Network Working Group
Request for Comments: 1833
Category: Standards Track

R. Srinivasan
Sun Microsystems

August 1995

Binding Protocols for ONC RPC Version 2

Status of this Memo

ABSTRACT

This document describes the binding protocols used in conjunction with the ONC Remote Procedure Call (ONC RPC Version 2) protocols.

TABLE OF CONTENTS

1. Introduction

This document specifies the binding protocols used in conjunction with ONC RPC Version 2. As a prerequisite, the reader is expected to be familiar with [1] and [2] which describe the ONC RPC Version 2 and XDR (eXternal Data Representation) protocols.

An RPC service is identified by its RPC program number, version number, and the transport address where it may be reached. The transport address, in turn, consists of a network address and a transport selector. In the case of a service available over TCP/IP or UDP/IP, the network address will be an IP address, and the transport selector will be a TCP or UDP port number.

A client program needs to know the RPC program number, version number, and the transport address corresponding to a service in order to utilize the service. Of these, the RPC program number and version number are usually built into the client program, as part of the service definition. The network address component of the transport address is usually available in a name service, or is given as a parameter to the client program. The transport selector (ie., the TCP or UDP port) is usually determined dynamically, and varies with each invocation of the service. Server programs allocate a transport address, and register it with a well-known lookup service (well-known because it uses a fixed transport selector, and resides at the same network address as the server). Client programs consult the lookup service in order to obtain the server's transport address.

Such a lookup service is very desirable because the range of well-known transport selectors is very small for some transports and the number of services is potentially very large. By running only the lookup service on a well-known transport selector, the transport addresses of other remote programs can be ascertained by querying the lookup service.

This document describes three versions of a lookup service, all of which use the same RPC program number (100000). They all use port 111 over TCP and UDP transports. Versions 3 and 4 are described in Section 2 ("RPCBIND Program Protocol"). Version 2 is described in Section 3 ("Port Mapper Program Protocol").

The distinguishing characteristic of RPCBIND (versions 3 and 4) is that this protocol uses a transport-independent format for the transport address, known as the universal address format. An address in universal address format is an ASCII string representation of the transport dependent address. String representation of addresses corresponding to a transport are defined by the addressing authority for the transport. The RPCBIND protocol can be used for binding ONC RPC clients and servers over any transport.

The Port Mapper (version 2), on the other hand, is an older protocol that is specific to TCP and UDP. It handles TCP and UDP ports directly.

2. RPCBIND Program Protocol

The RPCBIND program maps RPC program and version numbers to universal addresses, thus making dynamic binding of remote programs possible.

The RPCBIND program is bound to a well-known address of each supported transport, and other programs register their dynamically allocated transport address with it. The RPCBIND program then makes

those addresses publicly available.

The RPCBIND program also aids in broadcast RPC. A given RPC program will usually have different transport address bindings on different machines, so there is no way to directly broadcast to all of these programs. The RPCBIND program, however, does have a well-known address. So, to broadcast to a given program, the client actually sends its message to the RPCBIND program located at the broadcast address. Each instance of the RPCBIND program that picks up the broadcast then calls the local service specified by the client. When the RPCBIND program gets the reply from the local service, it sends the reply back to the client.

