A new name server architecture

John Schimmel, Larry McVoy & Andrew Chang

jes,lm,awc@engr.sgi.com

Silicon Graphics Engineering

What are the problems we are trying to solve?

General name service cleanup

Client side caching

Client side impervious to temporary name server failures

Support multiple nameserver protocols

Want less, or no, configuration

Want less and easier administration

Secure name service

What problems are we not trying to solve?

No new name server protocols

No new name server configuration

No new anything that is unnecessary

Use old technology where possible

Outline

Client side problems & solutions

Server side problems & solutions

Performance problems & solutions

Administration problems & solutions

Security problems & solutions

Open Issues

Release schedules, etc.

Client side name server issues

Client side interfaces to DNS, YP, etc. in libc

- Crufty, name server specific code
- Hard to fix bugs, change policy, and/or add new services

Client side caching is add hoc

- One entry cache is typical in libc
- Libc caches flushed on execv()
- DNS may/may not have a cache
- No negative caching anywhere

Client side changes

All name service specific code removed from libc

Add per client, system wide, service independent cache

Add "cache miss handler" (aka resolver)

Client side libc interfaces

No changes visible to applications

- gethostbyname() stays the same

New generic lookup interface

- nslookup()

Interfaces in libc greatly simplified

```
extern char *
nslookup(char *domain, char *map, char *key);

char *
getaliasbyname(char *alias)
{

    return (nslookup(0, "aliases", alias));
}
```

Client side cache

The cache is a multi reader/writer “database”

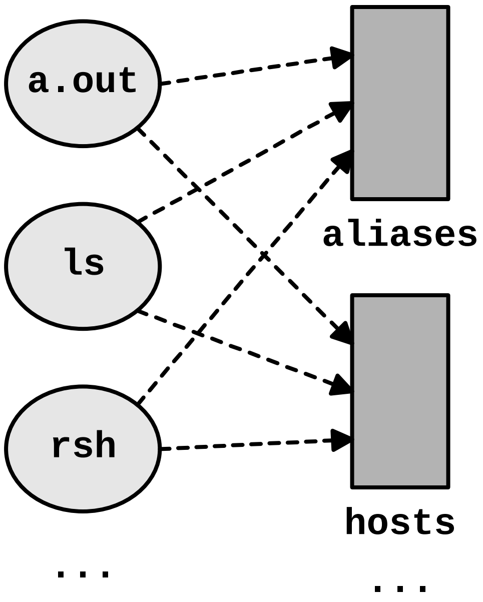
- Uses mmap (smaller & faster)
- Uses dbm compatible interface (with extensions)
- 64 bit, network byte order data structures
- Libc lookup queries the cache

