Sharing Files with NFS

ComS 252 — Iowa State University

Barry Britt and Andrew Miner

Motivation

This lecture discusses the UNIX equivalent of Samba

- I.e., setting up UNIX–style file sharing and network identity

- Covers NFS: Network File System
- Covers NIS: Network Information System

Motivation

This lecture discusses the UNIX equivalent of Samba

- I.e., setting up UNIX–style file sharing and network identity

- Covers NFS: Network File System
- Covers NIS: Network Information System

- But first: some other things
  - The Portmapper
  - `xinetd`
  - TCP Wrappers

Motivation

This lecture discusses the UNIX equivalent of Samba

- I.e., setting up UNIX–style file sharing and network identity

- Covers NFS: Network File System
- Covers NIS: Network Information System

- But first: some other things
  - The Portmapper
  - `xinetd`
  - TCP Wrappers
- Why those random things?
  - I have simplified things for you again
  - Now we have to discuss reality

RPC — What is it?

- RPC: Remote Procedure Call
  - Idea: call a procedure on another machine
  - Similar to Java Remote Method Invocation
  - Abstracts away communication details for programmers
    - Real work is done by user–space libraries and/or kernel
  - Can be implemented in different ways
- Underneath RPC: usually, the client/server model
  - With some protocol for client/server communication

RPC — What is it?

- RPC: Remote Procedure Call
  - Idea: call a procedure on another machine
  - Similar to Java Remote Method Invocation
  - Abstracts away communication details for programmers
    - Real work is done by user–space libraries and/or kernel
  - Can be implemented in different ways
- Underneath RPC: usually, the client/server model
  - With some protocol for client/server communication
- Typical high–level RPC flow
  1. Client program calls a library function
  2. Library constructs messages according to RPC protocol
  3. OS sends messages to server machine
  4. Server library builds function call from messages
  5. Server procedure is called
  6. Reply is similar

RPC Implementation

- The main one in UNIX: Sun Microsystems' RPC
- Is now "Open Network Computing RPC"
- Uses an open protocol — current version is RFC 5531
- Cilent/server data is
  - Transmitted in XDR ...
  - Via UDP or TCP

RPC Implementation

- The main one in UNIX: Sun Microsystems' RPC
- Is now "Open Network Computing RPC"
- Uses an open protocol — current version is RFC 5531
- Cilent/server data is
  - Transmitted in XDR ...
  - Via UDP or TCP

XDR: eXternal Data Representation

- Specifies format of transferred data
- Allows data transfer between machines with different CPUs
- This is "presentation layer" stuff

Fun details about ONC RPC

- Procedures on the server are handled by various programs
  - Are identified by a unique program number
  - File `/etc/rpc` lists the mapping from program to number
  - Each program can handle one or more procedures
- RPC programs have version numbers
  - Allows support for different protocol versions, simultaneously
- RPC programs may support TCP and UDP, simultaneously
  - E.g., use UDP unless data does not fit into a single datagram
- Each RPC program will listen on a fixed, reserved port
  - That way the client can find it

Fun details about ONC RPC

- Procedures on the server are handled by various programs
  - Are identified by a unique program number
  - File `/etc/rpc` lists the mapping from program to number
  - Each program can handle one or more procedures
- RPC programs have version numbers
  - Allows support for different protocol versions, simultaneously
- RPC programs may support TCP and UDP, simultaneously
  - E.g., use UDP unless data does not fit into a single datagram
- ~~Each RPC program will listen on a fixed, reserved port~~ NO!
  - And the client can find it, how?

Why port numbers are not fixed

Too inflexible for the server. Consider:

- Only 1024 reserved ports
  - We cannot have reserved ports for every service
  - Especially if we want to have *custom* services
- Only one program can bind to a given port
- Cannot use different programs for one fixed port number
  - E.g., for different protocol versions
  - E.g., for TCP vs. UDP
- Transparency is a good thing
  - Allows us to change port numbers on the server . . .
  - Without reconfiguring the client(s)

How this works

- Each port number can have at most one program bound to it
- Each program can bind to more than one port number
- And, a program is not tied to the reserved port number
  - E.g., we can use any port for `ssh`
- But how does the RPC client find the RPC server program?

