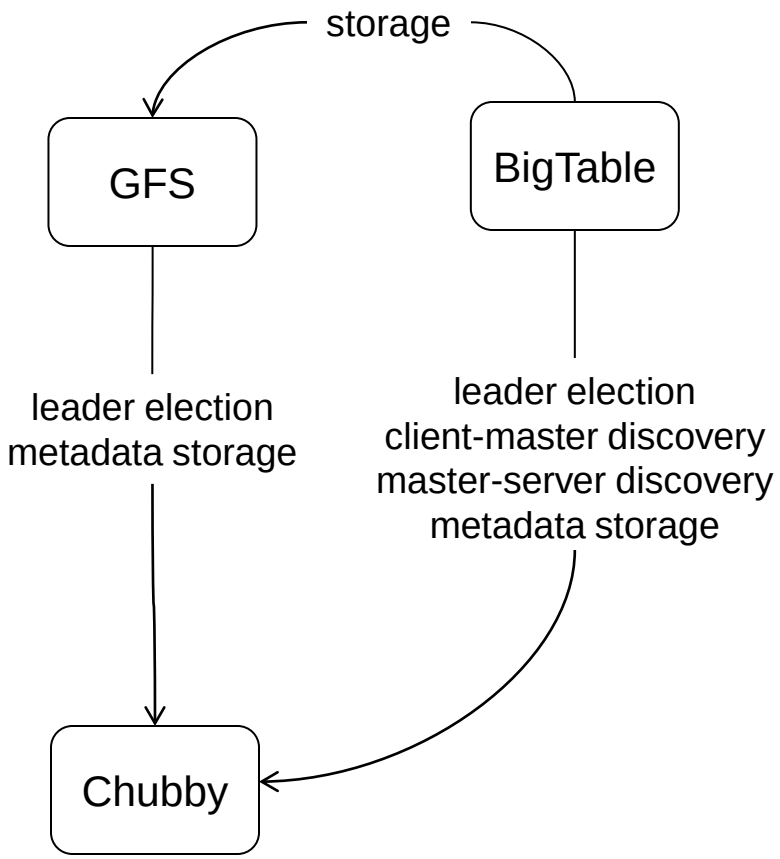
Chubby

Lock server for distributed applications

Overview

- Purpose
  - Client synchronization
    - course-grain locking
    - leader election
  - Environment information
    - naming
    - metadata
- Goals
  - High reliability/availability
  - Thousands of clients

