F7 Network/Filesystem Services & Applications — NFS
Department of Electrical and Computer Engineering
Assistant Professor Gregory R. Kriehn

F7 Network File Systems (NFS)

Network File System, or NFS, allows partitions, file systems, or directories to be mounted from a remote server onto a client computer via a local area network. It is extremely useful when you want to, say, keep user's home directories in a centralized location and serve them out to clients. I use NFS to mount my home partition and mail directory on my laptop when the ethernet connection is established, and follow it up with `rsync` scripts to allow for back-and-forth file synchronization of the local laptop home directory with the server. I can also mask the laptop from the server otherwise (i.e., when I'm using a wireless connection) by specifying two domain names for the laptop: `[laptop host].[domain name]` and `[laptop host].localdomain`. The `[laptop host].[domain name]` host name is used for the ethernet connection, and `[laptop host].localdomain` is used for wireless and dsl connections. (This can be specified in `/etc/hosts` and via your network settings). This way, I can work remotely if need be, but keep my **entire** filesystem synchronized when using my ethernet connection. When I was a graduate student, my old research professor allowed his file system to repeatedly(!) fracture, meaning that he had to keep local copies of the server's home directories, his current laptop's home directory, and his old laptop's home directory on a single hard drive. :p

I also use NFS to backup my entire home directory structure on a remote RAID 5 server. In this case, my local server acts as a client to the RAID server. Hourly, daily, weekly, and monthly backups are then a snap, again, using `rsync`. :)

Setting up NFS Version 3 (I have not bothered to learn V4 yet) requires simultaneous configuration of the NFS server and client, in addition to poking holes through the firewall to allow for NFS connections and setting up SELinux correctly. Both the firewall and SELinux should have already been set up. See the Firewall and SELinux pages for details. And from now on, you should already how to use `sudo` and various text editors to make changes to configuration files. I will assume that you can do this easily on your own from this point forward. See previous web pages (sudo, shell, etc.) for help if you need it.

Server Setup
To start, first configure the server:

Server 1. Add the following entry to **`/etc/exports`**:

```
/home [ip address of client computer](rw,root_squash)
/var/spool/mail [ip address of client computer](rw,root_squash)
```

When you make changes to `/etc/exports`, you should enter the following command to re-read the export table:

```
~> sudo exportfs -rv
```

On the server side, you can decide not to trust any requests made by `root` on the client. This is done using the `root_squash` option in the `/etc/exports` file. (This is, in fact, the default.) It should always be turned on unless you have a very good reason not to. To turn it off use the `no_root_squash` option.

Now, if a user with UID 0 (i.e., `root`'s user number) on the client attempts to access (read, write, delete) the file system, the server substitutes the UID of the server's "`nobody`" account. This means that the `root` user on the client cannot access or change files that only `root` on the server can change. This is good. You should probably use `root_squash` on all the file systems you export. "But the `root` user on the client can still use `su` to become any other user and access and change that user's files!" you say. To which the answer is: "Yes, and that's the way it is, and has to be, with Unix/Linux and NFS." This also has one important implication: all important binaries and other files should be owned by `root`, and not `bin` or other non-root accounts, since the only account the client's `root` user cannot access is the server's `root` account. In the `exports(5) man` page, there are several other squash options listed so that you can decide to mistrust whomever you (don't) like on the client computers.

The TCP ports 1-1024 are reserved for `root`'s use (and therefore are sometimes referred to as "secure ports"). A non-root user cannot bind these ports. Adding the secure option to an `/etc/exports` means that it will only list requests coming from ports 1-1024 on the client, so that a malicious non-root user on the client cannot come along and open up a spoofed NFS dialog on a non-reserved port. This option is set by default.

Server 2. Add the following entry to **`/etc/hosts.deny`**:

```
lockd:ALL
mountd:ALL
rpcbind:ALL
rquotad:ALL
statd:ALL
```

If you have configured NFS on RedHat/Fedora before, you will notice that `portmap` is missing in Fedora 7. This is simply because `rpcbind` is now being used in its place, which now contains the `portmapper` utility.