How this works

- Each port number can have at most one program bound to it
- Each program can bind to more than one port number
- And, a program is not tied to the reserved port number
  - E.g., we can use any port for `ssh`
- But how does the RPC client find the RPC server program?
  - There's a service for that ...
  - Called the portmapper
    - `portmap`, `rpc.portmap`, or `rpcbind`
  - The portmapper uses a fixed, reserved port
    - Port number 111

The portmapper daemon

- Makes RPC work
- Knows port numbers used by the RPC programs
  - RPC program registers its port number with the portmapper
- Programs may or may not use reserved ports

Typical sequence of events

1. Server machine starts server daemon
2. Server daemon binds to a port number
3. Server daemon registers itself with portmapper
4. Client machine starts client process
5. Client process connects to portmapper on server machine
6. Gets the port number
7. Client process uses that port number to connect to server

Using a firewall

How do we allow a portmapped service through the firewall?

Using a firewall

How do we allow a portmapped service through the firewall?

- Easy way:
  1. Set the service to use a fixed port number
  2. Configure the firewall to trust packets for that port number

Using a firewall

How do we allow a portmapped service through the firewall?

- Easy way:
  1. Set the service to use a fixed port number
  2. Configure the firewall to trust packets for that port number
- Hard way:
  1. Let the service use whatever port number it wants
  2. Update the firewall rules at run time based on the port number

Using a firewall

How do we allow a portmapped service through the firewall?

- Easy way:
  1. Set the service to use a fixed port number
  2. Configure the firewall to trust packets for that port number
- Hard way:
  1. Let the service use whatever port number it wants
  2. Update the firewall rules at run time based on the port number

How do we make a service use a fixed port number?

Configuring the portmapper daemon

1. Configure portmapper to start or not at boot time
   - Using `systemctl`, as usual
   - Might start even though not enabled in `systemctl`

Configuring the portmapper daemon

1. Configure portmapper to start or not at boot time
   - Using `systemctl`, as usual
   - Might start even though not enabled in `systemctl`
2. That's all. No other configuration required.

Configuring the portmapper daemon

1. Configure portmapper to start or not at boot time
   - Using `systemctl`, as usual
   - Might start even though not enabled in `systemctl`
2. That's all. No other configuration required.

But can't I configure the port number for service `xyz`?

Configuring the portmapper daemon

1. Configure portmapper to start or not at boot time
   - Using `systemctl`, as usual
   - Might start even though not enabled in `systemctl`
2. That's all. No other configuration required.

But can't I configure the port number for service `xyz`?

- Yes, you can do that
- But it is not done by configuring the portmapper
- Instead you configure service `xyz`
  - Depends on the service
  - Could be in `xyz`'s configuration file
  - Could be command–line options to `xyz`
    - And maybe these can be set in some other configuration file

Debugging port numbers

`rpcinfo`

- Your new best friend
- Lists programs available to handle RPCs

`-p` Show port numbers used by those programs

- Can run this locally and remotely
  - Give the IP or FQDN for the remote machine

Example: `rpcinfo`

```
client$
```

Example: rpcinfo

```
client$ rpcinfo -p server
```

Example: rpcinfo

```
client$ rpcinfo -p server
rpcinfo: can't contact portmapper
client$
```

Example: rpcinfo

```
client$ rpcinfo -p server
rpcinfo: can't contact portmapper
client$ ping -c 1 server
```

Example: rpcinfo

```
client$ rpcinfo -p server
rpcinfo: can't contact portmapper
client$ ping -c 1 server
PING server (192.168.0.1) 56(84) bytes of data.
64 bytes from 192.168.0.1: icmp_req=1 ttl=64 time=1.61ms

--- 192.168.0.1 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss
rtt min/avg/max/mdev = 1.616/1.616/1.616/0.000 ms
client$
```

Example: rpcinfo

```
client$ rpcinfo -p server
rpcinfo: can't contact portmapper
client$ ping -c 1 server
PING server (192.168.0.1) 56(84) bytes of data.
64 bytes from 192.168.0.1: icmp_req=1 ttl=64 time=1.61ms

--- 192.168.0.1 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss
rtt min/avg/max/mdev = 1.616/1.616/1.616/0.000 ms
client$ ssh server
```

Example: rpcinfo

```
client$ rpcinfo -p server
rpcinfo: can't contact portmapper
client$ ping -c 1 server
PING server (192.168.0.1) 56(84) bytes of data.
64 bytes from 192.168.0.1: icmp_req=1 ttl=64 time=1.61ms

--- 192.168.0.1 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss
rtt min/avg/max/mdev = 1.616/1.616/1.616/0.000 ms
client$ ssh server
Password:
```

