7] and RPC throughput [2, 9]. The effect on RPC performance of issues such as RPC data size [3, 2, 6, 7] and multi-threading [3, 6, 7] has also been investigated. Measurements have typically been based on synthetic RPC applications, for two reasons: first, because "real" applications are still quite rare, and, second, because the researchers want to exercise experimental control over parameters such as data size and the number of threads in the application. Most of the synthetic applications were simple data-transfer programs.

Round trip time, the time elapsed between the client's issuing the RPC call and receiving the response from the server, is the most commonly used measure of the speed of RPC service between the client and the server. The relationship between the RPC RTT and RPC data size has been studied in a variety of environments [3, 6, 2]. In all cases, RTT was found to increase with data size, but both the rate and the nature of the increase (linear or non-linear) varied with the environment.

The effects of fragmentation of RPC data were investigated in both [6] and [2]. In particular, fragmentation was used to explain jumps observed in RTT at certain data sizes. Corn *et al.* reported sharp increases in RTT for connection-oriented RPCs with between 2 KB and 4 KB of data, and between 4 KB and 8 KB of data for connectionless RPCs. They concluded that connection-oriented RPC performs better for small calls ($<$ 2 KB of data), while connectionless RPC performs better for larger calls. In fact, the larger the data transferred via a connectionless RPC call, the better the performance. They attributed this difference to the slow-start flow-control policy introduced for connectionless RPC.

5 The Experiments

5.1 Objectives

As mentioned earlier, RPC performance is influenced by a number of factors, including the nature of the DCE application (i.e. the workload), the system activity required to prepare data and present it to the network, and the amount of network traffic. To this point, our investigation has focused on two things: the packet arrival process and the time to complete RPC requests. We have looked at the effects of the packet decomposition process (as described in Figures 3 and 4) and the effects of using connection-oriented versus connectionless RPCs in the application. We have also examined the effects of RPC size and the effects of network traffic, all in the context of OS/2. The specific measures used are described in the next sub-section, which is followed by a description of the experiments themselves.

Two separate experiments have been carried out. The first experiment relates to traffic characterization, specifically the network traffic resulting from DCE RPC and the effect on the characteristics of this network traffic of the protocol used to transport the RPC data (connection-oriented versus connectionless) and design issues within DCE (such as flow-control mechanisms). We examined the interarrival times at various levels of decomposition of the RPC data in order to determine the characteristics at the various levels. It is expected that the protocol processing will induce clusters or bursts of IP packet arrivals, a process we are interested in characterising. We are also interested in the effect of network load on the packet arrival process. It is expected that an increased network load will increase both the mean and variance of interarrival times for the various transmission units.

The second experiment relates more directly to DCE RPC performance, specifically response times. For this experiment, our goal was to examine the RPC round trip time (RTT) for various RPC data sizes under different network loads. An increase in either factor should lead to increased RTT, as has been reported for other DCE implementations, but it is not clear which factor is more important.

5.2 The Performance Metrics

Our results are derived from empirical measurements. The following metrics form the basis of our analysis:

- The packet arrival process (interarrival time[3]):
  - *Inter-IP Time (IP)* is the time between the arrival of consecutive IP packets for RPC communications involving a particular client-server pair.
  - *Inter-PDU Time (PDU)* is the time between the arrival of the first IP packet in one RPC PDU and the arrival of the first IP packet in the next RPC PDU.
  - *Inter-IP Time within a PDU (PDU-IP)* is the time between consecutive IP packets resulting from a single RPC PDU.
- RPC response times:
  - *Round Trip Time (RTT)* is the time between the client's issuing the RPC call and receiving the (entire) response from the server.

5.3 The Experimental Setup

A single DCE cell was used for these experiments. The cell contains a mixture of OS/2 and Digital Unix machines, but all measurements reported in this paper were taken from communications between clients and servers running on the OS/2 machines[4]. The detailed layout is shown in Figure 5. Specifically, the cell contains the following machines, communicating over a 10 Mbps Ethernet:

- The cell server is a DEC Alpha 2100A-500MP with three 190 MHz ALPHA AXP 21064 CPUs and 256 MB of main memory running the Digital Unix operating system and OSF's DCE 1.3 for OSF/1.