The `portmapper` keeps a list of what services are running on what ports. When an RPC service is started, it tells `rpcbind` the address at which it is listening, and the RPC program numbers it is prepared to serve. When a client wishes to make an RPC call to a given program number, it first contacts `rpcbind` on the server machine to determine the address where RPC requests should be sent. Basically, the `portmapper` via `rpcbind` maintains a list that is used by a connecting machine to see what ports it wants to talk to to access certain services.

The `portmapper` is not in as bad a shape as a few years ago, but it is still a point of worry for many system admins. The `portmapper`, like NFS and NIS, should not really have connections made to it outside of a trusted local area network. If you have to expose them to the outside world — be careful, and keep up diligent monitoring of those systems. Therefore, by editing the `/etc/hosts.deny` file, everyone is denied access to `portmap`, `lockd`, `mountd`, etc., which are all used for NFS connections. The files `/etc/hosts.allow` and `/etc/hosts.deny` take effect immediately after you save them. No daemon needs to be restarted.

Closing the `portmapper` to everyone is a bit drastic, so we will open it again (slightly) by editing `/etc/hosts.allow`. It should basically list all machines that should have access to your `portmapper`. On a run-of-the-mill Linux system, there are very few machines that need any access for any reason. The `portmapper` administrates `nfsd`, `mountd`, `ypbind/ypserv`, `rquotad`, `lockd` (which shows up nowdays as `nlockmgr`), `statd` (which now shows up as `status`) and "`r`" services like `ruptime` and `rusers`. Of these, only `nfsd`, `mountd`, `ypbind/ypserv`, and perhaps `rquotad`, `lockd`, and `statd` are of any consequence. Only machines that need to access services on your machine should be allowed to.

IMPORTANT: Do not put anything but **IP NUMBERS** in the lines of these files. Host name lookups can indirectly cause `portmapper` activity which will trigger host name lookups. This can indirectly cause `portmapper` activity, which will trigger...

Server 3. Based upon the above information, add the following entry to **`/etc/hosts.allow`**:

```
lockd:[ip address of client #1],...,[ip address of client #N]
mountd:[ip address of client #1],...,[ip address of client #N]
rpcbind:[ip address of client #1],...,[ip address of client #N]
rquotad:[ip address of client #1],...,[ip address of client #N]
statd:[ip address of client #1],...,[ip address of client #N]
```

As previously mentioned, this prevents hosts from other networks from connecting to any of the NFS-related daemons.

NOTE: In speaking with Vijay Rao, we have come to the conclusion that the current version of `rpcbind` that is being used in Fedora 7 (0.1.4-6.fc7) lacks `libwrap` support, meaning that `rpcbind` is not properly honoring `/etc/hosts.{allow|deny}`. This means that `rpcbind` will bind to a specific UDP interface, but not discriminate between TCP interfaces, making the above entries to `/etc/hosts.deny` and `/etc/hosts.allow` kind of pointless. I did not notice the problem earlier, because the TCP ports were already locked down by our College firewall. However, Vijay did find out the following information about `rpcbind`:

- `rpcbind` is based upon Sun's `rpcbind/sunrpc` code.
- Wietse Venema originally added `libwrap` support to `rpcbind`.
- Wietse's additions added bugs, and subsequent versions of `rpcbind` removed `libwrap` support.

If you want, you can compile `rpcbind` from the source `.rpm` file. The package does not allow `libwrap` to be easily configured at compile time, but other linux distributions apparently have a patch for this:

http://lists.pld-linux.org/mailman/pipermail/pld-cvs-commit/Week-of-Mon-20070507/154482.html

Personally, I have chosen not to recompile `rpcbind`, simply because the TCP Ports are already blocked by our College's firewall. However, if security is an issue, I would strongly suggest compiling `rpcbind` by hand until a new version is released that fixes this problem.

Server 4. Setting up NFS to use Fixed IP Ports.