Client side cache picture

Only one copy of the data in memory

- The entire cache for all maps & processes can be 4K

All processes share all data



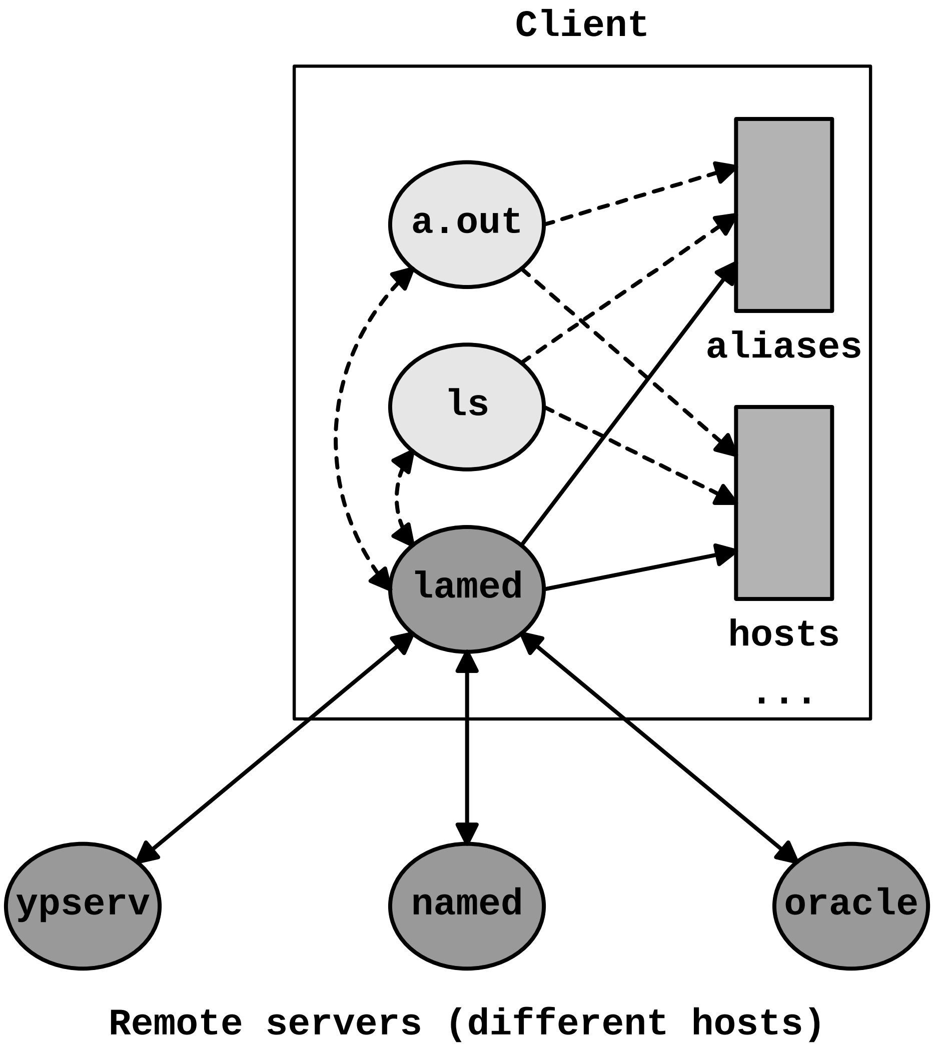
Client side resolver: lamed process

nslookup() calls lamed to resolve cache misses

lamed

- has generic “object” interface for querying name services
  - each name service is a shared library
  - reload when nsswitch.conf changes
- implements name service ordering (/etc/nsswitch.conf)
- manages the cache
  - timeouts, flushes, negative caching

Client side lamed picture



Client side summary

Remove name service specific code from libc

Add a fast system wide cache

Add a client side, name server independent resolver

Move name service specific code into shared libs

Add support for /etc/nsswitch.conf

Server side name server issues

N services imply N server processes

- i.e., ypserv, named, etc.