2.1 RPCBIND Protocol Specification (in RPC Language)

```
/*
 * rpcb_prot.x
 * rpcbind protocol, versions 3 and 4, in RPC Language
 */

/*
 * rpcbind address for TCP/UDP
 */
const RPCB_PORT = 111;

/*
 * A mapping of (program, version, network ID) to address
 *
 * The network identifier  (r_netid):
 * This is a string that represents a local identification for a
 * network. This is defined by a system administrator based on local
 * conventions, and cannot be depended on to have the same value on
 * every system.
 */
struct rpcb {
 unsigned long r_prog;    /* program number */
 unsigned long r_vers;    /* version number */
 string r_netid<>;        /* network id */
 string r_addr<>;         /* universal address */
 string r_owner<>;        /* owner of this service */
};

struct rp__list {
 rpcb rpcb_map;
 struct rp__list *rpcb_next;
};
```

```
typedef rp__list *rpcblist_ptr;        /* results of RPCBPROC_DUMP */


/*
 * Arguments of remote calls
 */
struct rpcb_rmtcallargs {
 unsigned long prog;         /* program number */
 unsigned long vers;         /* version number */
 unsigned long proc;         /* procedure number */
 opaque args<>;              /* argument */
};


/*
 * Results of the remote call
 */
struct rpcb_rmtcallres {
 string addr<>;             /* remote universal address */
 opaque results<>;          /* result */
};


/*
 * rpcb_entry contains a merged address of a service on a particular
 * transport, plus associated netconfig information.  A list of
 * rpcb_entry items is returned by RPCBPROC_GETADDRLIST.  The meanings
 * and values used for the r_nc_* fields are given below.
 *
 * The network identifier  (r_nc_netid):

 *   This is a string that represents a local identification for a
 *   network.  This is defined by a system administrator based on
 *   local conventions, and cannot be depended on to have the same
 *   value on every system.
 *
 * Transport semantics (r_nc_semantics):
 *  This represents the type of transport, and has the following values:
 *     NC_TPI_CLTS     (1)      Connectionless
 *     NC_TPI_COTS     (2)      Connection oriented
 *     NC_TPI_COTS_ORD (3)      Connection oriented with graceful close
 *     NC_TPI_RAW      (4)      Raw transport
 *
 * Protocol family (r_nc_protofmly):
 *   This identifies the family to which the protocol belongs.  The
 *   following values are defined:
 *     NC_NOPROTOFMLY   "-"
 *     NC_LOOPBACK      "loopback"
```

```
 *     NC_INET          "inet"
 *     NC_IMPLINK       "implink"
 *     NC_PUP           "pup"
 *     NC_CHAOS         "chaos"
 *     NC_NS            "ns"
 *     NC_NBS           "nbs"
 *     NC_ECMA          "ecma"
 *     NC_DATAKIT       "datakit"
 *     NC_CCITT         "ccitt"
 *     NC_SNA           "sna"
 *     NC_DECNET        "decnet"
 *     NC_DLI           "dli"
 *     NC_LAT           "lat"
 *     NC_HYLINK        "hylink"
 *     NC_APPLETALK     "appletalk"
 *     NC_NIT           "nit"
 *     NC_IEEE802       "ieee802"
 *     NC_OSI           "osi"
 *     NC_X25           "x25"
 *     NC_OSINET        "osinet"
 *     NC_GOSIP         "gosip"
 *
 * Protocol name (r_nc_proto):
 *   This identifies a protocol within a family.  The following are
 *   currently defined:
 *      NC_NOPROTO      "-"
 *      NC_TCP          "tcp"
 *      NC_UDP          "udp"
 *      NC_ICMP         "icmp"
 */
struct rpcb_entry {
 string          r_maddr<>;            /* merged address of service */