---

[3] All measurements of interarrival times are taken from a fixed observation point on the network.

[4] Comunications between platforms are presently being studied as well.

- The cell client running the application server is an IBM Personal System 350 with a Pentium 75 MHz processor and 16 MB of RAM. This client runs IBM's DCE 2.1 Beta for OS/2 on top of IBM's OS/2 Warp Connect operating system.
- The cell client running the application client is an IBM PS/2 Model 80 with a Reply 486dx2-66 motherboard and 16 MB of RAM. This client also runs IBM's DCE 2.1 Beta for OS/2 on top of IBM's OS/2 Warp Connect operating system.
- Two other clients (another DEC Alpha 2100A-500MP with a single CPU, and an IBM ThinkPAD) exist in the cell, but they were not used in the experiments reported in this paper.

The *measurement station* is a SUN SPARCstation IPC, which is not in the DCE cell but is connected to the same Ethernet segment. This machine collects traces of network traffic using the passive network monitoring package *tcpdump*. Care was taken to ensure that neither the tracing nor the saving of the traces introduced any traffic to the network while the experiments were underway. Four separate traces were collected in an attempt to account for the presence of uncontrollable factors, and packet loss was small: on the order of 0.05%. Trace analysis was done off-line.

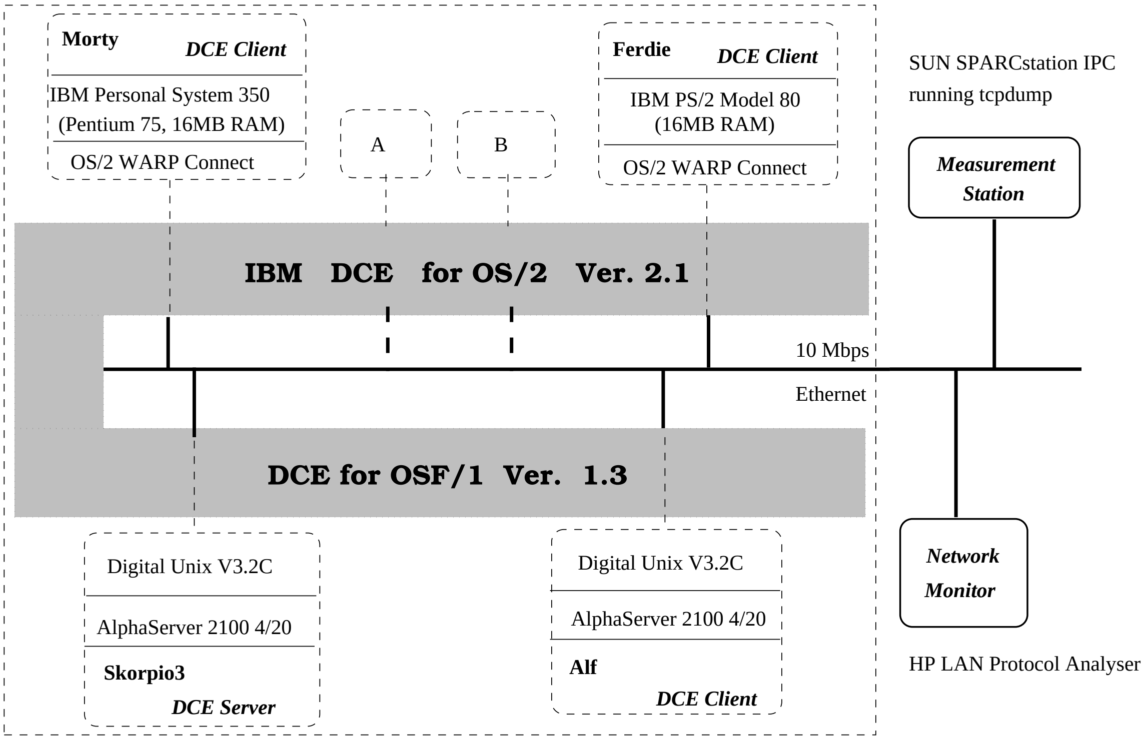


Figure 5: The DCE Cell

The "application" used in the experiments was a simple data transfer program written to move blocks of data between the application client and the application server. The data transferred was simple NULL-terminated character strings of various lengths, and the amount of data transferred for each call ranged from 0 to 24 KB. The application client had a single thread of execution while the application server had a variable number of threads.

