Go further, faster®

Peer to Peer NFS

Bryan Schumaker
bjschuma@netapp.com

Problem

- Several clients in a cluster boot and attempt to read the same set of libraries from a single NFS server
- Server bandwidth is overloaded from serving the same files over and over
- Could stripe data over pNFS to several data servers
  - Moves the bottleneck without fixing it
- Current solutions include cachefs, data replication, and proxy servers
  - Requires installing and configuring additional tools

Current Solutions

- Use pNFS
  - Stripe data over several DSs
  - Might just move bottleneck without fixing the problem
- Use additional tools
  - Preload data using cachefs
  - Data replication across multiple servers
  - Proxy servers
- Netapp has flexcache

New Solution: Peer to Peer

- Currently exists as a draft written by Trond Myklebust in 2009
  - Initially targeted for 4.2
  - Never managed to complete the prototype
- Clients can serve data out of their disk cache
  - Act as an adhoc DS
- The first wave of clients boot and read data from the MDS
- The second wave is referred to the first set of clients for the data
- MDS can only refer a client to a DS that holds a read delegation on the file

Benefits

- No additional tools required
- Any pNFS enabled client can read from an adhoc data server without changes
- Preexisting servers don't need full pNFS support
  - LAYOUTGET, GETDEVICEINFO, and the new p2p operations

New operations

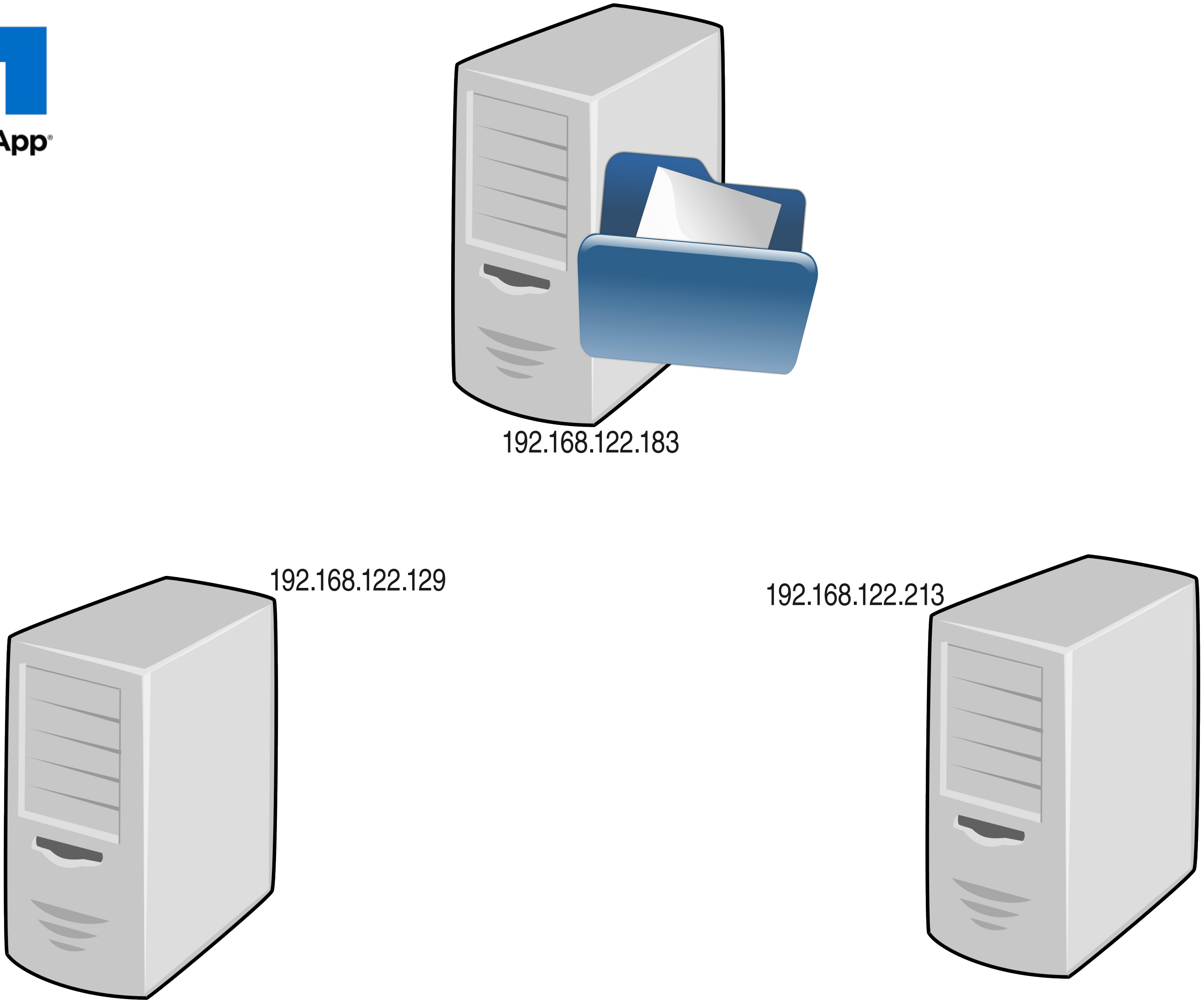
- REGISTER_DS
  - Sent by a client willing to act as a data server
- UNREGISTER_DS
  - Sent by a client when they are no longer acting as a DS
- PROXY_OPEN
  - Sent by a DS to the MDS to check if the client has access to a file
- CB_PROXY_REVOKE
  - Sent by the MDS to the DS to alert that a client's state has expired

