Chapter 5.3: Remote Procedure Calls

CS/ECPE 5516: Comm. Network
Prof. Abrams
Spring 2000

Problem

- Suppose you want to make a procedure call
- Caller and called code are on different Internet hosts
- Situation is “remote procedure call”

Sample Scenario…

- You run Internet banking app on your PC.
- App makes RPCs to bank’s computer
- You want to move $100 from your checking to savings account:

  \[
  \text{moveC2S}(100) \quad \text{via RPC}
  \]

Try this in groups…

- Is TCP a good protocol?
- Is UDP a good protocol?
- How does regular old procedure call get converted to network communication?
- Design your own protocol. Be sure it…
  - Is efficient
  - Completes call with high probability
- Include in your design…
  - List of tasks protocol must do
  - Header format

Did you consider these issues:

- How do you find Internet host running the remote procedure?
- Any way that remote procedure could be called twice (e.g., if network duplicates a packet containing the call)?
- What if remote procedure host executes procedure but crashes before sending reply?
- How to make caller wait until receiver is done executing procedure

Let’s answer the questions…
?: Is TCP a good RPC protocol?
Not really…
- RPC needs request/reply
- TCP is for streaming
- RPC needs messages
- TCP is for streaming
- RPC needs speed: maybe 2 messages
- TCP uses 3+2+4 messages

?: Is UDP a good RPC protocol?
Not really…
- UDP doesn’t care if message is lost

?: How does proc call get converted to network comm?
- A stub or proxy does conversion.
- It uses proc call on one side, network communication on the other.
- See Fig. 5.11 in Peterson/Davie
- Use “rcp compiler” to invisibly add proxies
- Proxy marshalls & unmarshalls procedure arguments.

?: How do you find host running remote procedure?
- Method 1: Caller names host to be called
- Method 2: Directory service:
  “I want a square-root function. What host on the net has one?”

Example: Sun’s JINI
- Allows simplifies network device interconnection with a Lookup/Discovery service
- Each device provides services to others
- Devices register when they join community; publishes service
  I’m a 600dpi B&W printer; my service is printing
- An app can lookup a service to discover a device

?: Any way that remote procedure could be called twice
- Yes: request is duplicated
- Want semantics to clarify requests:
  - At-most-once semantics:
    - At most one copy of request is delivered
    - “At most” because zero might be delivered
  - Alternate: idempotent procedures:
    - can be executed multiple times, but give same result
    - example: square-root
What if RPC executes but host crashes before sending reply?
- Need transaction semantics at host.
- Commit protocol on server guarantees no intermediate state visible
  Example:
  - Subtract $100 from checking
  - Crash
  - Oops! You lost $100 – if intermediate state visible
- Implemented by list on disk of actions to execute on reboot
- Java Enterprise Beans offers transactions

Outline of Peterson/Davie Solution (pp. 400-426)
- Three sub-protocols:
  - BLAST: sends message from host A to B
  - CHAN: implements request/reply over BLAST
  - SELECT: dispatches request to correct process

BLAST – Send Message
Sender:
- Fragment large message into packets
- Transmits packets
- Set timer
- If no response by timeout, give up
- If ack arrives, resend listed packets

BLAST (2)
Receiver
- Set timer
- Receive fragments:
  - place in order in buffer
  - reset timer
- When last fragment arrives, send ack
- If timer pops, send ack
- Ack lists missing packets
- After 3 timeouts, give up

CHAN – Implement request/reply
- Guarantees message delivery (retransmits)
- Eliminates dups (via sequence number)
- Implements at-most-once semantics
- Allows synchronization between sender/receiver
- Uses reply as implicit ack of request
- Sends I’m alive messages in case procedure takes long time to execute

SELECT
- CHAN runs between one app and one proc
- SELECT handles set of calls between apps to common remote procedure via serialization
- SELECT also provides hierarchical names for destinations, vs. ports:
  - (program #, proc #)
  - (file-server-program, read)
  - (file-server-program, write)