The load on the network is also a factor in this experiment, albeit one that is difficult to control (since our machines are part of our departmental network). In order to minimize the effect of network traffic from other sources, all experiments were conducted at off-peak hours. Steps were taken to present two distinct levels of network utilization: a utilization of approximately 10% (which we characterized as a "light" load) and a utilization of approximately 60% (characterized as "heavy"). The additional network load necessary to achieve the heavy load was introduced using an HP 4972A LAN Protocol Analyser, which was also used to measure the utilization at the time the experiment was conducted.

For the first experiment, the DCE application sent RPC data ranging from 0 to 24 KB from the client to the server. The size of the RPC data field was randomly selected from a set of fixed data sizes within this range. Both connection-oriented and connectionless RPCs were performed.

For the second experiment, RPC data sizes of 0, 1000, 1500, 6000 and 10000 bytes were sent (a number of times, in random order) from the client to the server. The 0 sized RPC (or "NULL" RPC) provides an indication of the overhead present in all RPCs, as it contains no data. The other values were chosen to reflect the effects of fragmentation in the following way:

- 1000 bytes of RPC data results in a single IP packet (that is, no PDU fragmentation).
- 1500 bytes of RPC data results in two IP packets, but only one RPC PDU (that is, no call fragmentation).
- 6000 bytes of data results in two RPC PDUs (that is, both call and PDU fragmentation, as illustrated in Figs. 3 and 4).
- 10000 bytes results in three RPC PDUs.

Again, the experiment was performed at off-peak hours to minimize the chances of other traffic on the network affecting our results, and additional load to achieve the different levels of utilization reported was introduced using the LAN Protocol Analyzer.

5.4 The Results and Analysis

Our experiments are ongoing, but the data collected to date shows some interesting things. Summaries of the trace data gathered for the first experiment (means, variances, and coefficients of variation of interarrival times) are given in Tables 1, 2 and 3. There is reasonable consistency across all four traces, suggesting that the effect of uncontrolled factors was small.

Table 1: Mean Interarrival Times with Different RPC Protocols

RPC Protocol	Transmission Unit	Mean Interarrival Times (msec)				
		Trace 1	Trace 2	Trace 3	Trace 4	Average
Connection-Oriented	PDU	8.870	8.840	8.788	8.976	8.868
	IP	3.498	3.485	3.465	3.540	3.497
	PDU-IP	1.341	1.322	1.304	1.354	1.330
Connection-less	PDU	9.906	10.102	9.932	10.002	9.986
	IP	3.906	3.982	3.914	3.942	3.936
	PDU-IP	1.157	1.211	1.150	1.152	1.167

Table 2: Variance of Interarrival Times with Different RPC Protocols

RPC Protocol	Transmission Unit	Variances of Interarrival Times			
		Trace 1	Trace 2	Trace 3	Trace 4
Connection-Oriented	PDU	41.508	41.478	46.952	50.908
	IP	29.103	29.083	30.958	33.073
	PDU-IP	1.771	1.466	3.099	2.286
Connection-less	PDU	31.498	38.376	34.767	35.850
	IP	26.790	29.661	28.243	28.960
	PDU-IP	0.312	1.558	0.284	0.284

Table 3: Coefficient of Variation (CoV) of Interarrival Times with Different RPC Protocols

RPC Protocol	Transmission Unit	CoV of Interarrival Times			
		Trace 1	Trace 2	Trace 3	Trace 4
Connection-Oriented	PDU	0.726	0.729	0.780	0.795
	IP	1.542	1.547	1.606	1.625
	PDU-IP	0.993	0.916	1.350	1.117
Connection-less	PDU	0.567	0.613	0.593	0.598
	IP	1.325	1.368	1.358	1.365
	PDU-IP	0.483	1.031	0.464	0.463

Full distributions of interarrival times at the different levels, for Trace 1, are shown in Figure 6.

