April 13, 2026 · 13 min read · By Yiğit Konur

mac-to-mac dosya sistemi: NFS dev için native SMB'den hızlı

software-engineering

ben de bir sürü insan gibi "openclaw" için bir Mac Mini aldım (lol).

geçen sene birkaç gün boyunca local LLM'lerin değerini abarttım ve sebepsiz birkaç bin dolar harcadım. ama işe yaradı: subagent'lar + daha ağır MCP client'larıyla Mac Mini sürekli açık duran local build makinem oldu, çoğu zaman MacBook'tan "vibe code" yapıp ona atıyorum.

problem: macOS'ta SMB üzerinden dosya paylaşımı büyük dosyalarda fena değil ama küçük dosyalarla dolu dev ağaçlarında (TypeScript, config, node_modules) acı verici yavaş. "canlı dosya sistemi" kısmını NFS'e geçirdim ve hem LAN'da hızlı hem de MacBook LAN'dan çıktığında sağ kalacak hâle gelene kadar tuneladım — işin zor yarısı burası oluyor ve NFS-on-macOS bloglarının çoğu tam bu kısımda yanılıyor.

bu workload için neden SMB yerine NFS

macOS'ta SMB küçük-dosya workload'larında zorlanıyor çünkü:

- Finder ve arkadaşları fazladan metadata işi yapıyor (extended attribute'lar dahil)
- SMB'de per-operation overhead daha yüksek (ve macOS'un implementation'ı "60k tiny files" için her zaman optimize hissettirmiyor)
- dizin listeleme bir sürü küçük round-trip'e dönüşüyor

NFS daha basit. benim LAN'ımda tuned NFS "büyük dizinleri listeleme"yi "bu niye bu kadar sürüyor"dan "tamam, kullanılabilir"e taşıdı.

-

test ettiğim setup

- server: Mac Mini (apple silicon), kablolu LAN, `192.168.1.200`
- client: MacBook (macOS), aynı network, LAN'a girip çıkıyor
- workload: ~60,000 dosya / ~8GB, çoğunluk TypeScript + config + node_modules
- rtt: ~4ms (Wi-Fi → switch → ethernet)

hiçbir ayarı kurcalamadan önce client profilini seç

aynı "NFS client" etiketinin altında tamamen farklı iki reçete var:

- **kablolu workstation** — sürekli açık, kablolu ethernet, hareket etmiyor, kapağı kapanmıyor. bir şey kaybolmadığı için agresif davranabilirsin.
- **mobil laptop** — uyuyor, network değiştiriyor, Wi-Fi olmayan odalara giriyor. client tarafındaki her knob server'ın *kesin* kaybolacağını varsaymak zorunda.

blogların çoğu workstation ayarlarını öneriyor ve dosyayı kapatıyor. bu ayarlar laptop'ta Raycast'i crash ettiriyor, Finder'ı beachball'a sokuyor ve `launchd` 'nin Wi-Fi'ına dakikada bir saldırmasına yol açıyor. server tuning ikisi için de aynı; yol client tarafında ayrılıyor.

server (Mac Mini): exports + nfsd tuning

/etc/exports

```
/Users/yigitkonur -alldirs -mapall=501:20 -network 192.168.1.0 -mask 255.255.255.0
```

önemli olanlar:

- `-alldirs`: sadece export kökünü değil, alt dizinleri de mount edebiliyorsun

-

- `-mapall=501:20`: client erişimlerinin hepsini tek bir local uid/gid'e mapliyor (tek-kullanıcılı dev için rahat)
- `-network ... -mask ...`: erişimi local subnet'inle sınırlıyor

not: `501:20` macOS'ta yaygın (ilk user + staff), ama garanti değil. kendi uid/gid'ini `id -u` / `id -g` ile al.