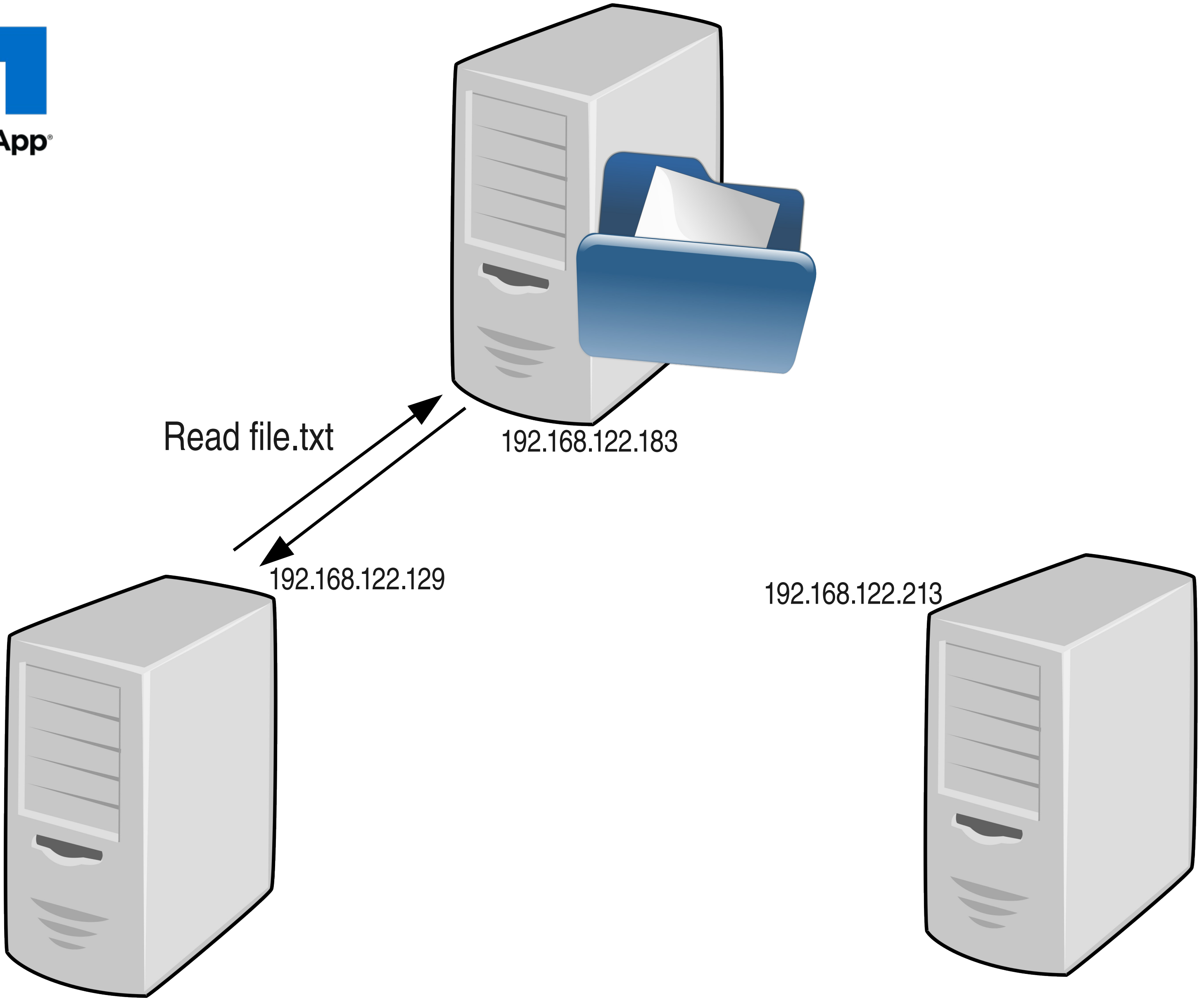
Prototype goals

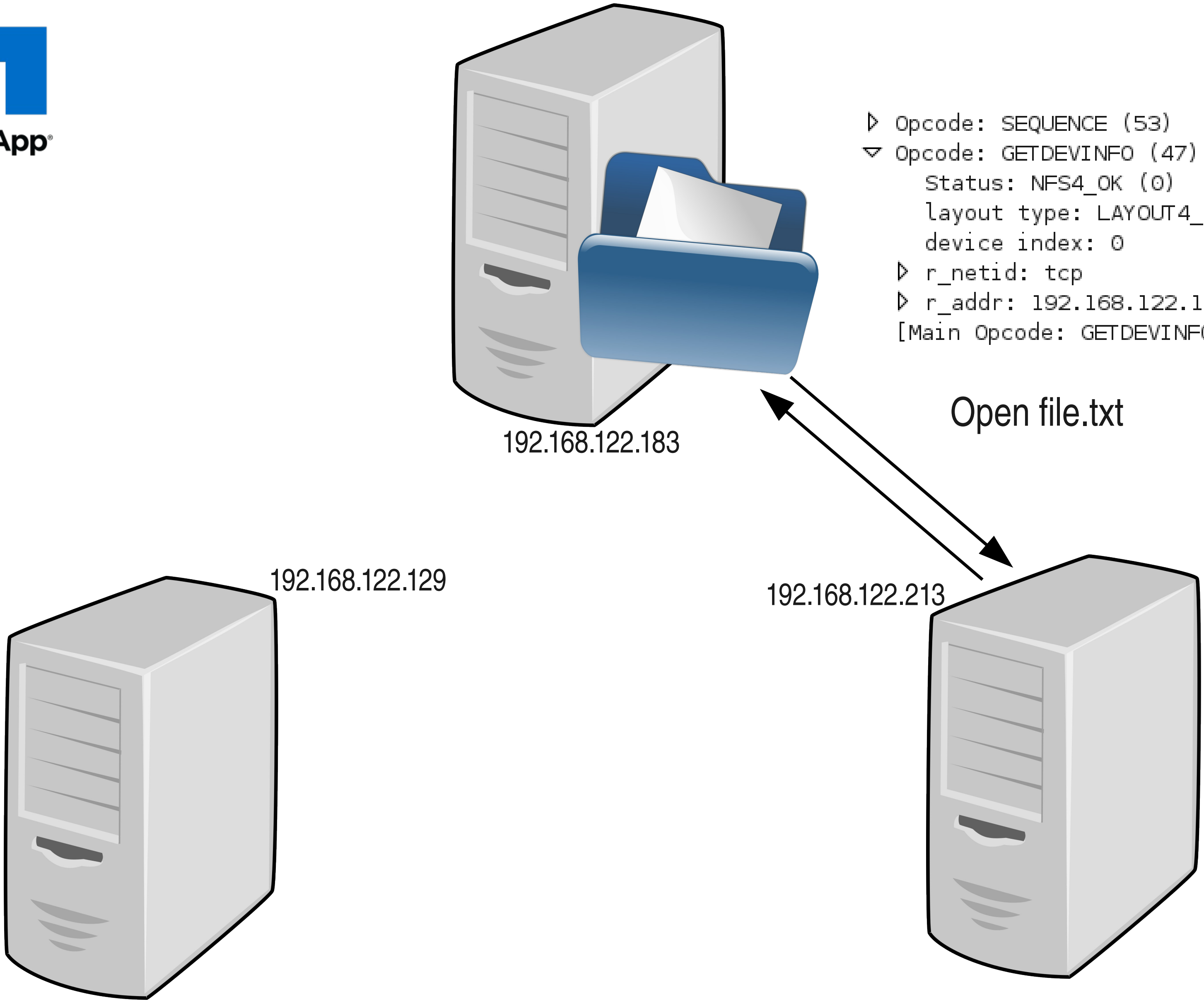
- Proof of concept
- Check that scale out with large number of clients works
- Check draft correctness