The summary data shows several things. As expected, mean interarrival times decrease as you move closer to the Ethernet. In other words, the mean interarrival time for IP packets is less than the mean interarrival time for PDUs. The mean interarrival time for IP packets within PDUs is the lowest of all since the only issue here is the time to put the packet on the network. The other processing is done at higher levels. Variances show a similar trend. Coefficients of variation (CoV) are high for inter-IP times, but low for inter-PDU time. The high values for inter-IP times reflect the fact that some inter-IP times are quite uniform (such as those within a PDU, as reflected by the very low CoV for inter-IP time within PDUs). Others (such as those between PDUs) are more variable. In other words, the IP packets tend to be introduced to the network in PDU-based *bursts*, with relatively constant spacing within a burst.

The effects of the choice of RPC protocol are apparent, and interesting. For connectionless RPC, flow control is done at the PDU level. This means that an entire PDU is presented for transmission, and the individual IP packets comprising it are then placed on the network as quickly as possible. With connection-oriented RPC, on the other hand, the TCP layer governs the speed at which the individual IP packets comprising a PDU are placed on the network. This is reflected in higher values for mean PDU-IP (interarrival times for IP packets comprising a PDU). The larger values for mean interarrival times for PDUs under connectionless RPC can be attributed to the flow-control mechanism, which presents PDUs slowly at first, and more rapidly as acknowledgements are received. In other words, entire PDUs may be held back during the ramping up phase of the slow start flow control mechanism. Flow control in connection-oriented RPC is applied at the level of TCP packets rather than at the level of PDUs. The control of transmission at the level of individual TCP (or IP) packets results in a lower mean interarrival time for IP packets for connection-oriented transmission. Despite the fact that the mean interarrival times are higher for connectionless RPC (for PDU and IP), the variances tend to be lower.

The effect of network load on interarrival times is shown in Tables 4, 5 and 6. Not surprisingly, an increase in network load leads to an increase in mean interarrival times at all levels, and a large increase in variance because of the increased contention for the network.

Figure 6 also shows the effects of the different mechanisms for flow control in connectionless or connection-oriented RPC. As expected, the interarrival times tend to be concentrated in small regions, with some noticeable exceptions. In two of the plots, inter-PDU time and inter-IP time, there is a distinct "second peak" in the case of connectionless RPC. We attribute this once again to the mechanism introduced for flow control. The flow-control mechanism for connection-oriented RPC can split up a PDU and present its individual TCP/IP packets to the network separately, something that is not possible for connectionless RPC, where flow control is managed at the PDU level. This can result in the holding up of IP packets for large data sizes, and thus a second peak in the inter-IP time distribution for connectionless RPC. The presence of the second peak in Figure 6(b), however, is puzzling. Our conjecture is that the flow-control mechanism in OS/2 DCE 2.1 Beta is not allowing the flow control window to grow to hold more than a single PDU. Were this the case, only one PDU at a time could be sent under the connectionless RPC protocol flow-control strategy,

Table 4: Effect of Network Load on Mean Interarrival Times (Connection-Oriented RPC)

Network Load	Transmission Unit	Mean Interarrival Times (msec)				
		Trace 1	Trace 2	Trace 3	Trace 4	Average
Light	PDU	8.870	8.840	8.788	8.976	8.868
	IP	3.498	3.485	3.465	3.540	3.497
	PDU-IP	1.341	1.322	1.304	1.354	1.330
Heavy	PDU	14.973	15.657	12.480	13.955	14.266
	IP	4.852	5.161	4.607	4.857	4.869
	PDU-IP	2.551	2.735	2.281	2.474	2.510

Table 5: Effect of Network Load on Variance of Interarrival Times (Connection-Oriented RPC)

