RPC Message Protocol [4]

This section defines the RPC message protocol in the External Data Representation (XDR) language.

Call and reply

4.1

The protocol begins with a call and reply, as follows:

```
enum msg type {
   CALL = 0,
   REPLY = 1
};
```

A reply to a call message indicates that the message was either accepted or rejected, as follows:

```
enum reply_stat {
MSG_ACCEPTED = 0,
MSG_DENIED = 1
};
```

If the call message was accepted, the status of an attempt to call a remote procedure is as follows:

```
enum accept_stat {
SUCCESS = 0,   /* RPC executed successfully */
PROG_UNAVAIL = 1, /* remote hasn't exported program */
PROG_MISMATCH= 2, /* remote can't support version # */
PROC_UNAVAIL = 3, /* program can't support procedure */
GARBAGE_ARGS = 4, /* procedure can't decode params */
SYSTEM_ERR = 5 /* failure in RPC system */
}
```

The following list gives reasons for a call message rejection:

```
enum reject_stat {
RPC_MISMATCH = 0,   /* RPC version number != 2 */
AUTH_ERROR = 1      /* Remote cannot authenticate caller */
};
```

The following list gives reasons for authentication failure:

```
enum auth_stat {
AUTH_BADCRED = 1,      /* Bad credentials (seal broken) */
AUTH_REJECTEDCRED=2,   /* Client must begin new session */
AUTH_BADVERF = 3,      /* Bad verifier (seal broken) */
AUTH_REJECTEDVERF=4,   /* Verifier expired or replayed */
AUTH_TOOWEAK = 5,      /* Rejected for security reasons */
AUTH_INVALIDRESP = 6,  /* Rejected because of bad response verifier */
AUTH_FAILED = 7        /* Failed for other (unknown) reason */
};
```

Message structure

4.2

All messages start with a transaction identifier, `xid`, followed by a two-armed, discriminated union. The union's discriminant is a `msg_type` that switches to one of the two types of the message. The `xid` of a `REPLY` message always matches that of the initiating `CALL` message.

Note: The `xid` field is used only for clients that match reply messages with call messages; the service side cannot treat this ID as any type of sequence number.

Consider the following structure:

```
struct rpc_msg {
unsigned    xid;
union switch (enum msg_type) {
    CALL:  struct call_body;
    REPLY: struct reply_body;
} };
```

The following structure shows the body of an RPC request call. In version 2 of the RPC protocol specification, `rpcvers` must be equal to 2. The `prog`, `vers`, and `proc` fields specify the remote program, its version number, and the procedure to be called from within the remote program, respectively. These fields are followed by two authentication parameters: `cred` (authentication credentials) and `verf` (authentication verifier). The two authentication parameters are followed by the parameters to the remote procedure, which are specified by the specific program protocol.

```
struct call_body {
unsigned rpcvers;    /* Must be equal to two (2) */
unsigned prog;
unsigned vers;
unsigned proc;
struct opaque_auth cred;
struct opaque_auth verf;
/* Procedure-specific parameters start here */
};
```

The following structure shows the body of a reply to an RPC request. The call message was either accepted or rejected.

```
struct reply_body {
union switch (enum reply_stat) {
    MSG_ACCEPTED:  struct accepted_reply;
    MSG_DENIED:    struct rejected_reply;
}; };
```

The following structure shows a reply to an RPC request that the server accepted. (An error might exist, however, even though the request was accepted.) The first field is an authentication verifier that the server generates to validate itself to the caller. It is followed by a union whose discriminant is an enumeration of `accept_stat`. The `SUCCESS` arm of the union is protocol-specific. The `PROG_UNAVAIL`, `PROC_UNAVAIL`, and `GARBAGE_ARGS` arms of the union are void. The `PROG_MISMATCH` arm specifies the lowest and highest version numbers of the remote program that the server supports.

```
struct accepted_reply {
struct opaque_auth   verf;
union switch (enum accept_stat) {
    SUCCESS: struct {
        /*
         * Procedure-specific results start here
         */
```

```
};
PROG_MISMATCH: struct {
    unsigned low;
    unsigned high;
};
default: struct {
    /*
     * void.  Cases include PROG_UNAVAIL,
     * PROC_UNAVAIL, and GARBAGE_ARGS.
     */
}; }; };
```

The following structure shows a reply to an RPC request that the server rejected. A request can be rejected either because the server is not running a compatible version of the RPC protocol (RPC_MISMATCH) or the server refuses to authenticate the caller (AUTH_ERROR). In the case of an RPC version mismatch, the server returns the lowest and highest supported RPC version numbers. In the case of refused authentication, failure status is returned.

```
struct rejected_reply {
union switch (enum reject_stat) {
    RPC_MISMATCH: struct {
        unsigned low;
        unsigned high;
    };
    AUTH_ERROR: enum auth_stat;
}; };
```