Prototype Implementation

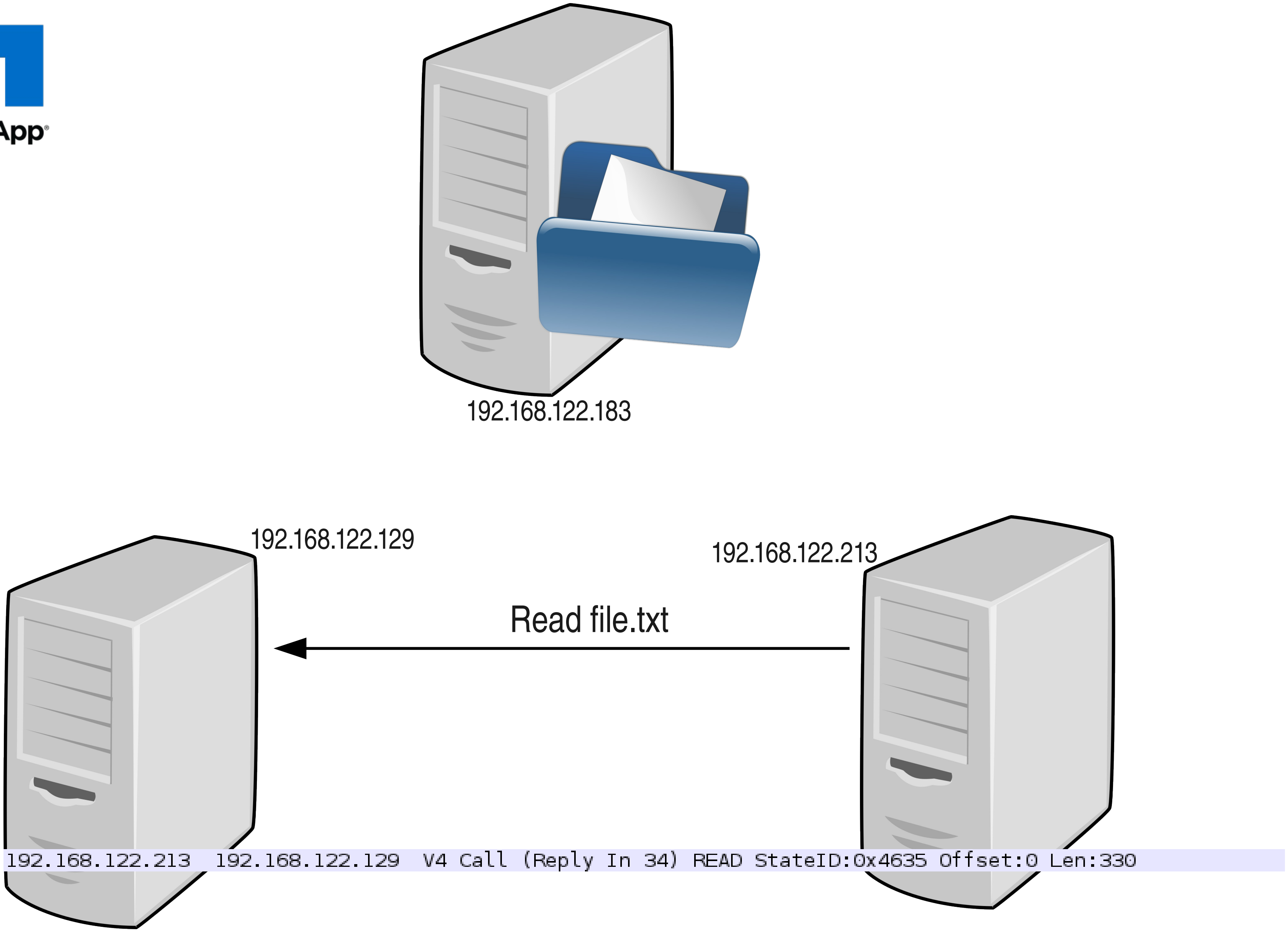
- Started with the most recent pNFS Linux server code
  - Currently maintained by Benny Halevy
  - Not merged into Linux yet
- Added in each new p2p operation until it worked
  - Some PROXY_OPEN code was based on work from summer 2008
- Most changes made to nfsd code
  - Added p2p functions to file layout module
- Server routing information stored in filehandle
  - Added 64-bit cookie to the front
- Don't need to re-export an nfs mount
  - Small hack into the VFS to find the requested file


