R. Srinivasan
Sun Microsystems

August 1995

# RPC: Remote Procedure Call Protocol Specification Version 2

## Status of this Memo

This document specifies an Internet standards track protocol for the Internet community, and requests discussion and suggestions for improvements. Please refer to the current edition of the "Internet Official Protocol Standards" (STD 1) for the standardization state and status of this protocol. Distribution of this memo is unlimited.

## ABSTRACT

This document describes the ONC Remote Procedure Call (ONC RPC Version 2) protocol as it is currently deployed and accepted. "ONC" stands for "Open Network Computing".

## TABLE OF CONTENTS

1. INTRODUCTION 2
2. TERMINOLOGY 2
3. THE RPC MODEL 2
4. TRANSPORTS AND SEMANTICS 4
5. BINDING AND RENDEZVOUS INDEPENDENCE 5
6. AUTHENTICATION 5
7. RPC PROTOCOL REQUIREMENTS 5
7.1 RPC Programs and Procedures 6
7.2 Authentication 7
7.3 Program Number Assignment 8
7.4 Other Uses of the RPC Protocol 8
7.4.1 Batching 8
7.4.2 Broadcast Remote Procedure Calls 8
8. THE RPC MESSAGE PROTOCOL 9
9. AUTHENTICATION PROTOCOLS 12
9.1 Null Authentication 13
10. RECORD MARKING STANDARD 13
11. THE RPC LANGUAGE 13
11.1 An Example Service Described in the RPC Language 13
11.2 The RPC Language Specification 14
11.3 Syntax Notes 15
APPENDIX A: SYSTEM AUTHENTICATION 16
REFERENCES 17
Security Considerations 18
Author's Address 18

## 1. INTRODUCTION

This document specifies version two of the message protocol used in ONC Remote Procedure Call (RPC). The message protocol is specified with the eXternal Data Representation (XDR) language [9]. This document assumes that the reader is familiar with XDR. It does not attempt to justify remote procedure calls systems or describe their use. The paper by Birrell and Nelson [1] is recommended as an excellent background for the remote procedure call concept.

## 2. TERMINOLOGY

This document discusses clients, calls, servers, replies, services, programs, procedures, and versions. Each remote procedure call has two sides: an active client side that makes the call to a server, which sends back a reply. A network service is a collection of one or more remote programs. A remote program implements one or more remote procedures; the procedures, their parameters, and results are documented in the specific program's protocol specification. A server may support more than one version of a remote program in order to be compatible with changing protocols.

For example, a network file service may be composed of two programs. One program may deal with high-level applications such as file system access control and locking. The other may deal with low-level file input and output and have procedures like "read" and "write". A client of the network file service would call the procedures associated with the two programs of the service on behalf of the client.

The terms client and server only apply to a particular transaction; a particular hardware entity (host) or software entity (process or program) could operate in both roles at different times. For example, a program that supplies remote execution service could also be a client of a network file service.

## 3. THE RPC MODEL

The ONC RPC protocol is based on the remote procedure call model, which is similar to the local procedure call model. In the local case, the caller places arguments to a procedure in some well-specified location (such as a register window). It then transfers control to the procedure, and eventually regains control. At that point, the results of the procedure are extracted from the well-specified location, and the caller continues execution.

The remote procedure call model is similar. One thread of control logically winds through two processes: the caller's process, and a server's process. The caller process first sends a call message to the server process and waits (blocks) for a reply message. The call message includes the procedure's parameters, and the reply message includes the procedure's results. Once the reply message is received, the results of the procedure are extracted, and caller's execution is resumed.

On the server side, a process is dormant awaiting the arrival of a call message. When one arrives, the server process extracts the procedure's parameters, computes the results, sends a reply message, and then awaits the next call message.

In this model, only one of the two processes is active at any given time. However, this model is only given as an example. The ONC RPC protocol makes no restrictions on the concurrency model implemented, and others are possible. For example, an implementation may choose to have RPC calls be asynchronous, so that the client may do useful work while waiting for the reply from the server. Another possibility is to have the server create a separate task to process an incoming call, so that the original server can be free to receive other requests.

There are a few important ways in which remote procedure calls differ from local procedure calls:

1. Error handling: failures of the remote server or network must be handled when using remote procedure calls.

2. Global variables and side-effects: since the server does not have access to the client's address space, hidden arguments cannot be passed as global variables or returned as side effects.

3. Performance: remote procedures usually operate one or more orders of magnitude slower than local procedure calls.

4. Authentication: since remote procedure calls can be transported over unsecured networks, authentication may be necessary. Authentication prevents one entity from masquerading as some other entity.

The conclusion is that even though there are tools to automatically generate client and server libraries for a given service, protocols must still be designed carefully.