When setting up a firewall for Linux systems running NFS, you hit the problem that some of the TCP/IP and UDP ports used by components of the services are randomly generated as part of the Remote Procedure Call (RPC) mechanism. On a system that is running with the NFS service active, the ports used by the components of the services can easily be listed using:

```
~> rpcinfo -p
```

The output will look something like:

```
   program vers proto   port
    100000    2   tcp    111  portmapper
    100000    2   udp    111  portmapper
    100024    1   udp  32814  status
    100024    1   tcp  33024  status
    100011    1   udp    670  rquotad
    100011    2   udp    670  rquotad
    100011    1   tcp    673  rquotad
    100011    2   tcp    673  rquotad
    100003    2   udp   2049  nfs
    100003    3   udp   2049  nfs
    100021    1   udp  32816  nlockmgr
    100021    3   udp  32816  nlockmgr
    100021    4   udp  32816  nlockmgr
    100005    1   udp  32818  mountd
    100005    1   tcp  33025  mountd
    100005    2   udp  32818  mountd
    100005    2   tcp  33025  mountd
    100005    3   udp  32818  mountd
    100005    3   tcp  33025  mountd
```

The listing shows the IP ports for the various versions of the services used in the 4th column. If you view this listing on different systems, or even after rebooting the same one, you will find that most of the port numbers are different. This is a real problem when configuring the firewall, which tends to assume that known port numbers are used for the services being configured. To make it possible to configure a firewall that effectively controls NFS, it is useful to be able to "tie" down the ports used by these services (`portmap`, `lockd`, `mountd`, `rquotad`, and `statd`) to fixed values.

The following table lists the needed NFS daemons and summarizes the relevant information for each of the necessary ports. The subsections that follow provide more detail for tying them down, if necessary.

Daemon Name	RPM Package	Standard Port	Suggested Port	What to Change
portmapper	rpcbind	111	111	Nothing
rpc.statd	nfs-utils	Random	4000	/etc/sysconfig/nfs
rpc.nfsd	nfs-utils	2049	2049	Nothing
rpc.lockd	nfs-utils/kernel	Random	4001	/etc/modrpobe.conf*
rpc.mountd	nfs-utils	Random	4002	/etc/sysconfig/nfs
rpc.rquotad	quota	Random	4003	/etc/sysconfig/nfs

*`rpc.lockd` cannot be configured in `/etc/sysconfig/nfs`, as it is compiled as a kernel module in Fedora 7.

NOTE: Now you can understand why we previously poked holes in the firewall using very specific port numbers for NFS. Again, see the Firewall page for details.

Server 4.1 Portmapper [Standard Port: 111]

As of Fedora 7, the `portmapper` is implemented by the `rpcbind` program, which is part of the "`rpcbind`" RPM package. The service uses Port 111 on both the TCP and UDP Internet Protocols.

The `portmapper` provides the mapping between application names and IP ports, and is therefore analogous to the `/etc/services` file except that it relates to Remote Procedure Call (RPC) programs only.

Firewall rules that refer to portmapper should reference TCP/IP and UDP packets on Port 111.

Server 4.2 Network Status Monitor [Random Port — Suggested Port: 4000]

The `rpc.statd` server implements the Network Status Monitor (NSM) RPC protocol. This service is somewhat misnamed, since it does not actually provide active monitoring, as one might suspect. Instead, NSM implements a reboot notification service. It is used by the NFS file locking service, `rpc.lockd`, to implement lock recovery when the NFS server machine crashes and reboots.

The `rpc.statd` program is part of the "`nfs-utils`" RPM package.

While `rpc.statd` is normally allocated a random port number by the portmapper, it is possible to configure a fixed port by supplying the appropriate variable name to **`/etc/sysconfig/nfs`**:

```
STATD_PORT=4000
```

Once the above change has been made, firewall rules should refer to TCP/IP and UDP packets on the chosen port.