 string          r_nc_netid<>;         /* netid field */
 unsigned long   r_nc_semantics;       /* semantics of transport */
 string          r_nc_protofmly<>;     /* protocol family */
 string          r_nc_proto<>;         /* protocol name */
};

/*
 * A list of addresses supported by a service.
 */
struct rpcb_entry_list {
 rpcb_entry rpcb_entry_map;
 struct rpcb_entry_list *rpcb_entry_next;
};

typedef rpcb_entry_list *rpcb_entry_list_ptr;
```

```
/*
 * rpcbind statistics
 */

const rpcb_highproc_2 = RPCBPROC_CALLIT;
const rpcb_highproc_3 = RPCBPROC_TADDR2UADDR;
const rpcb_highproc_4 = RPCBPROC_GETSTAT;

const RPCBSTAT_HIGHPROC = 13; /* # of procs in rpcbind V4 plus one */
const RPCBVERS_STAT     = 3; /* provide only for rpcbind V2, V3 and V4 */
const RPCBVERS_4_STAT   = 2;
const RPCBVERS_3_STAT   = 1;
const RPCBVERS_2_STAT   = 0;

/* Link list of all the stats about getport and getaddr */
struct rpcbs_addrlist {
 unsigned long prog;
 unsigned long vers;
 int success;
 int failure;
 string netid<>;
 struct rpcbs_addrlist *next;
};

/* Link list of all the stats about rmtcall */
struct rpcbs_rmtcalllist {
 unsigned long prog;
 unsigned long vers;
 unsigned long proc;
 int success;
 int failure;
 int indirect;    /* whether callit or indirect */
 string netid<>;
 struct rpcbs_rmtcalllist *next;
};

typedef int rpcbs_proc[RPCBSTAT_HIGHPROC];
typedef rpcbs_addrlist *rpcbs_addrlist_ptr;
typedef rpcbs_rmtcalllist *rpcbs_rmtcalllist_ptr;

struct rpcb_stat {
 rpcbs_proc              info;
 int                     setinfo;
 int                     unsetinfo;
 rpcbs_addrlist_ptr      addrinfo;
 rpcbs_rmtcalllist_ptr   rmtinfo;
};
```

```
/*
 * One rpcb_stat structure is returned for each version of rpcbind
 * being monitored.
 */

typedef rpcb_stat rpcb_stat_byvers[RPCBVERS_STAT];

/*
 * netbuf structure, used to store the transport specific form of
 * a universal transport address.
 */
struct netbuf {
 unsigned int maxlen;
 opaque buf<>;
};


/*
 * rpcbind procedures
 */
program RPCBPROG {
 version RPCBVERS {
     bool
     RPCBPROC_SET(rpcb) = 1;

     bool
     RPCBPROC_UNSET(rpcb) = 2;

     string
     RPCBPROC_GETADDR(rpcb) = 3;

     rpcblist_ptr
     RPCBPROC_DUMP(void) = 4;

     rpcb_rmtcallres
     RPCBPROC_CALLIT(rpcb_rmtcallargs) = 5;

     unsigned int
     RPCBPROC_GETTIME(void) = 6;

     netbuf
     RPCBPROC_UADDR2TADDR(string) = 7;

     string
     RPCBPROC_TADDR2UADDR(netbuf) = 8;
 } = 3;
```

```
 version RPCBVERS4 {
     bool
     RPCBPROC_SET(rpcb) = 1;

     bool
     RPCBPROC_UNSET(rpcb) = 2;

     string
     RPCBPROC_GETADDR(rpcb) = 3;

     rpcblist_ptr
     RPCBPROC_DUMP(void) = 4;

     /*
      * NOTE: RPCBPROC_BCAST has the same functionality as CALLIT;
      * the new name is intended to indicate that this
      * procedure should be used for broadcast RPC, and
      * RPCBPROC_INDIRECT should be used for indirect calls.
      */
     rpcb_rmtcallres
     RPCBPROC_BCAST(rpcb_rmtcallargs) = RPCBPROC_CALLIT;

     unsigned int

     RPCBPROC_GETTIME(void) = 6;

     netbuf
     RPCBPROC_UADDR2TADDR(string) = 7;

     string
     RPCBPROC_TADDR2UADDR(netbuf) = 8;

     string
     RPCBPROC_GETVERSADDR(rpcb) = 9;

     rpcb_rmtcallres
     RPCBPROC_INDIRECT(rpcb_rmtcallargs) = 10;

     rpcb_entry_list_ptr
     RPCBPROC_GETADDRLIST(rpcb) = 11;

     rpcb_stat_byvers
     RPCBPROC_GETSTAT(void) = 12;
 } = 4;
} = 100000;
```

