# Implementing RPC by Birrell & Nelson

Basic model, design decisions made, and implementation details are discussed

# Environment

- Part of Cedar Project
  - Dorado platform
  - Programming lang.: mesa, smalltalk, InterLisp, no asm
- Basic Idea:
  - Extend well-known procedure call mechanism to provide transfer of control and data over a communication network
  - Procedure call: based on call/return semantic
  - Remote proc call: call/return semantic across network

# Advantages and Goals

- Advantages:
  - Simple semantics: well-known and well-understood mechanism
  - Efficiency: simple for comm to be fast
  - General: comm b/w parts of algorithm is through procedure calls
- Goals:
  - Make distr  computation easy
  - Make RPC comm efficient
  - Make semantics of RPC as powerful as possible
  - Provide a secure comm

# Design Decisions

- Precise semantics of a call in presence of machine and comm failures, No timeout
- Address-containing arguments
- Binding: how a caller identifies loc and ID of callee
- Provide transparent integration: semantics of RPC = those of PC, compiler/runtime support: stubs and RPCRuntime

# Structures

- Stubs
  - Automatically generated by a program Lupine
  - Based on an interface: a list of proc names, types of arguments and results for each
  - Pointer args/results: no support
- import & export of an interface

# Binding

- Name of Interface: type and instance
  - Type: which interface to implement, mail server
  - Instance: which particular implementor,
- Locating a service: hard-code address, broadcast, a database (dictionary service)
- Use: Grapevine dist db
  - Distr over  multiple servers
  - ≥ 3 copies of each db entry: reliable, available via replicas
- Server: at sturt-up, exports its interface (to Grapevine) by telling its dispatcher
- Client: before calls, imports server: RPC runtime to Grapevine, connection setup via server

# Binding

- Exporting: interface name, and dispatcher proc
- On exporting machine: export table, record for each export (interface name, dispatch proc, (machine relative) unique id

# Binding Discussion

- Updates to Grapevine db: restricted via access controls, set of users able to export particular interface names
- Choices for binding time:
  - İmporter tells only type of interface, instance:dynamically chosen
  - İmporter tells type and instance, most common
  - İmporter tells net @ of an instance (binding occurs at compile time)

# Transport protocols

- Transport protocal specific for RPC, improved performance, 10x
- Minimize latency: request-reply unlikely to do large data transfer, no conn setup
- Call semantics:# of times procedure is executed
  - Exactly once: hard to achieve due to server and net failures
  - At most once: not executed at all or executed once
  - At least once: at least once but perhaps more, client keeps trying

# Simple Calls

- When all of args fit in a single packet,
  - Call packet: <call id, proc id, args>
  - Result packet: <call id, results>
- Ack: piggybacked on result packets:
- Min 2 packets per call
- Call identifier
  - To match result packet with correct call packet
  - Eliminate duplicate call packets: callee maintains a table of sequence numbers

# Complicated Calls

- Use multiple packets to send args and results
- For duplicate elimination, call relative sequence number
- Explicit ack is required for each but last to handle lost packests, long duration calls, long gaps b/w calls
- Probe packets: caller periodically sends to callee for long duration calls

# Conclusion

- Security:
  - Use Grapevine as an authentication service
  - Use DES (data enc std) to provide end-to-end encription of calls and results
  - Protection against eavesdropping, detection of modification, replay and fake calls
- Performance and Drawbacks
  - Remote calls measurements b/w two machines are reported:
  - Restriction on arg sizes and  types
  - Binding errors