Example: rpcinfo

```
client$ rpcinfo -p server
rpcinfo: can't contact portmapper
client$ ping -c 1 server
PING server (192.168.0.1) 56(84) bytes of data.
64 bytes from 192.168.0.1: icmp_req=1 ttl=64 time=1.61ms

--- 192.168.0.1 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss
rtt min/avg/max/mdev = 1.616/1.616/1.616/0.000 ms
client$ ssh server
Password:
server$
```

Example: rpcinfo

```
client$ rpcinfo -p server
rpcinfo: can't contact portmapper
client$ ping -c 1 server
PING server (192.168.0.1) 56(84) bytes of data.
64 bytes from 192.168.0.1: icmp_req=1 ttl=64 time=1.61ms

--- 192.168.0.1 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss
rtt min/avg/max/mdev = 1.616/1.616/1.616/0.000 ms
client$ ssh server
Password:
server$ rpcinfo -p
```

Example: rpcinfo

```
1 packets transmitted, 1 received, 0% packet loss
rtt min/avg/max/mdev = 1.616/1.616/1.616/0.000 ms
client$ ssh server
Password:
server$ rpcinfo -p
   program vers proto   port  service
    100000    4   tcp    111  portmapper
    100000    3   tcp    111  portmapper
    100000    2   tcp    111  portmapper
    100000    4   udp    111  portmapper
    100000    3   udp    111  portmapper
    100000    2   udp    111  portmapper
    100024    1   udp  47429  status
    100024    1   tcp  42343  status
server$
```

`xinetd` — What is it?

- eXtended InterNET services Daemon
  - Replaces older `inetd`
- Is a super service
  - One daemon listening for connections to lots of services
  - Some, but not all, services go through `xinetd`
    - Useful for low–traffic services
    - Should not need to touch this for homework

xinetd — What is it?

- eXtended InterNET services Daemon
  - Replaces older `inetd`
- Is a super service
  - One daemon listening for connections to lots of services
  - Some, but not all, services go through `xinetd`
    - Useful for low–traffic services
    - Should not need to touch this for homework
- Typical workflow
  1. Incoming connection for ftp
  2. `xinetd` is bound to ftp port
  3. `xinetd` checks rules for starting ftp service
     - More about this in a minute
  4. If rules are not satisfied, then drop the connection
  5. `xinetd` starts the ftp service
  6. `xinetd` passes control of the connection to ftp
  7. The ftp service handles the connection from then on

Benefits of `xinetd`

- Services run only when needed
  - Lower memory usage
  - Fewer "doors" into the system
- Central point for administration
- Access control for services
  - Based on client IP address or FQDN
- Logging
- Containment of services against Denial of Service attacks
  - Limit number of servers to run at once
  - Limit total number of services to run
  - Limit size of log files
- Can bind services to specific interfaces
- Can bind services to private addresses
- Can proxy
  - We have not discussed proxies yet...

Configuring xinetd defaults (apply to every service)

Example /etc/xinetd.conf

```
defaults
{
    instances           = 60
    log_type            = SYSLOG   authpriv
    log_on_success      = HOST PID
    log_on_failure      = HOST
    cps                 = 25 30
}
includedir /etc/xinetd.d
```

- Maximum of 60 simultaneous requests
- Log entries are written to /var/log/secure
- On success, host's IP address, and PID of service, are logged
- On failure, host's IP address is logged
- If $>$ 25 connections/sec. for a service: retire service for 30 sec.

/etc/xinetd.d

- Directory containing service–specific configuration files
- Same format as default configuration file
- File names should correlate to the service
- For example. . .

/etc/xinetd.d/ftp

```
service ftp {
    disable          = no
    socket_type      = stream             # stream=TCP; dgram=UDP
    user             = root               # Who the daemon runs as
    only_from        = <IP addrs>         # Authorized clients
    server           = /usr/sbin/in.ftpd # Server daemon path
    server_args      = <args>             # Arguments to ftp
    port             = 12345              # Port number, of course
    max_load         = 2                  # Max 2 servers at a time
    log_on_failure   += USERID            # Log userid on failure
}
```

What are TCP Wrappers?