Server 4.3 NFS Daemon [Standard Port: 2049]

The `rpc.nfsd` program implements the user-level part of the NFS service. The main functionality is handled by the `nfsd.o` kernel module; the user-space program merely starts the specified number of kernel threads.

The `rpc.nfsd` program normally listens on Port 2049, so firewall rules can be created to refer to that port (unless it is changed from the default value). If you want to change it, edit `/etc/sysconfig/nfs` and supply a port number using the `STATD_OUTGOING_PORT` variable. Just remember that if you do change it, you will need to update the firewall rules to reflect this.

Server 4.4 NFS Lock Manager [Random Port — Suggested Port: 4001]

The NFS lock manager is a kernel module. It implements the NLM (NFS Lock Manager) part of the NFS subsystem, and is used for handling file and resource locks of various types. This component is sometimes referred to as "`rpc.lockd`", and shows up in the output of `rpcinfo` as "`nlockmgr`" (hey - consistency would only make life boring!).

On systems where the lock manager is implemented as a loadable module, as is the case for Fedora Core 6, the port number used can be set at module load time, and is configured by adding (or editing) a line in **`/etc/modprobe.conf`**:

```
options lockd nlm_udpport=4001 nlm_tcpport=4001
```

This sets the UDP and TCP/IP Port numbers. Conventionally, these two numbers should be set to the same value.

If your system has the the lockd code compiled into the main kernel binary rather than as a loadable module, then the settings in `/etc/modprobe.conf` will not work. You will need to add the parameters "`lockd.udpport=4001 lockd.tcpport=4001`" to the kernel command line in the `lilo` or `grub` configuration file instead.

NOTE: In order for the changes to take effect, the module must be reloaded if it is already in use. You can use the commands `rmmod` and `modprobe` to reload the `lockd` module; however if there are module dependencies currently in use, a system restart may be required. For me this was the case, so go ahead and give your system a reboot also.

Technically, these options could be set in `/etc/sysconfig/nfs` as well, **but only if lockd is not loaded as a kernel module**. If you want to try it, add the following lines to `/etc/sysconfig/nfs`:

```
LOCKD_TCPPORT=4001
LOCKD_UDPPORT=4001
```

As before, configure the Firewall to manage the port number just selected for both the TCP/IP and UDP Ports.

Server 4.5 Mount [Random Port — Suggested Port: 4002]

The `rpc.mountd` program implements the NFS mount protocol. When receiving a mount request from an NFS client, it checks the request against the list of currently exported file systems. If the client is permitted to mount the file system, `rpc.mountd` obtains a file handle for the requested directory and returns it to the client.

While `rpc.mountd` is normally allocated a random port number by the portmapper, it is possible to configure a fixed port number by supplying the "`-p`" command line option when the program is launched. This is done by editing **`/etc/sysconfig/nfs`**:

```
MOUNTD_PORT=4002
```

Once this edit has been made, configure the firewall to manage the port number selected for both TCP/IP and UDP Ports.

Server 4.6 Quota [Random Port — Suggested Port: 4003]

`rquotad` is an RPC server that returns quotas for a user of a local filesystem mounted by a remote machine over NFS. It also allows setting of quotas on an NFS mounted filesystem. The results are used by `quota` to display user quotas for remote filesystems and by `edquota` to set them. The `rquotad` daemon is normally started at boot time from the system startup scripts.

There are two versions of `rpc.rquotad` that are commonly used with Linux systems: one is part of `nfs-utils`, and the other comes bundled with the "`quota`" package. You can fix the port number by editing **`/etc/sysconfig/nfs`**:

```
RQUOTAD_PORT=4003
```

Once this change has been made, configure the Firewall to manage the TCP/IP and UDP Port number selected.

Server 5. Configuring the Firewall (If you have not already!)

The firewall configuration file is found in `/etc/sysconfig/iptables`. Holes must be poked through Ports 111, 2049, and 4000:4003 for both the TCP and UDP Internet Protocols. See the Firewall page for details.