/etc/nfs.conf (server)

ini

```
nfs.server.mount.require_resv_port = 0
nfs.server.require_resv_port = 0
nfs.server.nfsd_threads = 16
nfs.server.async = 1
nfs.server.fsevents = 0
nfs.server.wg_delay = 0
nfs.server.wg_delay_v3 = 0
nfs.server.reqcache_size = 512
nfs.server.request_queue_length = 512
nfs.server.export_hash_size = 256
nfs.server.tcp = 1
nfs.server.udp = 0
nfs.server.user_stats = 0
nfs.server.bonjour = 0
nfs.server.verbose = 0
```

kısa "neden" tablosu:

-

daemon'u aç:

```bash
sudo nfsd enable
```

client — kablolu workstation profili

`/Volumes/yigitkonur` altında mount et, agresif tuning ve `hard` mount'larla. bu profil server'ın asla yokolmadığını varsayıyor.

-

/etc/auto_nfs

```bash
/Volumes/yigitkonur -vers=3,tcp,rw,hard,intr,noresvport,nfc,locallocks,nonegnamecache,
```

/etc/auto_master

sona ekle (yoksa `auto_nfs` es geçiliyor):

```
/-              auto_nfs
```

uygula:

```bash
sudo automount -cv
```

/etc/nfs.conf (client)

```ini
nfs.client.access_for_getattr = 1
nfs.client.nfsiod_thread_max = 32
nfs.client.allow_async = 1
nfs.client.access_cache_timeout = 60
nfs.client.statfs_rate_limit = 10
nfs.client.tcp_sockbuf = 16777216
nfs.client.readlink_nocache = 2
nfs.client.max_async_writes = 128
nfs.client.iosize = 1048576
```

`nfs.client.is_mobile = 0` **ekleme**. kablolu makinede bile `auto` doğru default — burada kapalıya zorlamanın sana bir kazancı yok ve reçeteyi bir gün laptop'a taşırsan felaket oluyor.

önemli mount option'lar

-

en büyük kazanç veren knob'lar (workstation)

-

client — laptop profili (gerçekten çalıştırdığım)

workstation reçetesinden dört şey değişiyor. dördü de önemli.

1. mount path: `~/mnt/mini`, `/Volumes/bir-şey` değil.

laptop'ta tek başına en etkili değişiklik bu. `/Volumes`'i Finder, Spotlight, Raycast, Dock, LaunchServices, Time Machine ve kurduğun her backup tool'u sürekli tarıyor. `/Volumes` altında takılan bir mount bu takılmayı hepsine yayıyor — Raycast crash ve Finder beachball buradan geliyor. `~/mnt/` altındaki takılı mount sadece oraya elle giren process'leri etkiliyor.

2. mount option'ları: `hard,intr`'ı koru, kısa `timeo` + `retrans` ekle, `nobrowse` ekle, `deadtimeout`'u at.

```bash
-vers=3,tcp,rw,hard,intr,noresvport,nfc,locallocks,nonegnamecache,rsize=1048576,wsize=
```

`hard,intr` data integrity'yi koruyor ve takılan interactive op'ları ctrl+c ile kırmana izin veriyor. asıl değişiklik kısa `timeo=30,retrans=3` (RPC başına ~90s budget): server ölüyse process'ler mount'tan saniyeler içinde sekip çıkıyor, `deadtimeout=600`'ün sattığı 10 dakika

yerine. `nobrowse` ise Finder'ın mount'u listelemesini tamamen engelliyor — her ihtimale karşı, bir gün yine de `/Volumes` altına koyarsın diye.

3. `/etc/nfs.conf` (client) — workstation bloğunun aynısı, ama `nfs.client.is_mobile`'ı kasıtlı olarak hiç ellemiyorsun.

macOS default'u (`auto`) "laptop'ta yanıtsız network volume'ü otomatik unmount et" davranışını açıyor. `0`'a çekmek o güvenlik ağını *kapatıyor* — disconnect'te yaşanan beachball'ın kökü bu ve "NFS on Mac" bloglarının çoğu bu yüzden LAN'da hızlı, LAN dışında kullanılmaz bir makine üretiyor. default'ta bırak. satırı dosyaya hiç koyma.