Some servers call other servers directly

- yp calls DNS

Server caching is ad hoc

Server setup is ad hoc

Server side super server

Teach server to answer other server requests

Support common name services

- ypserv, dns, others

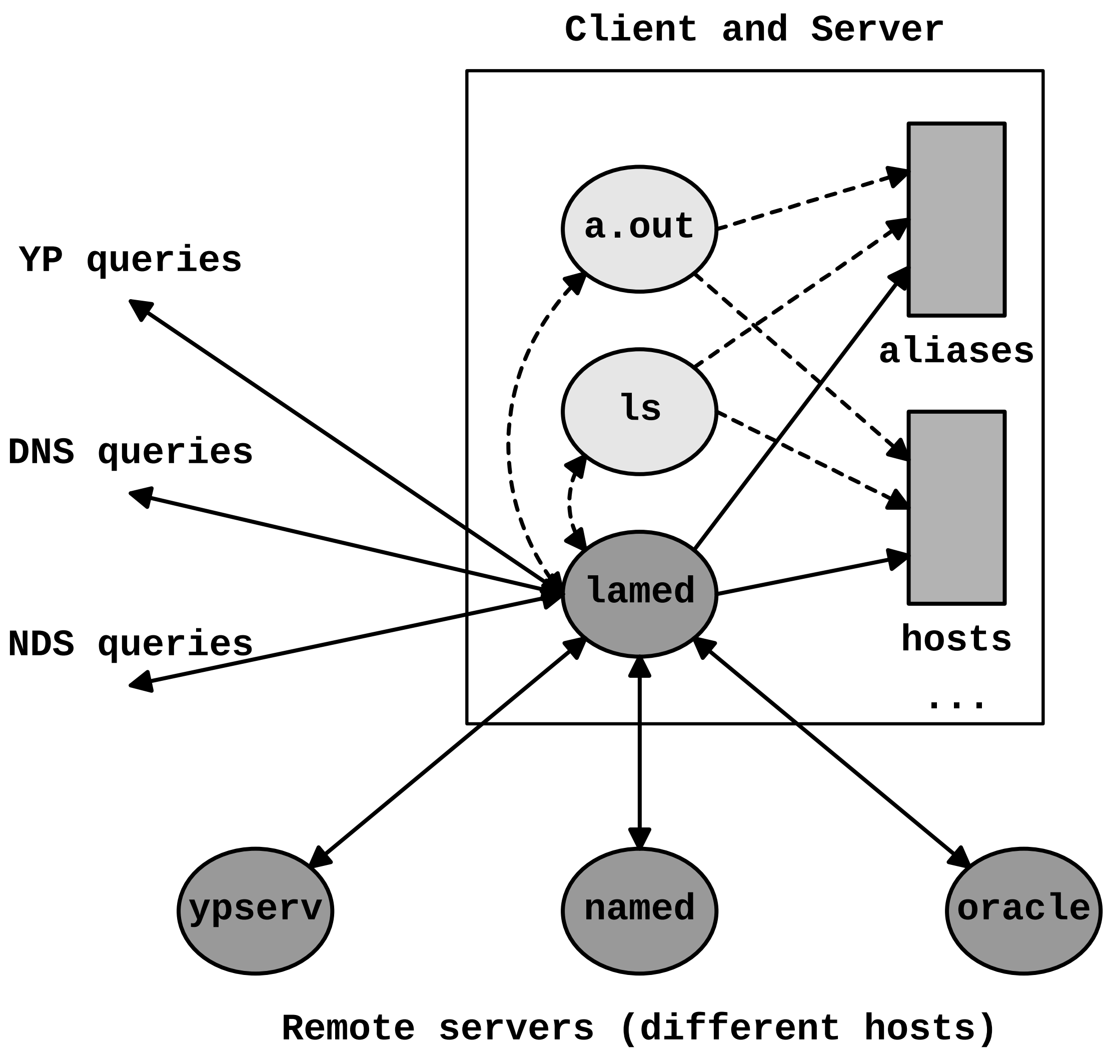
Server acts as a translator when combined with client side

- we get this “for free”

Lamed server acts as a cache when used as a translator

- we get this “for free” too

Single name server picture



Absorbing old protocols

So far, server is a translator and a cache

Can absorb old implementations

- ypserv has been re-implemented in lamed

Server to server protocol is DNS/Hesiod

- Possible, and likely, that DNS & lamed will merge

Performance problems & solutions

RPC performance

Database (cache) performance

Scaling problems

RPC performance

Blocking RPC’s kill performance

All lamed to name server queries are non blocking

- Decompose RPC into send, recv, and demux
- demux replies using RPC xid
- One process is fast enough to have 1000’s of outstanding RPCs.

For protocols w/o XIDs, use helper processes

- Send RPC style messages to helper
- Local files may be done this way

Database performance

Load 2 million entry passwd map

- 5 minutes, CPU bound
- Cold lookup
  - log2(database size / bits per page) + 1 disk reads
  - One lookup is ˜24 milliseconds
    + 4K pagesize, 128MB database
- Hot lookup
  - About 6 usecs on a 200Mhz R4K (no TLB misses)

Database memory usage

- hash is about 70% efficient
- 2 million entry passwd table was 215 MB

Scaling problems

Most name servers perform poorly with large databases

Initial server release will support (in a single domain)

- 5,000,000 users
- 7,500,000 hosts
- All associated data (groups, etc.)