- Access control wrappers
  - Before connection is established, check access rules...
  - Logs information on success or failure
    - Usually to `/var/log/secure` or `/var/log/messages`
  - On success: that's it for the wrapper
  - On failure: can do some interesting things...

What are TCP Wrappers?

- Access control wrappers
  - Before connection is established, check access rules...
  - Logs information on success or failure
    - Usually to `/var/log/secure` or `/var/log/messages`
  - On success: that's it for the wrapper
  - On failure: can do some interesting things...

- Are provided by a library (typically, `/usr/lib/libwrap.a`)
- A service is "TCP–wrapped" if it uses the library
  - Pro trick: you can list the libraries used by an executable with

```
ldd executable
```

What are TCP Wrappers?

- Access control wrappers
  - Before connection is established, check access rules...
  - Logs information on success or failure
    - Usually to `/var/log/secure` or `/var/log/messages`
  - On success: that's it for the wrapper
  - On failure: can do some interesting things...

- Are provided by a library (typically, `/usr/lib/libwrap.a`)
- A service is "TCP–wrapped" if it uses the library
  - Pro trick: you can list the libraries used by an executable with

```
ldd executable
```

- `xinetd` is (usually) TCP–wrapped
  - All services within `xinetd` are wrapped

Access rules for TCP Wrappers

- Are specified in two files (the "hosts access files"):
  - `/etc/hosts.allow`
  - `/etc/hosts.deny`
- Rules are specified in the same syntax for both files
  - We will discuss this in a minute
- A wrapped service checks rules in the following order
  1. If there is a matching rule in `/etc/hosts.allow`, then the connection is allowed
  2. Otherwise, if there is a matching rule in `/etc/hosts.deny`, then the connection is denied
  3. Otherwise, the connection is allowed
- Changes to these files take effect immediately
  - No need to restart any network services

Format of hosts access files

- Lines starting with "#" are ignored
- One rule per line
  - Be sure the last rule has a "newline" character
  - Use a backslash to split a line
- Rules are applied in order
- Rule syntax:

```
<daemons> : <clients> [: <option> : <option> ...]
```

`<daemons>` : Comma–separated list of process names
`<clients>` : Comma–separated list of hosts
  - Can use FQDNs, IP addresses, and patterns...

`<option>` : (Optional) action when the rule is triggered

- See `man hosts_access` for more information

Fun with access rules

Patterns for clients

`ALL` — match everything

- Can also be used for the daemons

`LOCAL` — any host that does not contain a dot

- E.g., "`localhost`"

`KNOWN` — host name and address, or user, are known

`UNKNOWN` — host name or address, or user are unknown

`PARANOID` — host name does not match host address

Note: `KNOWN`, `UNKNOWN`, and `PARANOID` require DNS

Fun with access rules (2)

Specifying clients: subnets

- Hostname starting with a dot
  - Will match all FQDNs ending with that name
  - E.g., "`.iastate.edu`" will match
    - `www.iastate.edu`
    - `www.cs.iastate.edu`
    - `...`
- IP address ending with a dot
  - Will match that subnet
  - E.g., "`192.168.`" will match `192.168.x.x` subnet
- IP address and netmask pair
  - E.g., "`192.168.0.0/255.255.0.0`"
- IP address and prefix length . . . does not work

"Proper" setup of hosts access files

- Get your services working first, then lock down the machine
  - That goes for the firewall, also
- IP addresses are preferred over FQDNs
  - FQDNs rely on DNS
  - DNS servers can be unavailable or compromised (cracked)
- Use the following `/etc/hosts.deny` file:

```
ALL : ALL
```

"Proper" setup of hosts access files

- Get your services working first, then lock down the machine
  - That goes for the firewall, also
- IP addresses are preferred over FQDNs
  - FQDNs rely on DNS
  - DNS servers can be unavailable or compromised (cracked)
- Use the following `/etc/hosts.deny` file:

```
ALL : ALL
```

Anything not specified in `/etc/hosts.allow` is denied

Example

/etc/hosts.allow

```
# Completely trust local machine
    ALL : localhost
# Allow ssh access from anywhere except 44.22.11.x
    sshd : 44.22.11.  : deny
    sshd : ALL
# Allow portmapper access from our local subnet
    portmap : 192.168.42.
```

/etc/hosts.deny

```
# Always be paranoid
    ALL : ALL
```

NFS — What is it?