4. reconnect LaunchDaemon yok. asla.

server'ı 30–60s'de bir pingleyen `StartInterval` daemon, Console.app'ini sonsuza kadar "attaching network" event'leriyle dolduran şey — evde ya da kafede fark etmiyor. laptop'ta asla doğru şekil değil. iki alternatif:

on-demand automount — `~/mnt/mini`'ye erişince mount tetikleniyor, idle kaldıkça unmount oluyor:

`/etc/auto_mini`:

```
/Users/yigitkonur/mnt/mini -fstype=nfs,vers=3,tcp,rw,hard,intr,noresvport,nfc,locallocc
```

`/etc/auto_master` (sona ekle):

```
/-            auto_mini
```

uygula:

```bash
sudo automount -cv
```

ya da elle — share'i gerçekten istediğinde çalıştırdığın küçük bir `mount-mini`:

-

```bash
# ~/bin/mount-mini
#!/bin/bash
set -e
MOUNT="$HOME/mnt/mini"
mkdir -p "$MOUNT"
sudo mount -t nfs -o vers=3,tcp,rw,hard,intr,noresvport,nfc,locallocks,nonegnamecache,
  192.168.1.200:/Users/yigitkonur "$MOUNT"
```

istediğin zaman çalıştırıyorsun. senin LAN'ınla alakası olmayan kafe Wi-Fi'ındayken çalıştırmıyorsun.

kapatabileceğin opsiyonel macOS overhead'leri

```bash
# network share'lerde .DS_Store oluşturmayı durdur
defaults write com.apple.desktopservices DSDontWriteNetworkStores -bool TRUE
# NFS mount'unda Spotlight indexing'i kapat
sudo mdutil -i off ~/mnt/mini
```

Gatekeeper'ın quarantine'ini "perf tweak" diye kapatma. bazı reçeteler `defaults write com.apple.LaunchServices LSQuarantine -bool NO` öneriyor. o flag NFS knob'u değil — *tüm* indirilen dosyalar için, her yerde quarantine prompt'unu kapatıyor. perf kazancı gürültü seviyesinde; security etkisi değil. pas geç.

security: bunu havada bırakma

NFSv3 + `sec=sys` uid/gid ile authenticate ediyor. encryption yok, signing yok, Kerberos yok. dürüst threat model: LAN'ında `501` uid'ini spoof edebilen her makine (kontrolündeki her Mac'te trivial) export ettiğin home dizinindeki her dosyayı okuyup yazabilir.

sadece senin cihazlarının olduğu kablolu ev network'ünde sorun yok. guest Wi-Fi, aynı subnet'te IoT cihazları, ev arkadaşı ya da coworking'e benzeyen herhangi bir yer varsa durum *başka*. share'i kilitsiz bir git remote gibi düşün — kimse aynı telin üstünde olmadığı için işe yarıyor.

reboot-safe hâle getirme

server tarafı (Mac Mini) — sürekli açık, kablolu, hareket etmiyor

```bash
sudo nfsd enable
```

opsiyonel: daemon çakarsa geri gelsin diye küçük bir watchdog.

```bash
# /usr/local/bin/nfsd-watchdog.sh
#!/bin/bash
if ! pgrep -x nfsd > /dev/null 2>&1; then
    nfsd enable && nfsd start
fi
if ! showmount -e localhost 2>/dev/null | grep -q "/Users/yigitkonur"; then
    nfsd update
fi
```

root crontab:

```bash
* * * * * /usr/local/bin/nfsd-watchdog.sh >> /tmp/nfsd-watchdog.log 2>&1
@reboot sleep 10 && /usr/local/bin/nfsd-watchdog.sh >> /tmp/nfsd-watchdog.log 2>&1
```

client tarafı (MacBook) — mobil, dolaşıyor

hiçbir şey. LaunchDaemon yok, cron yok, polling yok. yukarıdaki automount direct map "erişince mount et, idle kalınca unmount et" işini retry loop'u olmadan hallediyor. `cd ~/mnt/mini` yaparken LAN orada değilse, Raycast ve Finder'ı beraberinde götüren 10 dakikalık hang yerine saniyeler içinde mount hatası alıyorsun.