Requires 64 bit file system offsets & holey files

Long term goals are 100x initial goals

It’s all a database problem and mdbm scales

Administration problems and solutions

Zero configuration clients

Zero configuration slave servers

Administrative “shell”

Server administration

Pseudo sub domains

Well known master locations

Zero configuration clients

Multicast upon first boot looking for name server

Cache results

Use the closest server

Reverify servers periodically

Default behavior is to cache from the remote server

- Implies client has no /etc/nsswitch.conf, uses server's
- Can override by providing /etc/nsswitch.conf

Zero configuration slave servers

A lamed client becomes a slave server by adding "-s"

- No other configuration necessary
- Well, until you add security into the soup

That client will answer future server location multi casts

Administrative “shell”

ns command [options]

- **ns build creates the maps from flat files**
- **ns build -yp creates maps from yp maps**
- **ns chpass changes password**
- **etc.**

Server administration

Default data source is RCS versioned flat files

- live in /var/ns/etc..engr.sgi.com
- ns build looks in /var/ns/etc..* for multi domain service
  - /var/ns/etc..engr.sgi.com
  - /var/ns/etc..corp.sgi.com

DNS databases automatically generated from flat files

Pseudo sub domains

Used to allow change to a portion of a domain

- I.e., each building in engr.sgi.com is a sub domain

Implemented as sub directories in /var/ns/etc..engr.sgi.com

Each map is constructed from the concatenation of all files

- i.e., hosts B1/hosts B2/hosts B3/hosts ...

Permission checking is standard file system permissions

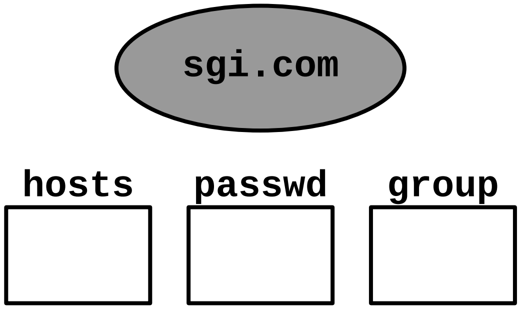
Could be made more sophisticated if needed

- Restrict IP address allocation, uid allocation, etc.

Small company example

Small company, one administrator, single flat domain space

`/var/ns/etc..sgi.com`

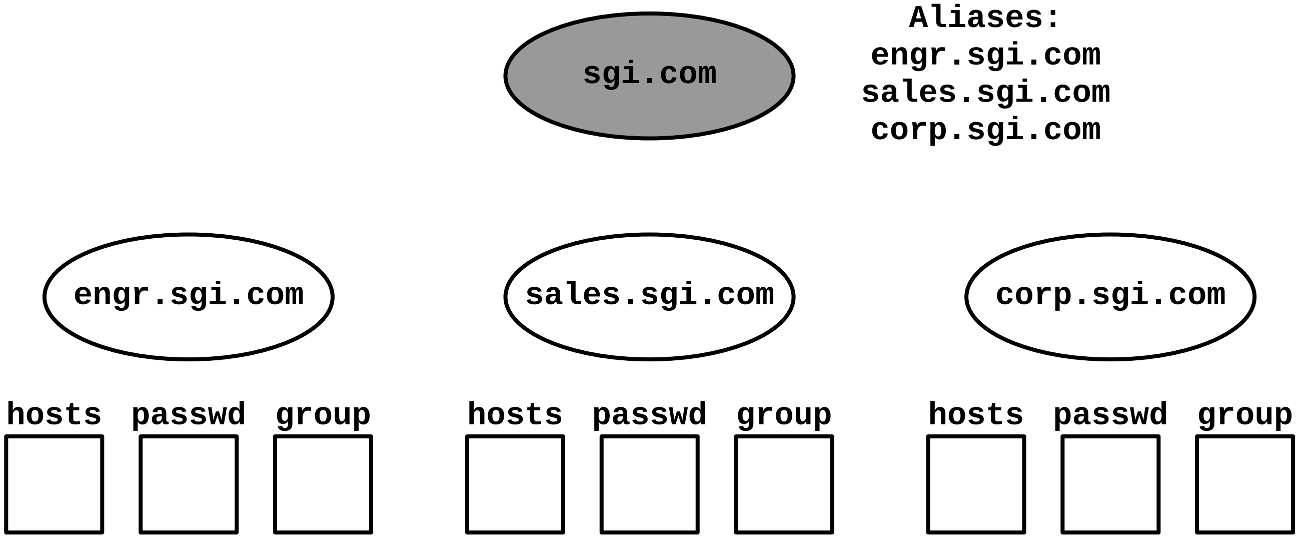


Growing company example

Still single domain

- multiple admins, 1 per pseudo domain
- names (users) unique across all pseudo domains