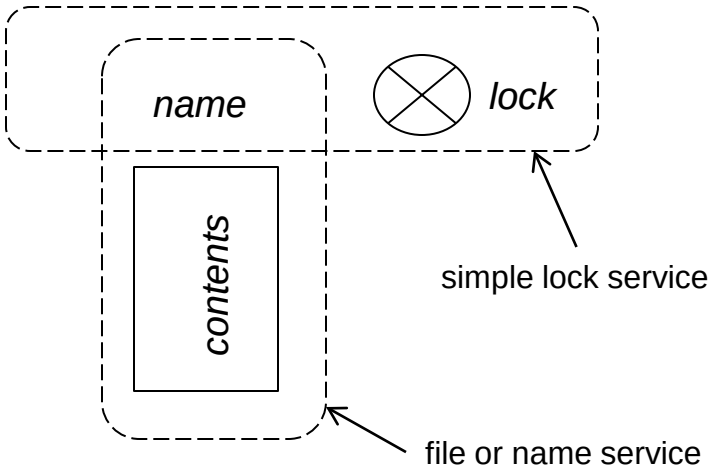


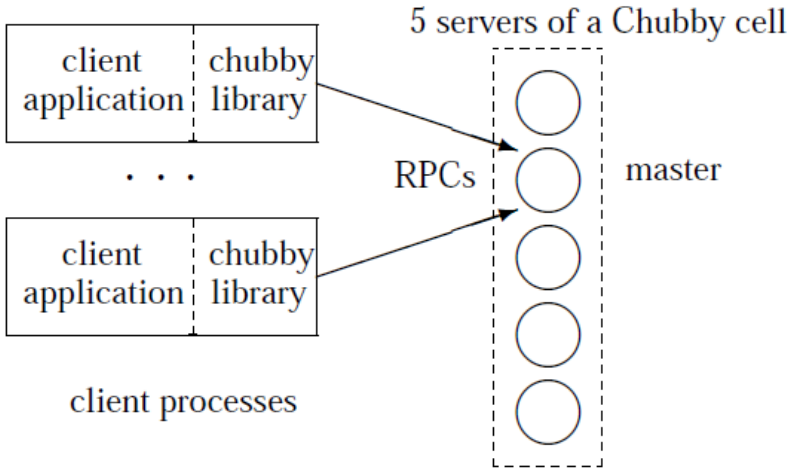
Design

- Service not a library (for consensus)
  - Easier to retrofit into evolving applications
  - Need environment information, not just consensus
  - More familiar interface (at least to their users)
  - Eases burden of maintaining a client quorum for consensus
- Services
  - Course grain locks (that survive failures)
    - Low acquisition rate
    - Held for considerable time
  - Small files
    - Whole file read/write
    - Limited to 256Kb
  - Consistent client-side caching
  - Event notification (of changes)
  - Access control

Structure

- Library provides API for client applications
- Chubby cell
  - replicas (identical, maintain copies of database)
  - election of a master via consensus protocol
  - master lease:
    - period of time for which master is elected
    - can be renewed