192.168.122.183
192.168.122.129
192.168.122.213

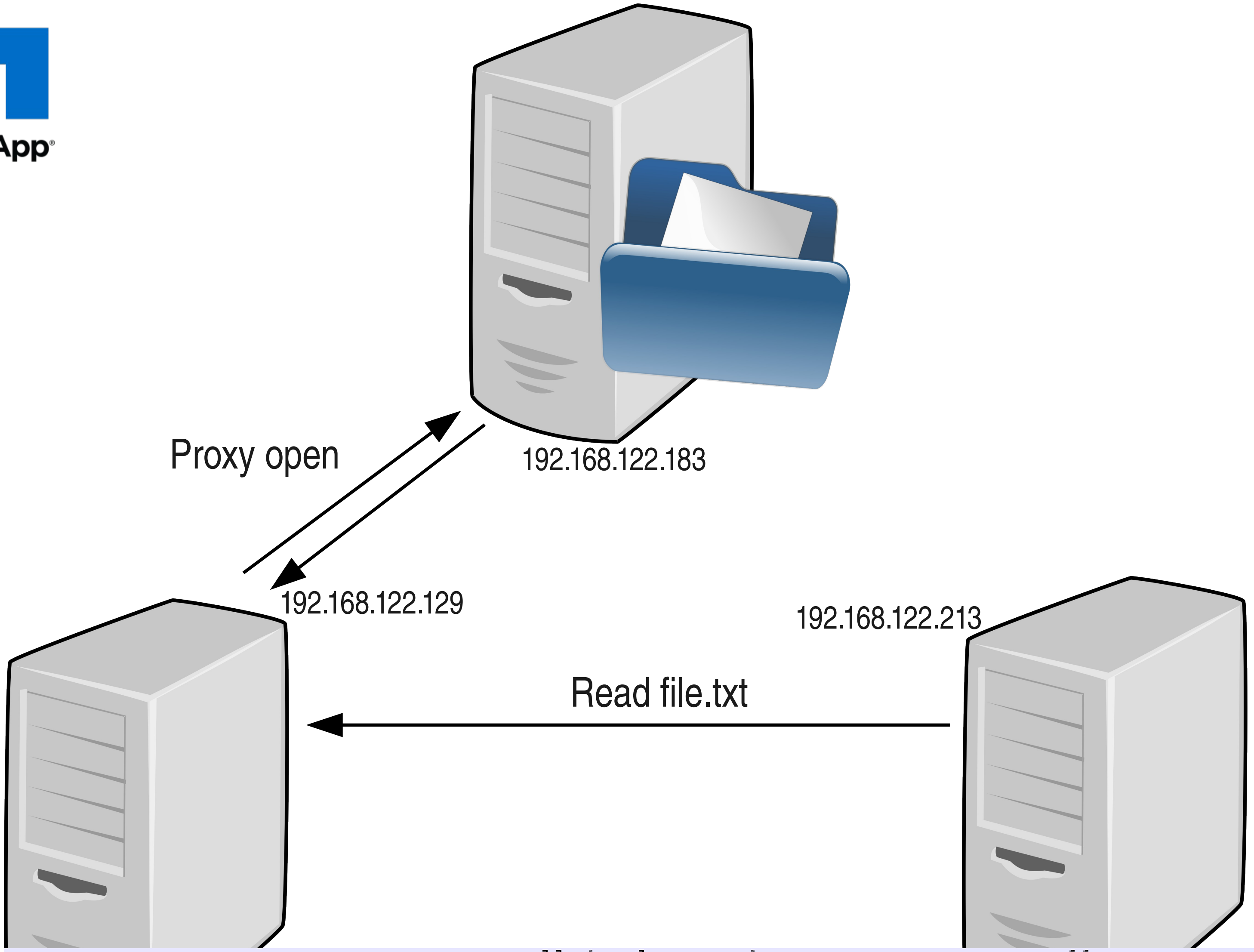

Read file.txt
192.168.122.183
192.168.122.129
192.168.122.213



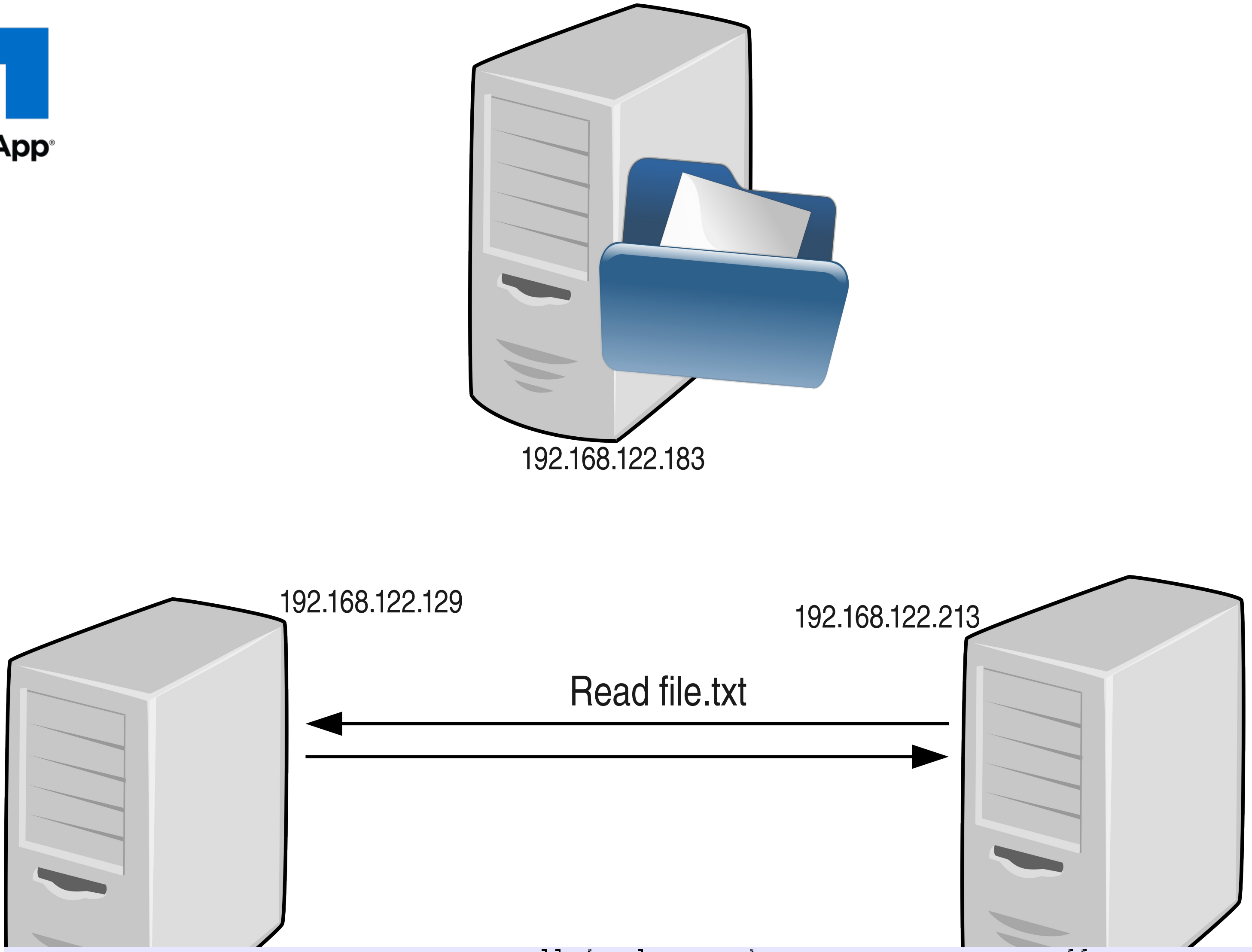
```
▷ Opcode: SEQUENCE (53)
▽ Opcode: GETDEVINFO (47)
    Status: NFS4_OK (0)
    layout type: LAYOUT4_NFSV4_1_FILES (1)
    device index: 0
  ▷ r_netid: tcp
  ▷ r_addr: 192.168.122.129.8.1
  [Main Opcode: GETDEVINFO (47)]
```



```
192.168.122.213  192.168.122.129  V4 Call (Reply In 34) READ StateID:0x4635 Offset:0 Len:330
```



```
192.168.122.213  192.168.122.129  V4 Call (Reply In 34) READ StateID:0x4635 Offset:0 Len:330
192.168.122.129  192.168.122.183  V4 Call (Reply In 30) SEQUENCE | PUTFH
192.168.122.183  192.168.122.129  V4 Reply (Call In 29) SEQUENCE | PUTFH