asıl ders: NFS mount üzerinden rsync yapma

-

önce şaşırdım ama küçük dosyalarda matematik affetmiyor:

```
LOOKUP → CREATE → WRITE → COMMIT = 4 RPC × 4ms RTT ≈ dosya başına minimum 16ms
```

60.000 dosya için asıl iş başlamadan önce sadece protokol overhead'ine dakikalar ödüyorsun. toplu transferde stream'le:

```bash
tar cf - --exclude='.git' --exclude='.DS_Store' -C /local/project . \
  | ssh mini "tar xf - -C ~/remote/project/"
```

bende: 60.000 dosya 1 dk 57 sn. rsync-over-nfs yanından bile geçmedi.

helper:

```bash
# /usr/local/bin/nfs-sync.sh
#!/bin/bash
# usage: nfs-sync.sh <local-dir> <remote-relative-dir>
LOCAL_DIR="${1:?Usage: nfs-sync.sh <local-dir> <remote-dir>}"
REMOTE_DIR="${2:?Usage: nfs-sync.sh <local-dir> <remote-dir>}"
FILE_COUNT=$(find "$LOCAL_DIR" -not -path '*/.git/*' -not -name '.DS_Store' | wc -l |
echo "syncing $FILE_COUNT files: $LOCAL_DIR → mini:~/$REMOTE_DIR"
ssh mini "mkdir -p ~/$REMOTE_DIR"
tar cf - --exclude='.git' --exclude='.DS_Store' -C "$LOCAL_DIR" . \
  | ssh mini "tar xf - -C ~/$REMOTE_DIR/"
```

benchmark özeti

sağlıklı LAN, tuning pass'lerinde aynı test suite:

-

sağlıklı LAN'da en büyük katkıyı yapanlar:

- `nfs.client.access_for_getattr = 1`
- `actimeo=10` + `rdirplus` (metadata verimliliği)
- zaman zaman gelen büyük dosya için daha büyük `rsize` / `wsize`

bu tabloda *olmayan* sayı, LAN gittiğinde laptop'un kullanılabilir kalıp kalmayacağını belirliyor: "server erişilemezken mount'ta `ls` ". laptop'a naif workstation ayarlarıyla (`is_mobile=0`, `hard,intr,deadtimeout=600`, `/Volumes` altında mount) gittiğinde on dakikalık beachball. yukarıdaki laptop profiliyle birkaç saniye ENOENT ve yoluna devam ediyorsun.

macOS'ta NFSv3 vs NFSv4: v4'e bulaşma

- macOS NFSv4.0'ı destekliyor, 4.1'i değil
-

- `vers=4.1` zaten genelde v3'e düşüyor
- `vers=4`, bazı Sonoma / Sequoia sürümlerinde regresyonlu
- tuned v3 dev workload'ları için zaten yeterince iyi