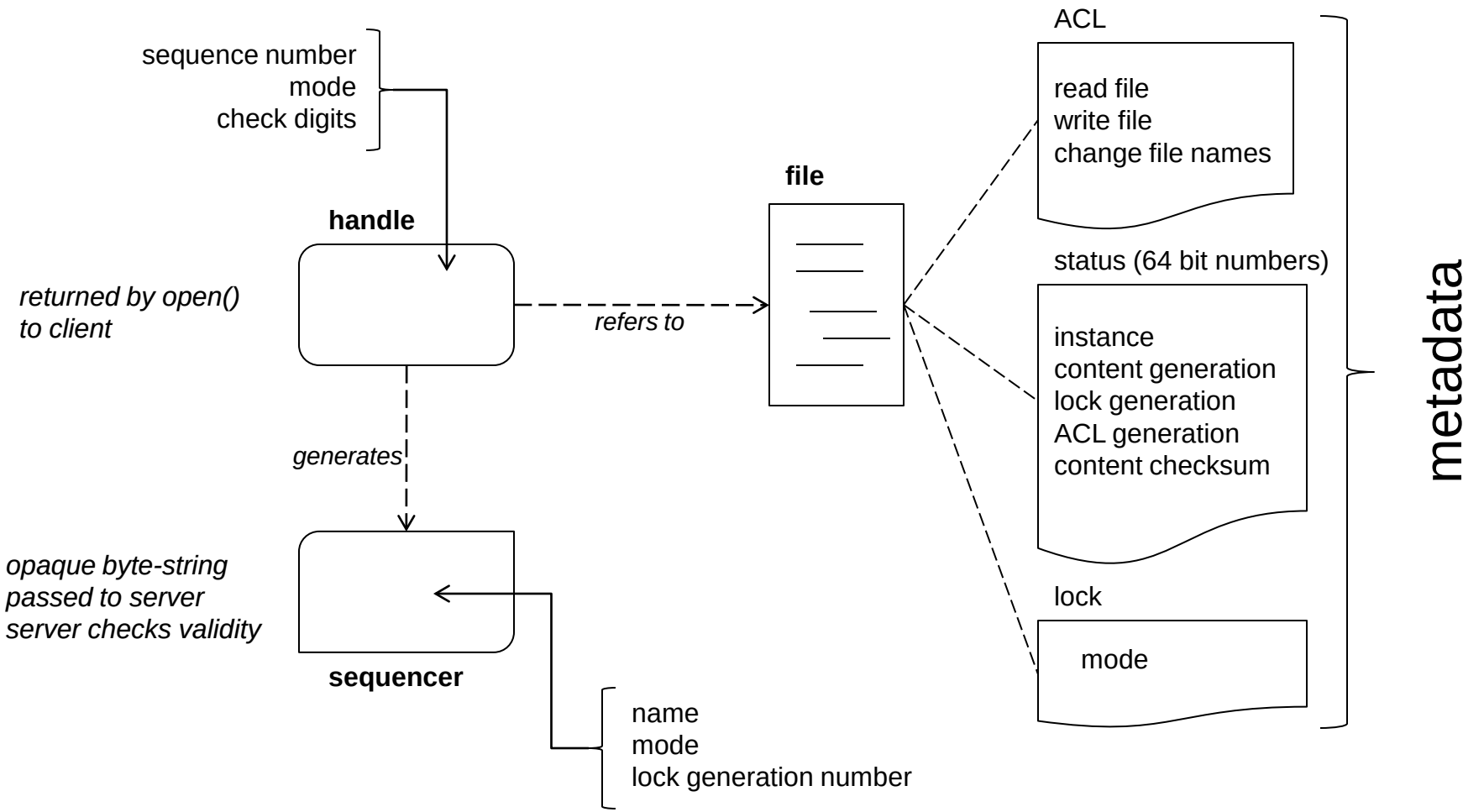




Files

- Naming
  - Tree of directories/files
  - Example: /ls/foo/wombat/pouch
    - /ls: lock service
    - /foo: name of chubby cell
    - /wombat/pouch: file on named cell
- Sub-trees may be served from different Chubby masters
  - Files cannot move between directories
  - No directory modified times
  - No path-dependent access control
  - No symbolic or hard links
  - Does not reveal last-access times