192.168.122.129  192.168.122.183  V4 Call (Reply In 33) ACCESS FH:0x70825b1d, [Check: RD MD XT XE]
192.168.122.183  192.168.122.129  V4 Reply (Call In 32) ACCESS, [Access Denied: XE], [Allowed: RD MD XT]
```



```
192.168.122.213  192.168.122.129  V4 Call (Reply In 34) READ StateID:0x4635 Offset:0 Len:330
192.168.122.129  192.168.122.183  V4 Call (Reply In 30) SEQUENCE | PUTFH
192.168.122.183  192.168.122.129  V4 Reply (Call In 29) SEQUENCE | PUTFH
192.168.122.129  192.168.122.183  V4 Call (Reply In 33) ACCESS FH:0x70825b1d, [Check: RD MD XT XE]
192.168.122.183  192.168.122.129  V4 Reply (Call In 32) ACCESS, [Access Denied: XE], [Allowed: RD MD XT]
192.168.122.129  192.168.122.213  V4 Reply (Call In 28) READ
```

Demo

Demo - Mount

```
pnfsd dmesg
-----------
[   63.650944] 1612 fs/nfsd/nfs4xdr.c nfsd4_decode_register_ds client addr? 192.168.122.129.8.1
over? tcp
[   63.650954] 1683 fs/nfsd/nfs4pnfsd.c print_stateid (506dbacd/00000001/00000000/00000001)
[   63.674751] 1612 fs/nfsd/nfs4xdr.c nfsd4_decode_register_ds client addr? 192.168.122.213.8.1
over? tcp
[   63.674764] 1683 fs/nfsd/nfs4pnfsd.c print_stateid (506dbacd/00000002/00000001/00000001)

p2pds dmesg
-----------
[   62.067956] nfs4filelayout_init: NFSv4 File Layout Driver Registering...
[   62.069453] 5997 fs/nfs/nfs4xdr.c decode_register_ds Stateid:
(506dbacd/00000001/00000000/00000001)

p2pclient dmesg
---------------
[   56.651721] nfs4filelayout_init: NFSv4 File Layout Driver Registering...
[   56.652975] 5997 fs/nfs/nfs4xdr.c decode_register_ds Stateid:
(506dbacd/00000002/00000001/00000001)
```