NFS vs SMB vs alternatifler

kullandığım perf test script'i

`/tmp/nfs-perftest.sh` içine koy (`NFS_TARGET`'ı kendi mount'unun içine ayarla):

-

```bash
#!/bin/bash
set -e
NFS_TARGET="$HOME/mnt/mini/dev/some-project"
TEST_DIR="$NFS_TARGET/.nfs-perftest-$$"
t() { perl -MTime::HiRes -e 'print Time::HiRes::time()'; }
log() { printf "  %-35s %7.2fs  %s\n" "$1" "$2" "$3"; }
cleanup() { rm -rf "$TEST_DIR" 2>/dev/null; }
trap cleanup EXIT
mkdir -p "$TEST_DIR"
echo; echo "  NFS perf test"; echo "  $(printf '%.0s-' {1..45})"
t0=$(t)
for i in $(seq 1 100); do dd if=/dev/urandom bs=$((1024+RANDOM%9216)) count=1 of="$TES
d=$(echo "$(t) - $t0" | bc); log "create 100 files (1-10KB)" "$d" "$(echo "100/$d"|bc)
t0=$(t)
for f in "$TEST_DIR"/f*; do cat "$f">/dev/null; done
d=$(echo "$(t) - $t0" | bc); log "read 100 files" "$d" "$(echo "100/$d"|bc) files/s"
t0=$(t)
for f in "$TEST_DIR"/f*; do stat -f "%z" "$f">/dev/null; done
d=$(echo "$(t) - $t0" | bc); log "stat 100 files" "$d" "$(echo "100/$d"|bc) ops/s"
t0=$(t)
for i in $(seq 1 50); do echo "mod $i $(date +%s%N)">"$TEST_DIR/f$i"; done; sync
d=$(echo "$(t) - $t0" | bc); log "overwrite 50 files" "$d" "$(echo "50/$d"|bc) files/s
t0=$(t); ls -la "$TEST_DIR">/dev/null
d=$(echo "$(t) - $t0" | bc); log "ls -la (100 files)" "$d"
t0=$(t); dd if=/dev/zero of="$TEST_DIR/big" bs=1048576 count=10 2>/dev/null; sync
d=$(echo "$(t) - $t0" | bc); log "write 10MB sequential" "$d" "$(echo "10/$d"|bc) MB/s
echo "  $(printf '%.0s-' {1..45})"; echo
```

appendix: Craft.do'ya API üzerinden post atıyorsan (markdown'ı otomatik güvene al)

üç-tırnak code fence'li "github-flavored" versiyonun varsa, POST atmadan önce fence'li kod bloklarını indent'li kod bloklarına çevirerek Craft.do-safe markdown'a dönüştürebilirsin. jq filter'ı:

-

```
def craft_safe_md:
  gsub("\r\n"; "\n")
  | (split("\n")) as $lines
  | reduce $lines[] as $line (
      {out: [], in_code: false};
      if ($line | test("^[[:space:]]*```")) then
        .in_code = (.in_code | not)
        | .out += [""]
      else
        if .in_code then
          .out += ["    " + $line]
        else
          .out += [$line]
        end
      end
    )
  | .out
  | join("\n")
  | rtrimstr("\n") + "\n";
```

yollayacağın markdown string'inde çalıştır (tek geçişli jq; shell variable'ına sokup round-trip yaptırma).

tldr

- macOS'ta küçük dosyalarda SMB yavaş hissettiriyorsa NFSv3 dene — server tuning herkes için aynı, client tuning'de yollar şekle göre ayrılıyor
- server tarafı: `require_resv_port=0`, `nfsd_threads=16`, `async=1`, sadece TCP
- workstation client: `/Volumes/*`, `hard,intr,actimeo=10,rdirplus`, automount direct map
- laptop client: `~/mnt/*`, aynı option'lar artı `timeo=30,retrans=3,nobrowse`, `nfs.client.is_mobile` **'ı default'ta bırak**, reconnect LaunchDaemon yok
- en büyük perf knob'u `nfs.client.access_for_getattr = 1`; laptop'ta en tehlikeli ayar `nfs.client.is_mobile = 0`
- Gatekeeper'ın `LSQuarantine` 'ini "perf tweak" diye kapatma
- NFSv3 + `sec=sys` sadece uid/gid auth'u; share'i güvenilir subnet'te tut
- toplu kopya için mount üzerinden rsync değil, `tar | ssh` kullan

SOURCES

- Apple nfsd man page (https://developer.apple.com/library/archive/documentation/System/Conceptual/ManPages_iPhoneOS/man8/nfsd.8.html)
- Apple exports man page (https://developer.apple.com/library/archive/documentation/System/Conceptual/ManPages_iPhoneOS/man5/exports.5.html)
- Apple automount man page (https://developer.apple.com/library/archive/documentation/System/Conceptual/ManPages_iPhoneOS/man8/automount.8.html)
- Apple mount_nfs man page (https://developer.apple.com/library/archive/documentation/System/Conceptual/ManPages_iPhoneOS/man8/mount_nfs.8.html)

-