## 4. TRANSPORTS AND SEMANTICS

The RPC protocol can be implemented on several different transport protocols. The RPC protocol does not care how a message is passed from one process to another, but only with specification and interpretation of messages. However, the application may wish to obtain information about (and perhaps control over) the transport layer through an interface not specified in this document. For example, the transport protocol may impose a restriction on the maximum size of RPC messages, or it may be stream-oriented like TCP with no size limit. The client and server must agree on their transport protocol choices.

It is important to point out that RPC does not try to implement any kind of reliability and that the application may need to be aware of the type of transport protocol underneath RPC. If it knows it is running on top of a reliable transport such as TCP [6], then most of the work is already done for it. On the other hand, if it is running on top of an unreliable transport such as UDP [7], it must implement its own time-out, retransmission, and duplicate detection policies as the RPC protocol does not provide these services.

Because of transport independence, the RPC protocol does not attach specific semantics to the remote procedures or their execution requirements. Semantics can be inferred from (but should be explicitly specified by) the underlying transport protocol. For example, consider RPC running on top of an unreliable transport such as UDP. If an application retransmits RPC call messages after time-outs, and does not receive a reply, it cannot infer anything about the number of times the procedure was executed. If it does receive a reply, then it can infer that the procedure was executed at least once.

A server may wish to remember previously granted requests from a client and not regrant them in order to insure some degree of execute-at-most-once semantics. A server can do this by taking advantage of the transaction ID that is packaged with every RPC message. The main use of this transaction ID is by the client RPC entity in matching replies to calls. However, a client application may choose to reuse its previous transaction ID when retransmitting a call. The server may choose to remember this ID after executing a call and not execute calls with the same ID in order to achieve some degree of execute-at-most-once semantics. The server is not allowed to examine this ID in any other way except as a test for equality.

On the other hand, if using a "reliable" transport such as TCP, the application can infer from a reply message that the procedure was executed exactly once, but if it receives no reply message, it cannot

assume that the remote procedure was not executed. Note that even if a connection-oriented protocol like TCP is used, an application still needs time-outs and reconnection to handle server crashes.

There are other possibilities for transports besides datagram- or connection-oriented protocols. For example, a request-reply protocol such as VMTP [2] is perhaps a natural transport for RPC. ONC RPC uses both TCP and UDP transport protocols. Section 10 (RECORD MARKING STANDARD) describes the mechanism employed by ONC RPC to utilize a connection-oriented, stream-oriented transport such as TCP.

## 5. BINDING AND RENDEZVOUS INDEPENDENCE

The act of binding a particular client to a particular service and transport parameters is NOT part of this RPC protocol specification. This important and necessary function is left up to some higher-level software.

Implementors could think of the RPC protocol as the jump-subroutine instruction ("JSR") of a network; the loader (binder) makes JSR useful, and the loader itself uses JSR to accomplish its task. Likewise, the binding software makes RPC useful, possibly using RPC to accomplish this task.

## 6. AUTHENTICATION

The RPC protocol provides the fields necessary for a client to identify itself to a service, and vice-versa, in each call and reply message. Security and access control mechanisms can be built on top of this message authentication. Several different authentication protocols can be supported. A field in the RPC header indicates which protocol is being used. More information on specific authentication protocols is in section 9: "Authentication Protocols".

## 7. RPC PROTOCOL REQUIREMENTS

The RPC protocol must provide for the following:

(1) Unique specification of a procedure to be called.
(2) Provisions for matching response messages to request messages.
(3) Provisions for authenticating the caller to service and vice-versa.

DOCKET ALARM

# Explore Litigation Insights

Docket Alarm provides insights to develop a more informed litigation strategy and the peace of mind of knowing you're on top of things.

## Real-Time Litigation Alerts

Keep your litigation team up-to-date with **real-time alerts** and advanced team management tools built for the enterprise, all while greatly reducing PACER spend.

Our comprehensive service means we can handle Federal, State, and Administrative courts across the country.

## Advanced Docket Research

With over 230 million records, Docket Alarm's cloud-native docket research platform finds what other services can't. Coverage includes Federal, State, plus PTAB, TTAB, ITC and NLRB decisions, all in one place.

Identify arguments that have been successful in the past with full text, pinpoint searching. Link to case law cited within any court document via Fastcase.

## Analytics At Your Fingertips

Learn what happened the last time a particular judge, opposing counsel or company faced cases similar to yours.

Advanced out-of-the-box PTAB and TTAB analytics are always at your fingertips.

## API

Docket Alarm offers a powerful API (application programming interface) to developers that want to integrate case filings into their apps.

**LAW FIRMS**
Build custom dashboards for your attorneys and clients with live data direct from the court.

Automate many repetitive legal tasks like conflict checks, document management, and marketing.

**FINANCIAL INSTITUTIONS**
Litigation and bankruptcy checks for companies and debtors.

**E-DISCOVERY AND LEGAL VENDORS**
Sync your system to PACER to automate legal marketing.