Implementing Remote Procedure Calls

- an introduction to the fundamentals of RPCs, made during the advent of the technology.
- what is an RPC?
- what different resources are used?
- how does it work?
Background on the Paper

- RPCs first conceived/developed in 1976
- This approach outlines one of the earlier implementations of RPC designed to:
  - give programmers easy access to features
  - optimize performance
  - minimize load on high traffic servers
RPC Fundamentals

- Design is similar incorporates concepts from event based and thread based programming.
- We have procedures on a user machine, and procedures on a server machine.
- In an RPC, data is passed across a network, where it is executed in a procedure on the server machine.
- After processing, the data is passed back across the network to the user machine.
RPC Skeleton

User → User Stub → RPCRuntime
RPCRuntime ← Server Stub ← Server

- For RPC’s each process has a stub generated for it to process I/O
- Stubs import or export interfaces which allow an “exclusive” connection to take place between stubs.
Major Factors in Implementation

- Dealing with Communication Failures
- Shared address space?
- Fitting into programming systems, how to allow for flexibility?
- Transfer protocols; how to make sure/verify that data goes through?
- Security of transfer?
Goals of the Authors' Implementation

- Make an easy to use facility for programmers to utilize RPC
- Efficiency
- Powerful semantics, to set absolute rules for how RPC is processed.
Authors' Implementation

- In Cedar system, written in Mesa
- Uses procedures because of support
  could pass messages or perform remote fork with minimal hassle instead!
- No shared address space
  Not efficient; not built into languages
- Make semantics similar to local procedure calls
Fig. 1. The components of the system, and their interactions for a simple call.
Features provided by Cedar:

- stub, RPCRuntime

Procedure interface: import = user, export = server

- The stubs are automatically generated taking burden off of programmer

- Responsible for packing unpacking
Binding

- Each process can only be connected to one other process
- Grapevine DB used to facilitate process
- Entries have 2 fields:
  - type = function
  - instance = particular server
- Entries for type are a list of instances with exported interfaces in DB
- Table of export data kept in RpcRuntime
Binding (2)
Dealing with a Crash

- No guard for deadlock
- Polling informs user process of crash
Packet Transfer

- unlike large transfers; RPC's are supposed to be lightweight
- High cost for setup are NOT acceptable
- Single packet interaction:
Packets (2)
Exceptions

- Some are handled on server side
- Others get passed back to the user procedure and are handled there via exception packet
- Then catch is executed on the user side, and a subsequent package is sent back to the server
Processes

- In Mesa, it seems that processes are the equivalent of threads
- A pool of idle threads are created in the server
- When a process completes, processes are recycled
- If a process is waiting on another packet, this state is stored in the Ethernet interrupt handler
- When the packet is received, it is matched up with the appropriate process.