Demo - Read

```
pnfsd dmesg
-----------
[  111.953814] 1874 fs/nfsd/nfs4pnfsd.c pnfs_p2p_find_deviceid Returning p2p devid:
5795493685126758401 (1349368525/1, 192.168.122.129.8.1)
[  111.954159] 1833 fs/nfsd/nfs4pnfsd.c find_client_by_devid devid? 5795493685126758401
(1349368525/1)
[  111.954162] 1906 fs/nfsd/nfs4pnfsd.c pnfs_p2p_set_device_daddr Mapping devid 5795493685126758401
-> IP 192.168.122.129.8.1
[  111.980182] 1634 fs/nfsd/nfs4xdr.c nfsd4_decode_proxy_open

p2pds dmesg
-----------
[  110.397760] Proxy-opening filehandle at ffff88003dbcc600 is 36 bytes, crc: 0xcf6db39d:
[  110.398007]  00000000 00000000 0100094d 00000000
[  110.398153]  56180000 00000000 05000000 00000000
[  110.398299]  05220000
[  110.398756] Server returned new filehandle at ffff88003dbcc600 is 28 bytes, crc: 0x70825b1d:
[  110.398987]  0100014d 00000000 56180000 00000000
[  110.399145]  05000000 00000000 05220000
```

Demo - Unmount

```
pnfsd dmesg
-----------
[  118.957851] 1626 fs/nfsd/nfs4xdr.c nfsd4_decode_unregister_ds
[  118.957857] 1683 fs/nfsd/nfs4pnfsd.c print_stateid (506dbacd/00000001/00000000/00000001)
[  118.957858] 1664 fs/nfsd/nfs4pnfsd.c unregister_p2p_client Unregistering client:
192.168.122.129.8.1
[  118.957860] 1673 fs/nfsd/nfs4pnfsd.c unregister_p2p_client Client: 192.168.122.129.8.1 had 1
proxy-opened files
[  118.957863] 4058 fs/nfsd/nfs4xdr.c nfsd4_encode_unregister_ds err? 0
[  118.958637] 1917 fs/nfsd/nfs4pnfsd.c pnfsd_p2p_expire_client
[  119.605228] 1626 fs/nfsd/nfs4xdr.c nfsd4_decode_unregister_ds
[  119.605242] 1683 fs/nfsd/nfs4pnfsd.c print_stateid (506dbacd/00000002/00000001/00000001)
[  119.605245] 1664 fs/nfsd/nfs4pnfsd.c unregister_p2p_client Unregistering client:
192.168.122.213.8.1
[  119.605246] 1673 fs/nfsd/nfs4pnfsd.c unregister_p2p_client Client: 192.168.122.213.8.1 had 0
proxy-opened files
[  119.605248] 4058 fs/nfsd/nfs4xdr.c nfsd4_encode_unregister_ds err? 0
[  119.606309] 1917 fs/nfsd/nfs4pnfsd.c pnfsd_p2p_expire_client

p2pds dmesg
-----------
[  118.024304] 1917 fs/nfsd/nfs4pnfsd.c pnfsd_p2p_expire_client
```

What's working

- REGISTER_DS
  - For all filesystems and files used by client
  - Don't have controls for specific files / filesystems
- UNREGISTER_DS
- PROXY_OPEN
  - Don't check user access permissions
- Serve files from disk cache
- DS rereads files that are no longer cached

Not implemented (yet)

- State recovery
  - CB_PROXY_REVOKE
  - Ran into bugs in the existing pNFS base code that created infinite recovery loops
  - Both bugs fixed, but no chance to get back to this yet

Future work

- Test in large scale environment
- Better p2pds selection
  - Currently refer to the first DS found
  - Instead, send complete list of registered DSs and have client choose
- Check all error cases
  - Machines are assumed to be non-malicious, mostly targeting data centers now
- User controls if client acts as a data server

Changes to current draft

- Current draft was submitted October 2010
- Need to add netid4 to REGISTER_DS arguments
  - Provides IP address and port for contacting adhoc DSs

NetApp®

Thank you