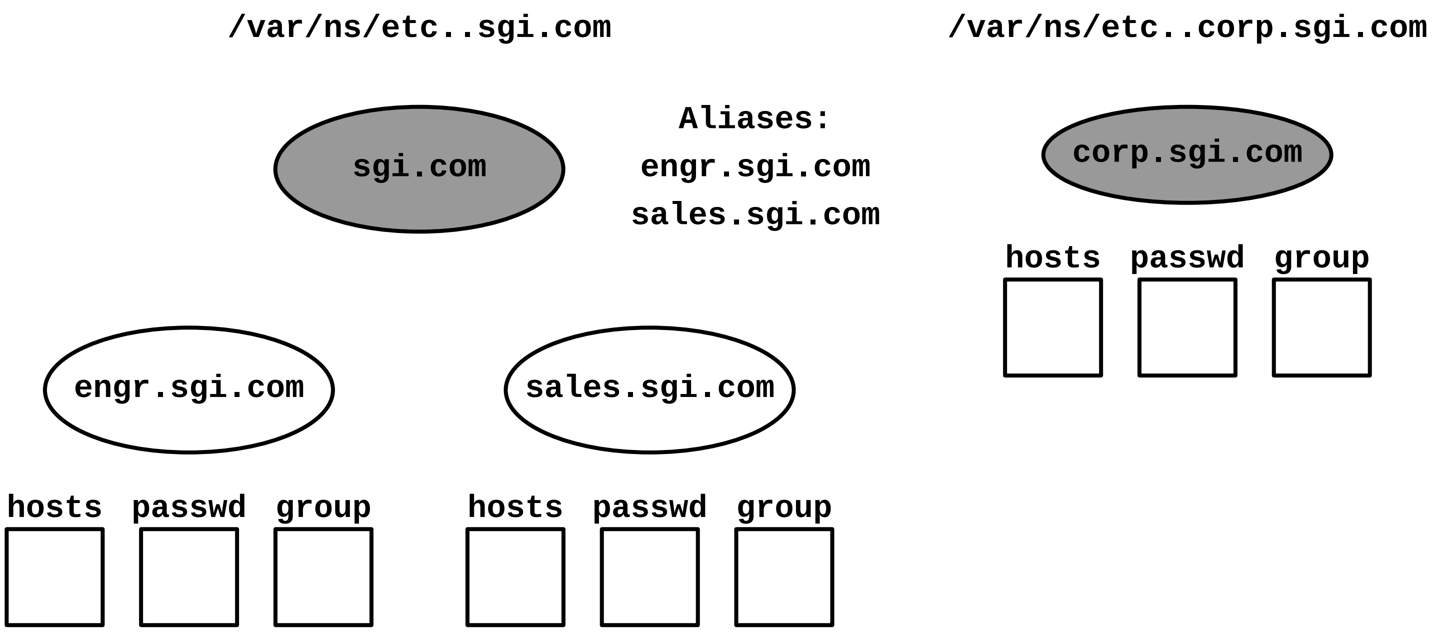
`/var/ns/etc..sgi.com`



Promoting a pseudo domain to a real domain

Just move the pseudo domain directory up one level

- sgi.com no longer aliased to corp.sgi.com
- names unique with engr & sales, but disjoint from corp



Well known master locations

ns.domain.name

ns.engr.sgi.com

ns.sgi.com

ns.com

ns.

Remote security

Current model is yp style security (i.e., none)

Moving towards RSA model

- Not until release 2.0
- Probably not until US gov deregulates encryption

Goals

- provide correct (digitally signed) data
- allow remote updates (chpass)

Open Issues

Boot strapping

Export regulations

DNS & lamed merge o matic

Schedules and standardization

Code will come standard in future IRIX versions

- Client side due out spring ’96
- Server side in staged releases
  - No security or DNS merge in server release 1.0

Reference port will be distributed for free in Linux

Code available from SGI under ONC style license

Several RFCs forthcoming

- Client side, server side, location mechanisms, etc.

A new name server architecture

John Schimmel, Larry McVoy & Andrew Chang

jes,lm,awc@engr.sgi.com

Silicon Graphics Engineering