2.2 RPCBIND Operation

RPCBIND is contacted by way of an assigned address specific to the transport being used. For TCP/IP and UDP/IP, for example, it is port number 111. Each transport has such an assigned, well-known address. The following is a description of each of the procedures supported by RPCBIND.

2.2.1 RPCBIND Version 3

RPCBPROC_SET:

When a program first becomes available on a machine, it registers itself with RPCBIND running on the same machine. The program passes its program number "r_prog", version number "r_vers", network identifier "r_netid", universal address "r_addr", and the owner of the service "r_owner". The procedure returns a boolean response whose value is TRUE if the procedure successfully established the mapping and FALSE otherwise. The procedure refuses to establish a mapping if one already exists for the ordered set ("r_prog", "r_vers", "r_netid"). Note that neither "r_netid" nor "r_addr" can be NULL, and that "r_netid" should be a valid network identifier on the machine making the call.

RPCBPROC_UNSET:

When a program becomes unavailable, it should unregister itself with the RPCBIND program on the same machine. The parameters and results have meanings identical to those of RPCBPROC_SET. The mapping of the ("r_prog", "r_vers", "r_netid") tuple with "r_addr" is deleted. If "r_netid" is NULL, all mappings specified by the ordered set ("r_prog", "r_vers", *) and the corresponding universal addresses are deleted. Only the owner of the service or the super-user is allowed to unset a service.

RPCBPROC_GETADDR:

Given a program number "r_prog", version number "r_vers", and network identifier "r_netid", this procedure returns the universal address on which the program is awaiting call requests. The "r_netid" field of the argument is ignored and the "r_netid" is inferred from the network identifier of the transport on which the request came in.

RPCBPROC_DUMP:

This procedure lists all entries in RPCBIND's database. The procedure takes no parameters and returns a list of program, version, network identifier, and universal addresses.

RPCBPROC_CALLIT:

This procedure allows a caller to call another remote procedure on the same machine without knowing the remote procedure's universal address. It is intended for supporting broadcasts to arbitrary remote programs via RPCBIND's universal address. The parameters "prog", "vers", "proc", and args are the program number, version number, procedure number, and parameters of the remote procedure.

Note - This procedure only sends a response if the procedure was successfully executed and is silent (no response) otherwise.

The procedure returns the remote program's universal address, and the results of the remote procedure.

RPCBPROC_GETTIME:

This procedure returns the local time on its own machine in seconds since the midnight of the First day of January, 1970.

RPCBPROC_UADDR2TADDR:

This procedure converts universal addresses to transport specific addresses.

RPCBPROC_TADDR2UADDR:

This procedure converts transport specific addresses to universal addresses.

2.2.2 RPCBIND, Version 4

Version 4 of the RPCBIND protocol includes all of the above procedures, and adds several additional ones.

RPCBPROC_BCAST:

This procedure is identical to the version 3 RPCBPROC_CALLIT procedure. The new name indicates that the procedure should be used for broadcast RPCs only. RPCBPROC_INDIRECT, defined below, should be used for indirect RPC calls.

RPCBPROC_GETVERSADDR:

This procedure is similar to RPCBPROC_GETADDR. The difference is the "r_vers" field of the rpcb structure can be used to specify the version of interest. If that version is not registered, no address is returned.

RPCBPROC_INDIRECT:

Similar to RPCBPROC_CALLIT. Instead of being silent about errors (such as the program not being registered on the system), this procedure returns an indication of the error. This procedure should not be used for broadcast RPC. It is intended to be used with indirect RPC calls only.

RPCBPROC_GETADDRLIST:

This procedure returns a list of addresses for the given rpcb entry. The client may be able use the results to determine alternate transports that it can use to communicate with the server.

RPCBPROC_GETSTAT:

This procedure returns statistics on the activity of the RPCBIND server. The information lists the number and kind of requests the server has received.

Note - All procedures except RPCBPROC_SET and RPCBPROC_UNSET can be called by clients running on a machine other than a machine on which RPCBIND is running. RPCBIND only accepts RPCBPROC_SET and RPCBPROC_UNSET requests by clients running on the same machine as the RPCBIND program.

3. Port Mapper Program Protocol

The port mapper program maps RPC program and version numbers to transport- specific port numbers. This program makes dynamic binding of remote programs possible. The port mapper protocol differs from the newer RPCBIND protocols in that it is transport specific in its address handling.