File, Handles, Sequencers



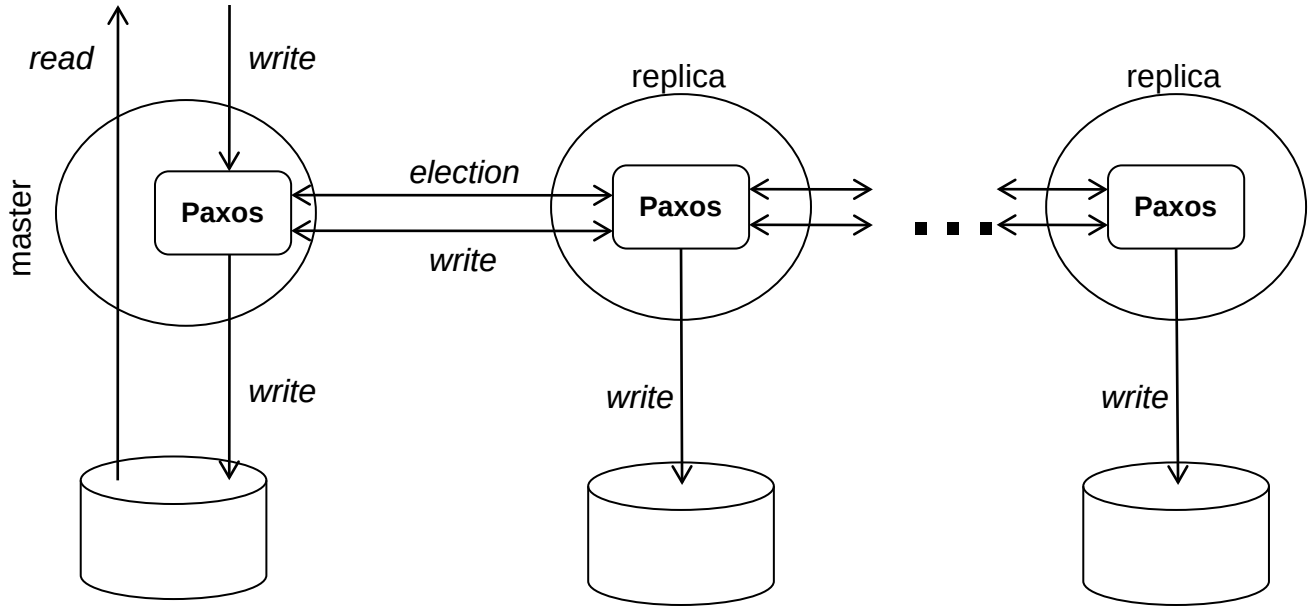
Locks and Ordering

- Locks
  - modes: read (shared), write (exclusive)
  - permissions: requires write permission to file
  - are advisory: the lock associated with a file does NOT control access to the file's contents
- Ordering
  - problem
    - after acquiring a lock and issuing a request (R1) a client may fail
    - another client may acquire the lock and issue its own request (R2)
    - R1 arrive later at the server and be acted upon (possible inconsistency)
  - solution
    - client requests sequencer for lock (sequence contains lock generation number)
    - client passes sequence to server along with request
    - server verifies that sequencer is still valid
    - server can maintain session with Chubby and cache sequencer informaiton

Locks and election

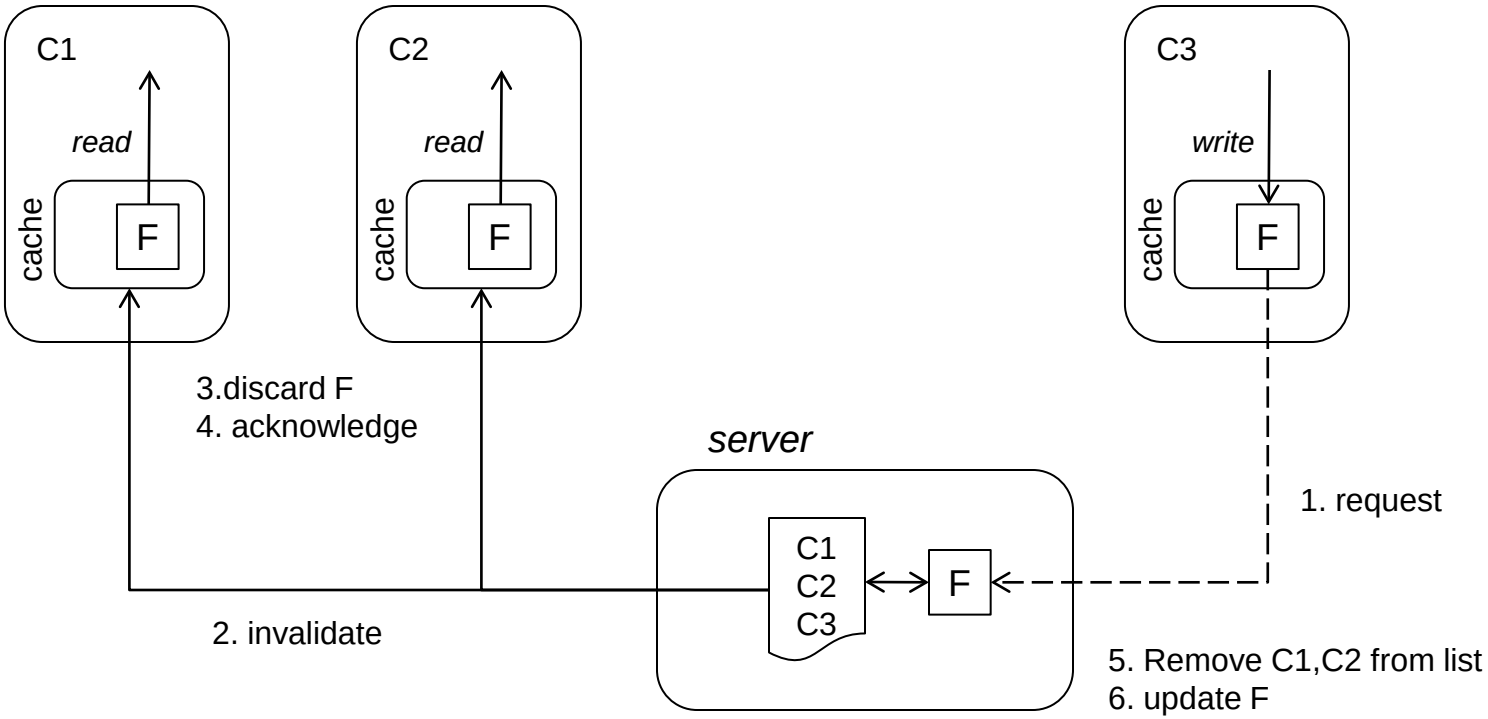
- clients in a distributed applications using Chubby may elect a primary using the Chubby interface
- steps in primary election
  - clients open a lock file and attempt to acquire the lock in exclusive mode
  - one client succeeds (becomes the primary) and writes its name/identity into the file
  - other clients fail (become replicas) and discover the name of the primary by reading the file
  - primary obtains sequencer and passes it to servers (servers can insure that the elected primary is still valid)