Network Load	Transmission Unit	Variances of Interarrival Times			
		Trace 1	Trace 2	Trace 3	Trace 4
Light	PDU	41.508	41.478	46.952	50.908
	IP	29.103	29.083	30.958	33.073
	PDU-IP	1.771	1.466	3.099	2.286
Heavy	PDU	375.952	427.498	205.955	246.167
	IP	116.232	131.381	85.185	91.226
	PDU-IP	52.536	51.184	28.367	35.044

Table 6: Effect of Network Load on Coefficient of Variation (CoV) of Interarrival Times (Connection-Oriented RPC)

Network Load	Transmission Unit	CoV of Interarrival Times			
		Trace 1	Trace 2	Trace 3	Trace 4
Light	PDU	0.726	0.729	0.780	0.795
	IP	1.542	1.547	1.606	1.625
	PDU-IP	0.993	0.916	1.350	1.117
Heavy	PDU	1.295	1.320	1.150	1.124
	IP	2.222	2.221	2.003	1.966
	PDU-IP	2.842	2.616	2.335	2.393

forcing subsequent PDUs of a multi-PDU RPC to wait for acknowledgement that the previous PDU was successfully transmitted. Since slightly more than half of the PDUs in this experiment are the second or later PDUs of a multi-PDU RPC, given the sizes of the RPC data in the experimental application, these PDUs would be forced to wait. This additional waiting would account for the second peak observed for connectionless RPC PDUs, as well as contribute to the second peak in Figure 6(d). Further investigation of this behaviour is underway.

Finally, the results in Figure 7 show the effects of RPC packet size and network load on the time to service the RPC requests. Not surprisingly, RTT increases with both data size and network load, as has been reported for other environments, but it appears that data size is the more significant factor. Interestingly, the RTT for NULL RPCs appears to be insensitive to the network load. We are continuing this part of our investigation as well.

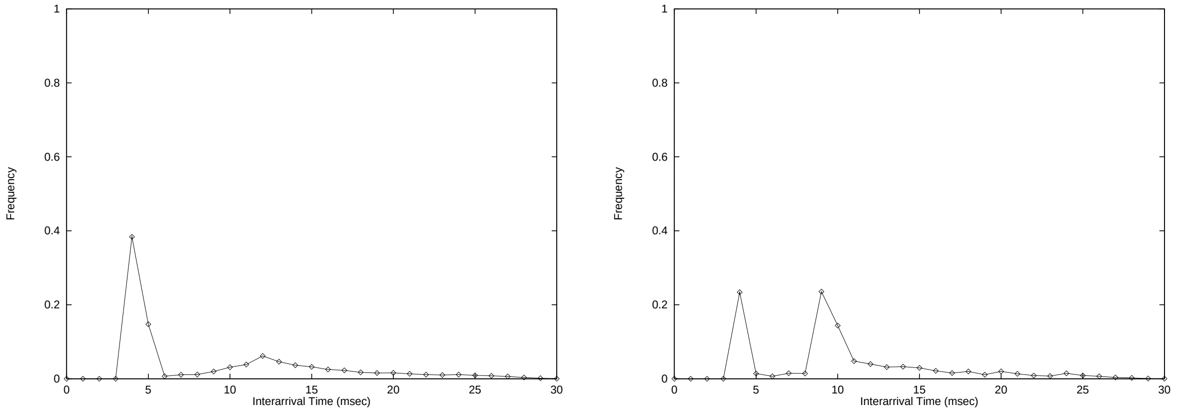
6 Conclusions

We are conducting a series of measurement experiments to increase our understanding of DCE RPC in an OS/2 environment. Our focus is on issues relating to performance, specifically the characterization of network traffic and RPC response time. In general, the results obtained to this point for the OS/2 environment are consistent with those obtained in other DCE implementations and also identify some issues specific to OS/2. Thus we feel they provide a useful update to the growing body of quantitative data on DCE. Of particular concern are the results pertaining to connectionless versus connection-oriented RPC, which suggest an implementation bug. These results provide motivation for a deeper study of this factor in the OS/2 environment.

