NFS: Sun’s Network File System

Goals:

- Machine and OS independence
- Crash recovery
- Transparent access
- UNIX semantics maintained on client
- Reasonable performance
NFS — transparency

- Remote filesystems need to look exactly like any other filesystem
- But the existing kernel only supported one filesystem
- Solution: A new layer of indirection (vnodes). Each filesystem provides its own implementation of the vnode interface (open, close, rename, etc.)
NFS — crash recovery

NFS is stateless.

- Every NFS command is self-contained — no session information necessary
- No open — just lookup to obtain an fhhandle
- No close — just stop talking to server!
- If a server crashes, the client just repeats request until the server comes back up and answers
- If a client crashes, server doesn’t care
NFS — crash recovery (cont’d)

NFS is idempotent.

- Issuing the same request more than once has no extra effect
- Duplicate requests are no problem — read or write the same thing twice, and nothing will be different
If NFS were stateful

- Server would maintain state for each client (which files are open, where client is in each file, etc.)
- Client would have to detect server crashes to rebuild server’s state
- Server would have to detect client crashes to discard client’s state
- Would make crash recovery much harder!
NATs — network address translators

- There are roughly $2^{32}$ IP addresses
- But many go unused due to partitioning scheme (MIT has $2^{24} \approx 16.7$ million IP addresses!)
- IPv6 supports $2^{128}$ IP addresses but requires changes to infrastructure and end-hosts
- NATs: a stop-gap (maybe permanent?) solution requiring no changes to routers or hosts
How NATs work

- Each host in an organization’s subnet gets a “fake,” not necessarily globally unique, IP address (10.x.x.x, 192.168.x.x, 172.16-31.x.x)
- Border router gets a set of real IP addresses (let’s say 18.18.18.x)
- Border router intercepts packets to Internet, changing source address to one of its IPs (18.18.18.3)
- Border router intercepts packets from Internet, changing destination address to the appropriate internal IP
How NATs work, cont’d

Which internal IP is the real destination? Border router must keep a table remembering the appropriate internal IP for each connection.

<table>
<thead>
<tr>
<th>External IP</th>
<th>Internal IP</th>
</tr>
</thead>
<tbody>
<tr>
<td>18.18.18.1</td>
<td>192.168.0.100</td>
</tr>
<tr>
<td>18.18.18.2</td>
<td>192.168.4.34</td>
</tr>
<tr>
<td>18.18.18.3</td>
<td>192.168.1.205</td>
</tr>
</tbody>
</table>

What if the NAT doesn’t have enough IP addresses for all the internal users (e.g., more than 256 can be online at once)? Use port numbers too
What NATs break

- Protocols with IP addresses in data, e.g., FTP — each host knows only its *fake* IP addresses
- Servers inside a NAT
- New protocols — NATs may need to know about them
- Plenty of other stuff (see paper)
Google

- Precision: relevance of returned information
- Recall: breadth of returned information
- Recall is easy enough by brute force. Precision is tough!
- Google utilizes the link structure of the Web to gauge the relevance of each page (PageRank)
  - If lots of people link to a page, it’s probably an important page
  - Links from important pages are probably important
  - PageRanks are eigenvalues of normalized link matrix
Google — miscellany

- Factors Google uses to improve precision:
  - Proximity
  - Anchor text
  - Text colors, fonts, sizes

- Results for multi-word query: intersection of results for individual search terms, with proximity taken into account

- Google is highly parallel — 8,000 PCs, each with two 80-gigabyte disks (1,280 terabytes!)