Fault-Tolerant Middleware: Communication

Reliable communication middelware for distributed systems

Kyle E. Benson

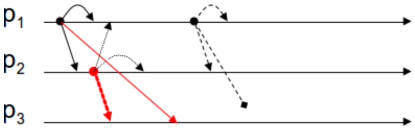
Department of Computer Science
University of California, Irvine
Irvine, California 92697

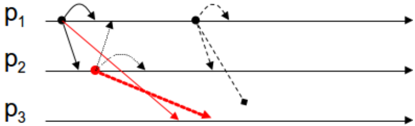
kebenson@uci.edu

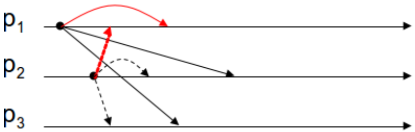
May 30, 2013

Introduction

- Fault-tolerant communication goals
  - **Correctness** of messages, non-corruption guarantee
  - **Ordering** of messages
    - FIFO: If M_a sent before M_b, M_a received before M_b
    - Causal: If M_a causes M_b to be sent, M_a received before M_b at all processes
    - Total: If M_a delivered before M_b at process P_j, M_a delivered before M_b at all other P_i too
  - **Delivery** guarantees, bounds on latency





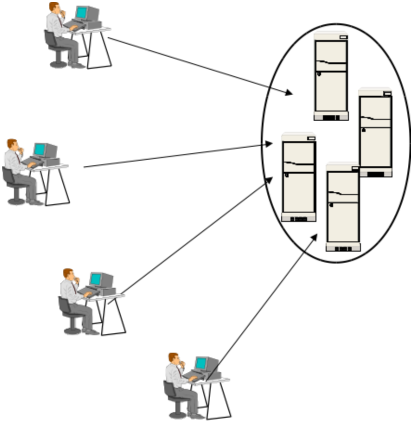


Foundations of Reliability

- How to make unicast reliable?
  - Prevent omission failures
    - Guarantee message delivery
    - Assume correct processes will deliver messages
    - Redeliver on *timeout*
    - No bound on time before reply
  - Guarantee ordering; ignore repeated messages
    - Sequence numbers
    - Timestamps
    - Logical clocks
  - Message integrity
    - Hashing
    - Certificates
    - Keys

Reliable Group Communication

- Why not just use TCP?
  - Consider 100 machines each running 10 processes
  - 1000+ TCP connections at each machine
  - 1 million+ total!
  - Not scalable!
  - Relies on timeouts
  - Ordering harder
  - Similar problems with other client-server connection-oriented protocols
- How to exploit redundancy in communication paths?
- Answer: multicast trees

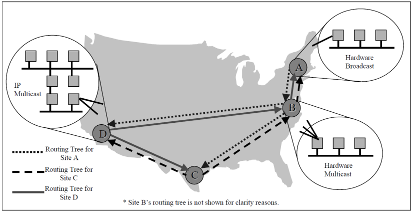


Reliable Distributed Multicast

- **Observation:** distributed systems naturally address groups of processes
  - Coordinating events
  - Replica communication
  - Anycast
  - Reduction
  - Parallel computation
- Distributed process groups $\rightarrow$ multicast groups
  - IP multicast not always supported
  - Make application layer multicast
  - Let the *middleware* handle delivering message to proper groups
  - Decouples machine address from distributed function target
  - More efficient network usage

Spread Toolkit

- Open-source tools for group communication
- `www.spread.org`
- Hierarchical
  - Wide area: hop protocol
  - Local area: ring protocol
- Scales: tens of sites, tens of machines in each
- Bindings in many languages / platforms
  - C/C++
  - Java
  - Python, Perl, Ruby
  - Windows (98 - XP)
  - BSD / Linux / Solaris / Irix
  - Mac OS X

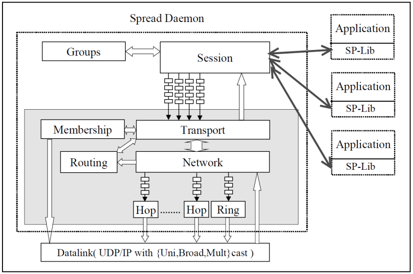


Spread's Daemon-client Model

- Daemons provide messaging services
- User applications contact closest daemon for group communication
- Minimizes expensive membership changes
- Can tune number of daemons
- Wide area dissemination
  - Each site has one representative daemon for wide area dissemination
  - Routing trees rooted at each site
  - Each site = node in tree
  - Supports pruning, fast-retransmit, non-blocking send
- Daemons could even be run on routers
  - More for better performance
  - Fewer for less costly recovery

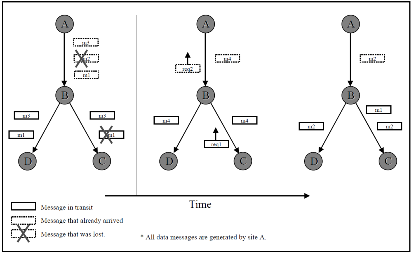
Spread Architecture

- Several queues between session and transport
- Can support priorities
- Network module info $\rightarrow$ Routing module $\rightarrow$ Routing trees
- Can have multiple hops, only one ring
- Extended virtual synchrony
  - Handle network partitions
  - Handle re-merges
  - Joins
  - Leaves



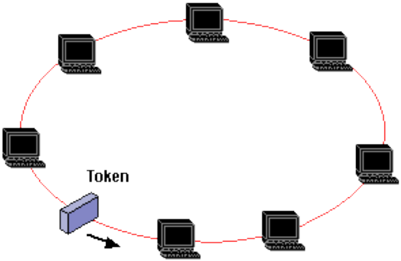
Spread's Hop Protocol

- Uses UDP/IP
- Handle losses hop-by-hop, not end-to-end
- Forward immediately, ignoring order
- NACKs for retransmit
  - NACK all outstanding packets
  - Wait timeout before requesting retransmit
  - Declares sender dead if retries > threshold
  - Latency bound
- cumulative ACKs
- Sliding window
- Token/leaky bucket for flow control



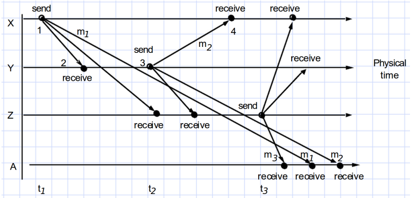
Spread's Ring Protocol

- Multiple daemons in one site
- Local ordering, reliable dissemination
- Receives a token
  - Retransmits requested by previous holder
  - Receive messages
  - Send packets
  - Update and forward token



Spread's Message Ordering

- Based on Lamport Time Stamp (logical clock)
- Sequence numbers
- Agreed
  - FIFO and Causal
  - Consistent across groups
- Safe
  - Consistent with agreed
  - Each site generates All-Received-Upto values
  - Disseminate across sites
  - Global All-Received-Upto values



References

- Andreas Larsson. *Lab 2 Group Communication*. `http://www.cse.chalmers.se/edu/year/2011/course/TDA297/slides/presentation_lab2.ppt`
- ccsejc. *Distributed Systems*. `http://www.scribd.com/doc/23723915/28/Reliable-Communication`
- The Spread Toolkit. `http://www.spread.org/SpreadPlatforms.html`
- Prashant Shenoy. *CS 677 Remote Procedure Calls*. `http://lass.cs.umass.edu/~shenoy/courses/677/lectures.html`
- Yair Amir and Jonathan Stanton. *The Spread Wide Area Group Communication System*