As a final note, make sure that the "`REJECT`" and the "`COMMIT`" lines mentioned are the very last lines at the end of the file and are not found elsewhere. Otherwise, when the rules are implemented, everything after "`COMMIT`" will be ignored and NFS will not work if the "`REJECT`" line is located before the NFS ones. Once the changes are made to the firewall, restart it:

```
~> sudo service iptables restart
```

Server 6. Start portmapper, the NFS daemon, and the other related daemons:

With all of the changes made, we are ready to start up the appropriate daemons:

```
~> sudo service portmap restart
~> sudo service nfs restart
~> sudo service nfslock restart
```

You should not see any "**`FAILED`**" messages when you restart the daemons if you set everything up correctly. Everything should be "**`OK`**".

Server 7. Verification

Make sure everything is working properly. Let's check the RPC information to make sure that all of the ports are tied down correctly:

```
~> rpcinfo -p
```

You should see something similar to:

```
   program vers proto   port
    100000    2   tcp    111  portmapper
    100000    2   udp    111  portmapper
    100024    1   udp   4000  status
    100024    1   tcp   4000  status
    100021    1   udp   4001  nlockmgr
    100021    3   udp   4001  nlockmgr
    100021    4   udp   4001  nlockmgr
    100021    1   tcp   4001  nlockmgr
    100021    3   tcp   4001  nlockmgr
    100021    4   tcp   4001  nlockmgr
    100011    1   udp   4003  rquotad
    100011    2   udp   4003  rquotad
    100011    1   tcp   4003  rquotad
    100011    2   tcp   4003  rquotad
    100003    2   udp   2049  nfs
    100003    3   udp   2049  nfs
    100003    4   udp   2049  nfs
    100003    2   tcp   2049  nfs
    100003    3   tcp   2049  nfs
    100003    4   tcp   2049  nfs
    100005    1   udp   4002  mountd
    100005    1   tcp   4002  mountd
    100005    2   udp   4002  mountd
    100005    2   tcp   4002  mountd
    100005    3   udp   4002  mountd
    100005    3   tcp   4002  mountd
```

Notice that all of the daemons are now tied down to their appropriate port numbers. If `nlockmgr` is misbehaving, it is because it's being loaded as a kernel module and you will need to reboot. `:(`

Next, check the mount information to verify that the appropriate directories will be exported:

```
~> showmount -e
```

You should see something like:

```
Export list for [server hostname]:
/home           [client ip address #1],...,[client ip address #N]
/var/spool/mail [client ip address #1],...,[client ip address #N]
```

The `netstat -tauap` command will also give you a good idea of which TCP and UDP ports are listening for connections.

Verify that the portmapper and all NFS-related daemons start at boot time. You can do this using `system-config-services` via `sudo`, or with the following commands:

```
~> sudo chkconfig portmap off
~> sudo chkconfig nfs off
~> sudo chkconfig nfslock off
```

```
~> sudo chkconfig --level 345 portmap on
~> sudo chkconfig --level 345 nfs on
~> sudo chkconfig --level 345 nfslock on
```

Again, you should **make sure** that the firewall settings for iptables are properly set so that the connections are not automatically blocked!

With that, the NFS server should now be setup.

Client Setup
Configuring the client computer is much easier. The client must be running the portmapper service and the `rpc.statd` service. If you need file locking, you must also be running the NFS lock daemon `rpc.lockd`. You do not need to be running `rquotad`, `nfsd`, or `mountd`. By running `/etc/init.d/portmap` and `/etc/init.d/nfslock`, everything should be ready for an NFS mount, after making some additional changes:

Client 1. Add the following entry to **`/etc/hosts.deny`**:

```
lockd:ALL
mountd:ALL
rpcbind:ALL
rquotad:ALL
statd:ALL
```

Client 2. Add the following entry to **`/etc/hosts.allow`**:

```
lockd:[ip address of server]
mountd:[ip address of server]
rpcbind:[ip address of server]
rquotad:[ip address of server]
statd:[ip address of server]
```

Client 3. Tie down the necessary ports.