Operations



- reads:
  - served by master alone
  - creates potential performance bottleneck (→ client side caching)
- writes:
  - acknowledged after consensus by a majority of replicas
  - insures all (non-failed) replicas have identical databases (and can become master in case the current master fails)

Caching - Invalidation

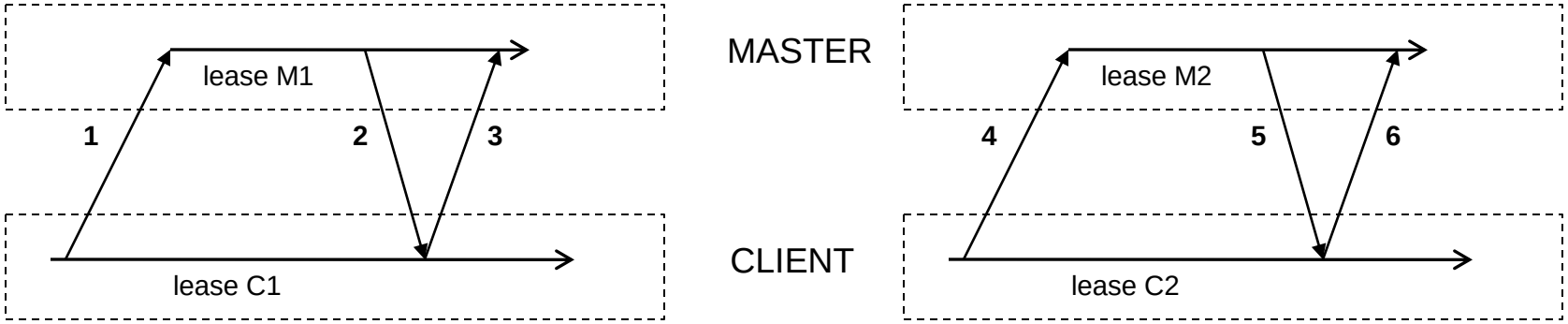


- Clients maintain cached copy for reading; server maintains list of clients with a copy
- On write attempt, server invalidates cached copies
- When invalidated, clients discard cached copy and acknowledge
- When acknowledgments received, server update cache list and data (F)