- Network File System
- File sharing protocol developed by Sun Microsystems
- Allows servers to "export" (share) a folder
- Allows clients to mount an exported folder
- Supports UNIX–style filesystems
  - Permissions can be exported
  - File owners and groups can be exported (as ID numbers)
- Different versions of the protocol
  - Version 1: for experimental purposes
  - Version 2: RFC 1094, March 1989
    - Specifies UDP
  - Version 3: RFC 1813, June 1995
    - Allows TCP
  - Version 4: RFC 3530, April 2003
    - Some significant changes

NFS client

- NFS versions 2 and 3 use the portmapper
  - For file locking
- Make sure the portmapper is running on the client
- Server exports can be mounted using

```
mount -t nfs server:/export /mount/point/as/usual
```

  - The `-t nfs` is optional
  - Mount knows that `server:/export` means NFS
- Server exports may also be automounted
- See `man mount.nfs` and `man nfs` for mounting options
  - Will want the defaults for most options

NFS server

- Need the portmapper running for versions 2 and 3
- Need the server daemon running
  - Use "`systemctl`" to start service "`nfs`"
  - This may start several daemons
- Server configuration: in `/etc/exports`
  - Lines starting with "#" are ignored
  - One rule per line
  - Rule format: `folder IPs(options) IPs(options) ...`
    - IP can be a complete address
    - IP can be a subnet of the form IP/mask or IP/prefix length
- See `man exports` for more information
  - Especially for the allowed options

NFS server

- Need the portmapper running for versions 2 and 3
- Need the server daemon running
  - Use "`systemctl`" to start service "`nfs`"
  - This may start several daemons
- Server configuration: in `/etc/exports`
  - Lines starting with "#" are ignored
  - One rule per line
  - Rule format: `folder IPs(options) IPs(options) ...`
    - IP can be a complete address
    - IP can be a subnet of the form IP/mask or IP/prefix length
- See `man exports` for more information
  - Especially for the allowed options

Example `/etc/exports`

```
/home     192.168.42.0/24(rw,sync)
/special  192.168.42.3(ro) 192.168.42.4(ro)
```

NIS — What is it?

- Network Information Service
- Originally developed by Sun Microsystems
- Keeps information "synchronized" among hosts
  - Conceptually: configuration files are synchronized
  - In practice: file data is transferred over the network
- Was originally called Yellow Pages or YP
  - This name is trademarked
  - Sun had to change the name
  - But this explains utility names like `ypcat`
- Is old, and not very secure
  - Was supposed to be replaced by NIS+ in 1992
    - Better security, flexibility, scalability
    - But "cumbersome" to set up and administer
    - So many users stuck with NIS
  - Today, LDAP is often the better choice

How the NIS server works

- Information to share is drawn from source files
  - Also known as master files
  - Can draw from lots of fun files, including:
    - `/etc/passwd` and `/etc/shadow`
    - `/etc/group` and `/etc/gshadow`
    - `/etc/rpc`
    - `/etc/hosts`
    - `/etc/auto.master` and `/etc/auto.*` map files
- Information is converted into map files
  - As `dbm` database files
  - Each map file is indexed by one of the columns
    - Can look up records quickly by index
  - May build more than one map file from each source file
    - E.g., passwords indexed by userID
    - E.g., passwords indexed by username

Setting up an NIS server

- The service is `ypserv`
  - Manage using `systemctl` as usual
- Set the NIS domain name
  - Analogous to the "Workgroup Name" in Windows
  - Can be different from DNS domain
  - In Fedora, done with an entry in `/etc/sysconfig/network`

```
NISDOMAIN=MyDomainName
```

  - Service `nis-domainname` will set NIS domain name
  - Utility `nisdomainname` will show the NIS domain name
- Edit server configuration
  - File is `/etc/ypserv.conf`
  - Default should be fine for homework

Setting up an NIS server (2)

- Edit the access control file `/var/yp/securenets`
  - Similar in idea to `/etc/hosts.allow`, but much simpler
  - Lines starting with "#" are ignored
  - Other lines are of the form: `subnet_mask IP_address`
    - Requests from the specified subnet are allowed
  - The local host must be included

Setting up an NIS server (2)

- Edit the access control file `/var/yp/securenets`
  - Similar in idea to `/etc/hosts.allow`, but much simpler
  - Lines starting with "#" are ignored
  - Other lines are of the form: `subnet_mask IP_address`
    - Requests from the specified subnet are allowed
  - The local host must be included