We intend to apply the results of our characterization of the packet arrival process to the modelling of DCE network traffic. It is clear from these results that a multi-level traffic model, such as was presented by Ling *et al.* [8], is necessary to capture the packet bursts resulting from the decomposition of RPC calls for network transport. A simple-minded model that assumes independent arrivals of IP packets will not suffice. This is an important issue in workload modelling and similar work is underway in other domains [12].

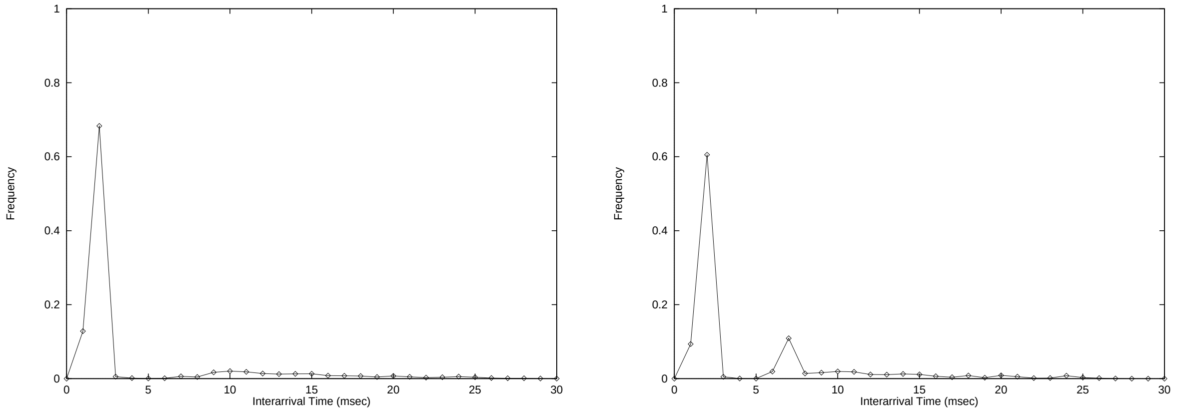
7 Acknowledgements

Support for this research has been provided by IBM Canada and by the Natural Sciences and Engineering Research Council of Canada (NSERC), through Research Grant OGP0003707 and Cooperative Activities Program Grant CAP171120.



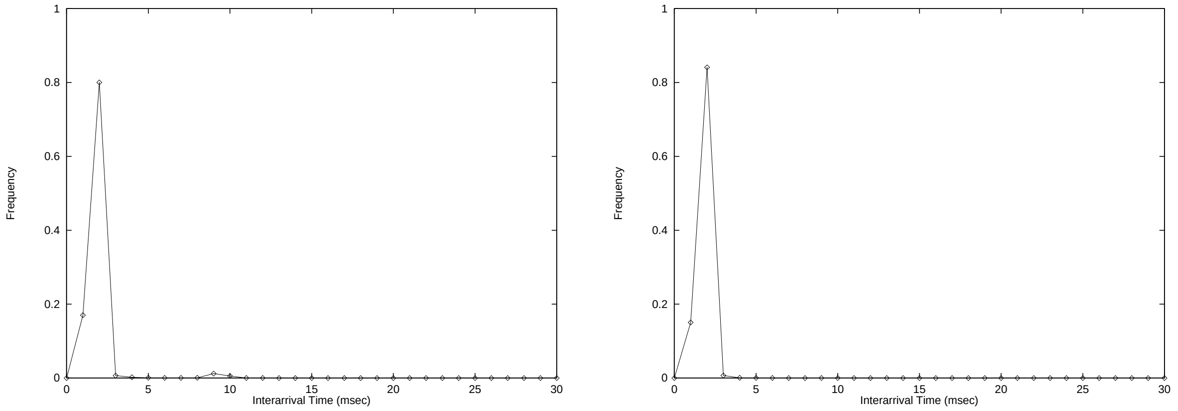
(a) The Inter-PDU Time Distribution (RPC Over TCP)

(b) The Inter-PDU Time Distribution (RPC Over UDP)



(c) The Inter-IP Time Distribution (RPC Over TCP)