Leases

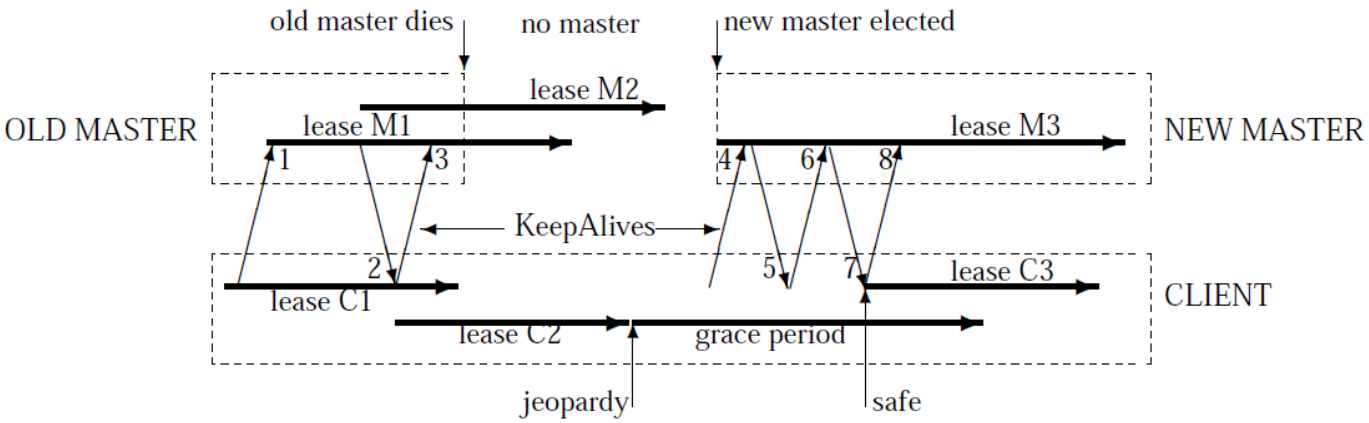
- Lease
  - a promise by one party to abide by an agreement for a given interval of time unless the promise is explicitly revoked
  - may be renewed or extended by either party
  - at expiration of interval, agreement no longer valid
- Uses in Chubby
  - master lease (role oriented) – interval during which one replica holds the right to serves as the cell master
  - session lease (resource oriented) – interval during which client's cached items (handles, locks, files) are valid
- Leases contribute to reliability/fault tolerance
  - at expiration of lease held by a failed party, the non-failed party knows the state of the resource associated with the lease

Session Leases and KeepAlive Messages



- (1) lease requested by client via RPC; master accepts lease request by not returning
- (2) master returns to indicate near-term expiration of lease
- (3) client immediately re-issues lease request to extend lease interval
- (4) lease requested by client via RPC; master accepts lease request by not returning
- (5) master returns piggybacking cache invalidation notification; client discards invalidated cached objects;
- (6) client re-issues lease request to extend lease interval

Master lease management



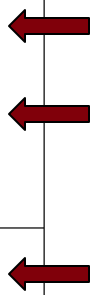
- client maintains local lease: a conservative estimate of master's lease interval
- if client's local lease expires,
  - client disables cache (session is in *jeopardy*)
  - sends "*jeopardy*" event to application
  - application can suspend activity
- client attempts to exchange KeepAlive messages with master during grace period
  - succeed : re-enables cache; send s"safe" event to application
  - fail: discards cache; sends "expired" event to application
- master fail-over shown in figure

Scaling

- reducing communication with the master
  - create an arbitrary number of Chubby cells to divide the load of numerous separate applications
  - increase the lease time to process fewer KeepAlive messages
  - client caching of file data, metadata, absence of files, and open handles to reduce contact with server
  - use protocol-conversion servers
- use proxies
  - handles KeepAlive and read requests
  - introduces another point of failure
- partition name space

Usage

time since last fail-over	18 days
fail-over duration	14s
active clients (direct)	22k
additional proxied clients	32k
files open	12k
naming-related	60%
client-is-caching-file entries	230k
distinct files cached	24k
names negatively cached	32k
exclusive locks	1k
shared locks	0
stored directories	8k
ephemeral	0.1%
stored files	22k
0-1k bytes	90%
1k-10k bytes	10%
> 10k bytes	0.2%
naming-related	46%
mirrored ACLs & config info	27%
GFS and Bigtable meta-data	11%
ephemeral	3%
RPC rate	1-2k/s
KeepAlive	93%
GetStat	2%
Open	1%
CreateSession	1%
GetContentsAndStat	0.4%
SetContents	680ppm
Acquire	31ppm



- most files are small
- popular as a name server
- RPC traffic dominated by KeepAlive