Example `/var/yp/securenets`

```
# The local machine (required)
255.255.255.255         127.0.0.1
# Accept from the 192.168.42.0/24 subnet
255.255.255.0           192.168.42.0
```

Setting up an NIS server (3)

- Can have several NIS servers running
  - One is the Primary (or "Master") server
  - Others are Secondary (or "Slave") servers
  - For homework — just set up one server
- Initialize the Primary server
  - Run `ypinit -m`
  - `-m` to initialize the maps
    - More about the maps, in a minute...
  - `ypinit` typically is NOT found in the PATH
    - Usually in `/usr/lib64/yp` or `/usr/lib/yp`
  - You will be asked to list the NIS server hosts
- Initialize each Secondary server
  - Run `ypinit -s primary`

Changing maps on the primary NIS server

- Edit `/var/yp/Makefile` to change which maps are built
  - Default normally includes the user and group files, i.e., `/etc/passwd`, `/etc/shadow`, `/etc/group`, `/etc/gpasswd`
  - Default may include other interesting files
  - Check the `all` target to add other maps
- Run `make` in `/var/yp` to rebuild the map files
  - Do this whenever a source file changes
    - E.g., if you add an entry to `/etc/hosts`

Changing maps on the primary NIS server

- Edit `/var/yp/Makefile` to change which maps are built
  - Default normally includes the user and group files, i.e., `/etc/passwd`, `/etc/shadow`, `/etc/group`, `/etc/gpasswd`
  - Default may include other interesting files
  - Check the `all` target to add other maps
- Run `make` in `/var/yp` to rebuild the map files
  - Do this whenever a source file changes
    - E.g., if you add an entry to `/etc/hosts`
- What about passwords?

Changing maps on the primary NIS server

- Edit `/var/yp/Makefile` to change which maps are built
  - Default normally includes the user and group files, i.e., `/etc/passwd`, `/etc/shadow`, `/etc/group`, `/etc/gpasswd`
  - Default may include other interesting files
  - Check the `all` target to add other maps
- Run `make` in `/var/yp` to rebuild the map files
  - Do this whenever a source file changes
    - E.g., if you add an entry to `/etc/hosts`
- What about passwords?
- Reasonable solutions:
  1. Rebuild the map files every few hours, and have users change passwords on the NIS server
  2. Run `yppasswdd`...
- For homework — do not worry about this

NIS password updates

`yppasswdd`

- The NIS password update daemon
- Runs only on the primary server
- Updates `/etc/shadow` and NIS map files

`yppasswd`

- Use instead of `passwd` on the clients
- Will send password updates to `yppasswdd`

System accounts

What about system accounts?

- Want NIS to work in a heterogeneous environment
- Meaning, different clients may have different system accounts
- Do not want system accounts "synchronized" over NIS
- Do not want `root` account "synchronized" over NIS

System accounts

What about system accounts?

- Want NIS to work in a heterogeneous environment
- Meaning, different clients may have different system accounts
- Do not want system accounts "synchronized" over NIS
- Do not want `root` account "synchronized" over NIS

Solution:

- Can specify the lowest userID to include in NIS maps
- Can specify the lowest groupID to include in NIS maps
- Use variables in `/var/yp/Makefile`:

  `MINUID` Red Hat default: 500
  `MINGID` Red Hat default: 500

How the NIS client works

- Entries in files to be sent from the server
- But the files are not modified
- On the client, need to specify
  - Should the system use local files
  - Should the system use NIS files
  - Should the system use both
    - If so, who gets precedence

Setting up an NIS client

- The service is ypbind
  - Manage using `systemctl` as usual
- Set the NIS domain name and server(s)
  - Entries in `/etc/yp.conf`

```
domain MyDomainName server MyServerNameOrIP
```

  - Can have multiple server entries for one domain
  - Can specify more than one domain
- Set the domain name in `/etc/sysconfig/network`
  - Not needed if only one domain in `/etc/yp.conf`

Allow NIS users on client

Old systems: edit configuration file(s)

- `/etc/nsswitch.conf` (Name Service Switch)
- Can mess up file and then *nobody* can log in!