`portmap` is already tied down to Port `111`, so tie down `rpc.statd` by editing **`/etc/sysconfig/nfs`**:

```
STATD_PORT=4000
```

Client 4. Configuring the firewall.

Since the client only needs to be running `portmap` and `nfslock`, the only ports that need opening are TCP and UPD Ports 111 and 4000. Modify the firewall configuration file, **`/etc/sysconfig/iptables`**:

```
# Allow NFS Connections
-A RH-Firewall-1-INPUT -m state --state NEW -m tcp -p tcp --dport 111 -j ACCEPT
-A RH-Firewall-1-INPUT -m state --state NEW -m tcp -p tcp --dport 4000 -j ACCEPT
-A RH-Firewall-1-INPUT -m state --state NEW -m udp -p udp --dport 111 -j ACCEPT
-A RH-Firewall-1-INPUT -m state --state NEW -m udp -p udp --dport 4000 -j ACCEPT

-A RH-Firewall-1-INPUT -j REJECT --reject-with icmp-host-prohibited
COMMIT
```

Again, make sure that the "`REJECT`" and "`COMMIT`" lines are the very last lines at the end of the file and are not found elsewhere. Then, reload the firewall rules:

```
~> sudo service iptables restart
```

Client 5. Verification

Check RPC connectivity to the NFS server from the client. The following commands will be helpful. Make sure they are being run from the client computer:

```
~> ping [server]
~> showmount -e [server]
~> rpcinfo -p [server]
~> tracepath [server]/2049
```

Based on the output of these commands, you should be able to see if the client will be able to make an NFS connection to the server or not.

Also, make sure that the `portmap` and the `nfslock` daemons start at boot time:

```
~> sudo chkconfig portmap off
~> sudo chkconfig nfs off
~> sudo chkconfig nfslock off
```

```
~> sudo chkconfig --level 345 portmap on
~> sudo chkconfig --level 345 nfslock on
```

Remember: You do not need the NFS daemon on the client!

Client 6. Mount Points

Make specific mount points for the directories to be mounted. I like to mount everything in `/mnt`:

```
~> sudo mkdir /mnt/nfs/[server]/home
~> sudo mkdir /mnt/nfs/[server]/mail
```

Next, we need to let the `netfs` service know that it is supposed to mount the NFS directories. This is done by first editing **`/etc/fstab`**:

```
[server]:/home            /mnt/nfs/[server]/home  nfs    rw,hard,intr,bg    0 0
[server]:/var/spool/mail  /mnt/nfs/[server]/mail  nfs    rw,hard,intr,bg    0 0
```

The `rw` option allows the client to both read and write to the mounted directory. The `hard` option ensures that a process or program accessing a file on an NFS mounted file system cannot be interrupted or killed if the server crashes (which causes the system to hang), unless we specify the `intr` option, which we have done. When the NFS server is back online, the program will continue uninterrupted where it left off. The other possibility is to use the `soft` option, but this leads to corrupted files and data loss. :(Finally, the `bg` option allows the client to continue to try to mount the directories in the background if it cannot establish a connection to the server, allowing for NFS to gracefully fail instead of just hanging the computer. This is an extremely useful option for my laptop, since it often is being used on some wireless network where I do not have access to the server.

Client 7. Mount the directories!

From now on at boot time, if the client can establish a connection to the server, the NFS directories will be mounted. To mount the NFS file systems now, we simply restart the `netfs` (Network File Systems) service:

```
~> sudo service netfs restart
```

If everything is setup properly, you should see something similar to:

```
Mounting NFS filesystems:                                  [  OK  ]
Mounting other filesystems:                                [  OK  ]
```

You can now change your directory to, say, `/mnt/nfs/[server]/home` on the client computer and you will access `/home` on the server. Yeah!

References:

http://www.redhat.com/magazine/010aug05/departments/tips_tricks
http://www.lowth.com/LinWiz/nfs_help.html
http://www.vanemery.com/Linux/NFS-Van.html
http://nfs.sourceforge.net/nfs-howto/