3.1 Port Mapper Protocol Specification (in RPC Language)

```
const PMAP_PORT = 111;      /* portmapper port number */
```

A mapping of (program, version, protocol) to port number:

```
struct mapping {
   unsigned int prog;
   unsigned int vers;
   unsigned int prot;
   unsigned int port;
};
```

Supported values for the "prot" field:

```
const IPPROTO_TCP = 6;      /* protocol number for TCP/IP */
const IPPROTO_UDP = 17;     /* protocol number for UDP/IP */
```

A list of mappings:

```
struct *pmaplist {
   mapping map;
   pmaplist next;
};
```

Arguments to callit:

```
struct call_args {
   unsigned int prog;
   unsigned int vers;
   unsigned int proc;
   opaque args<>;
};
```

Results of callit:

```
struct call_result {
   unsigned int port;
   opaque res<>;
};
```

Port mapper procedures:

```
program PMAP_PROG {
   version PMAP_VERS {
      void
      PMAPPROC_NULL(void)         = 0;

      bool
      PMAPPROC_SET(mapping)       = 1;

      bool
      PMAPPROC_UNSET(mapping)     = 2;

      unsigned int
      PMAPPROC_GETPORT(mapping)   = 3;

      pmaplist
      PMAPPROC_DUMP(void)         = 4;

      call_result
```

```
            PMAPPROC_CALLIT(call_args)  = 5;
         } = 2;
      } = 100000;
```

3.2 Port Mapper Operation

The portmapper program currently supports two protocols (UDP and TCP). The portmapper is contacted by talking to it on assigned port number 111 (SUNRPC) on either of these protocols.

The following is a description of each of the portmapper procedures:

PMAPPROC_NULL:

This procedure does no work. By convention, procedure zero of any protocol takes no parameters and returns no results.

PMAPPROC_SET:

When a program first becomes available on a machine, it registers itself with the port mapper program on the same machine. The program passes its program number "prog", version number "vers", transport protocol number "prot", and the port "port" on which it awaits service request. The procedure returns a boolean reply whose value is "TRUE" if the procedure successfully established the mapping and "FALSE" otherwise. The procedure refuses to establish a mapping if one already exists for the tuple "(prog, vers, prot)".

PMAPPROC_UNSET:

When a program becomes unavailable, it should unregister itself with the port mapper program on the same machine. The parameters and results have meanings identical to those of "PMAPPROC_SET". The protocol and port number fields of the argument are ignored.

PMAPPROC_GETPORT:

Given a program number "prog", version number "vers", and transport protocol number "prot", this procedure returns the port number on which the program is awaiting call requests. A port value of zeros means the program has not been registered. The "port" field of the argument is ignored.

PMAPPROC_DUMP:

This procedure enumerates all entries in the port mapper's database. The procedure takes no parameters and returns a list of program, version, protocol, and port values.

PMAPPROC_CALLIT:

This procedure allows a client to call another remote procedure on the same machine without knowing the remote procedure's port number. It is intended for supporting broadcasts to arbitrary remote programs via the well-known port mapper's port. The parameters "prog", "vers", "proc", and the bytes of "args" are the program number, version number, procedure number, and parameters of the remote procedure. Note:

(1) This procedure only sends a reply if the procedure was successfully executed and is silent (no reply) otherwise.

(2) The port mapper communicates with the remote program using UDP only.

The procedure returns the remote program's port number, and the reply is the reply of the remote procedure.

References

[1] Srinivasan, R., "Remote Procedure Call Protocol Version 2", RFC 1831, Sun Microsystems, Inc., August 1995.

[2] Srinivasan, R., "XDR: External Data Representation Standard", RFC 1832, Sun Microsystems, Inc., August 1995.

Security Considerations

Security issues are not discussed in this memo.

Author's Address

Raj Srinivasan
Sun Microsystems, Inc.
ONC Technologies
2550 Garcia Avenue
M/S MTV-5-40
Mountain View, CA 94043
USA

Phone: 415-336-2478
Fax: 415-336-6015
EMail: raj@eng.sun.com