New systems: use utility `authselect`

- Idea is to select an authentication "profile"
- To list available profiles:

```
authselect list
```

- To switch profiles (as root):

```
authselect select profile
```

Pro trick: `autofs` configuration over NIS

On the server

- Edit `Makefile` and run `make` in `/var/yp`
  - Specify `auto.master` as a source file for NIS
  - Specify `auto.foo` and other "maps" as source files for NIS

On the client

- Specify `files nis` or `nis files` for `automount`
  - Only for `auto.master`; what about maps like `auto.foo`?

Pro trick: `autofs` configuration over NIS

On the server

- Edit `Makefile` and run `make` in `/var/yp`
  - Specify `auto.master` as a source file for NIS
  - Specify `auto.foo` and other "maps" as source files for NIS

On the client

- Specify `files nis` or `nis files` for `automount`
  - Only for `auto.master`; what about maps like `auto.foo`?

`autofs` "maps" and NIS

- Use an entry in `auto.master` with type `yp`:

```
/foo     yp:auto.foo   --timeout 60
```

Utilities for debugging NIS on a client

ypcat

- Usage: `ypcat [options] mapname`
- Displays all keys in a NIS database
- E.g.: `ypcat passwd`

ypmatch

- Usage: `ypmatch [options] key mapname`
- Find and display a matching key in a NIS database
- E.g.: `ypmatch bob passwd`
- Same output as `ypcat passwd | grep bob`

`authselect` : configure authentication sources

`ldd` : print shared library dependencies

`nisdomainname` : show the NIS domain name

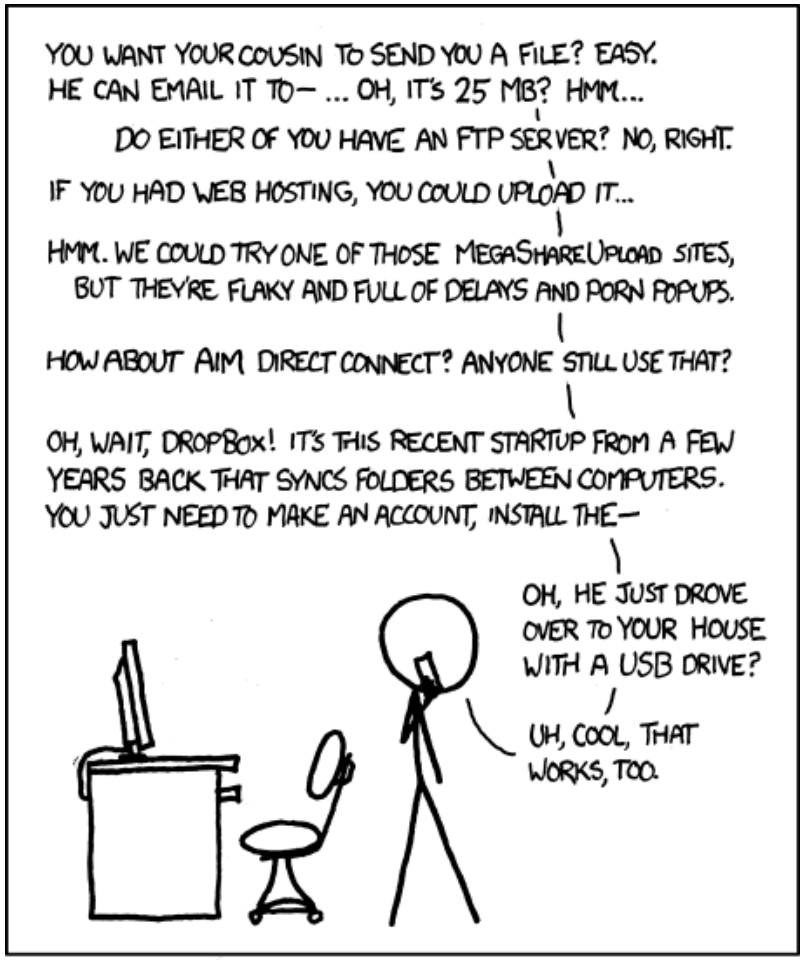
`rpcinfo` : show RPC programs (`-p` for port numbers)

`ypcat` : show an NIS database

`ypmatch` : show a matching key in an NIS database

`yppasswd` : change passwords on an NIS client

A somewhat appropriate comic: http://xkcd.com/949



I LIKE HOW WE'VE HAD THE INTERNET FOR DECADES, YET "SENDING FILES" IS SOMETHING EARLY ADOPTERS ARE STILL FIGURING OUT HOW TO DO.

End of lecture