(d)The Inter-IP Time Distribution (RPC Over UDP)



(e) The Inter-PDU-IP Time Distribution (RPC Over TCP)

(f) The Inter-PDU-IP Time Distribution (RPC Over UDP)

Figure 6: Interarrival Time Distributions (Trace 1, 10% Network Utilization)

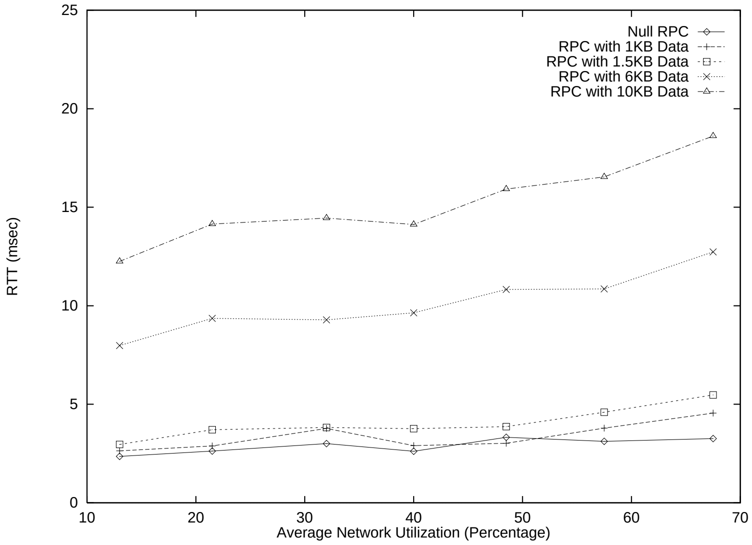


Figure 7: The Effect of Data Size and Network Utilization on RPC Round Trip Time

References

[1] A.D. Birrell and B.J. Nelson, "Implementing Remote Procedure Calls", *ACM Transactions on Computer Systems*, Vol. 2, No. 1, February 1984, pages 39-59.

[2] C. Corn, T. Li, R. Pekowski, and B. Santeford, "Performance of Key Functions in DCE for OS/2", *Personal Systems*, January/February 1994, pages 60-70.

[3] B. Dasarathy, K. Khalil, and D.E. Ruddock, "Some DCE Performance Results", *Proc. International DCE Workshop*, University of Karlsruhe, Germany, 1993, pages 47-62.

[4] International Business Machines Corporation, *Distributed Computing Environment: Understanding the Concepts*, Form Number GC09-1478-00, IBM Canada Ltd. Laboratory, Toronto, 1993.

[5] V. Jacobson, "Congestion Avoidance and Control", *Proc. ACM SIGCOMM '88 Conference*, Stanford, California, August 1988, pages 314-329.

[6] A.M. Khandker, P. Honeyman, and T.J. Teorey, "Performance of DCE RPC", *Proc. Second International Workshop on Services in Distributed and Networked Environments*, Whistler, British Columbia, June 1995, pages 2-10.

[7] A.M. Khandker and T.J. Teorey, "Case Study: How Analytic Modeling Can Reveal Performance Problems in Distributed Systems", *Proc. Third International Workshop on Services in Distributed and Networked Environments*, Macau, June 1996.

[8] S.H. Ling, R.B. Bunt, and D.L. Eager, "Characterizing Client-Server Workload on an Ethernet", *Proc. CMG '91 Conference,* Nashville, Tennessee, December 1991, pages 59-69.

[9] B. Perry and B. Russell, "IBM DCE for OS/2 Multiuser Application Performance", *Personal Systems*, January/February 1994, pages 50-58.

[10] W. Rosenberry, D. Kenney, and G. Fisher, *Understanding DCE*, O'Reilly & Associates, 1992.

[11] A.S. Tannenbaum, *Distributed Operating Systems*, Prentice Hall, 1995.

[12] P.P. Ware, T.W. Page Jr., and B.L. Nelson, "Modeling a Two-Level Arrival Process, with Applications to Computer Performance Simulation", unpublished manuscript.