Peering & the ~50-node wall

Why the network wasn't growing past ~50 mining nodes — root cause and the fix.

Bloch-SIS-PoW · ownerless protocol · Postern Labs engineering note · 2026-07-15 · **UNAUDITED**

> **Honesty baseline.** The chain runs the relaxed **k=4 regime — proof-of-work is trivially forgeable and no security is claimed**; it is **unaudited** and 51%-attackable with few nodes. The coin (BLCH) is **not a security and not an asset** — no sale, no listing, no price, **no value claim**. This note is engineering, not an investment statement. The protocol is **ownerless**: nothing below is a mandate — node operators adopt changes or don't, the Bitcoin way.

1 · Symptom

New nodes — including nodes started to *mine* — could not join the live network. The public bootstrap node sat at **exactly 50 peers** and stayed there: incoming connections from newcomers were accepted and then **immediately dropped**. Freshly-started miners, unable to sync, produced blocks that went nowhere. In practice the network could not grow past its existing peer set, and added mining capacity contributed **zero** real hashrate to the canonical chain.

2 · Root causes

2.1 — A hardcoded 50-peer cap (the headline)

The node capped established *inbound* connections at a compile-time constant, `max_peers = 50`. Once a well-connected bootstrap/hub node fills those 50 slots (e.g. with existing miners), the swarm layer **refuses every further inbound connection** — so the 51st node onward literally cannot attach to that hub. On a small network with one obvious bootstrap, that is a hard ceiling on participation.

2.2 — Fresh miners fork from genesis (so "mining nodes" contributed nothing)

A node started with `--mine` on an empty data directory begins mining from the genesis block *immediately*. On the forgeable k=4 regime that is trivial, so within minutes it builds its own isolated chain — **before it has synced the real one**. Worse, once it holds a deep private fork, the reorg-depth protection (added in the security pass) can **prevent it from ever adopting the real chain**. Net effect: a "mining node" that never joins, burning CPU on a fork nobody sees.

2.3 — Cold-start friction (by design, but unassisted)

The shipped seed list is deliberately empty (`DEFAULT_SEEDS = []`) — the ownerless ethos: no privileged, foundation-run bootstrap. The honest cost is that a new node **sits alone until an operator hands it a reachable peer**, and there was no DNS-seed convenience to discover several peers at once.

2.4 — Cloud networking gotchas (deployment, not protocol)

On a single cloud provider, same-account machines often **cannot reach each other's public IP** (hairpin NAT), and a node that listens on IPv4 only (`--listen /ip4/0.0.0.0/…`) **refuses private-IPv6 dials**. Together these islanded the demo followers so they never actually joined the miner's chain — masking the real height and compounding the impression that "nodes don't connect."

3 · What was done to unblock it

A focused, **non-consensus** change (branch `feature/peering-ux`) — no touch to proof-of-work, block validation, serialization, or GhostDAG. Three operator-facing improvements:

Change	What it does	Fixes
`--max-peers <N>`	The 50-peer cap is now an operator knob. A provisioned hub can run `--max-peers 500` and accept newcomers; a leaf node keeps it modest.	§2.1
Fresh-miner fork guard	A node configured to *join* (has `--peer` or `--dns-seed`) but with 0 peers now **pauses mining until it connects**, instead of forking from genesis. Intentional solo/island nodes (no peer configured) are unaffected.	§2.2
`--dns-seed <host>`	Opt-in, Bitcoin-style cold-start: the node resolves a hostname's A/AAAA records and dials them as peers. `DEFAULT_SEEDS` stays empty — **non-privileged**: point at, or run, any seed you trust.	§2.3

Deployment note (honest): the code is implemented and compiles; raising `--max-peers` on the public demo nodes is an *operator* action, and the change ships as a candidate any node operator may adopt — the base is ownerless. The cloud-networking issues in §2.4 are resolved by running a node on a normal host with a reachable address, or by peering over the correct interface.

4 · How to set up a node correctly, from now on

Build

```
git clone <gitlab.com/blochsispow-group/BlochSISPoW-project> bloch && cd bloch
cargo build --release            # needs rust + clang/cmake/pkg-config
```

Join & sync FIRST (as a relay)

```
./target/release/bloch --testnet --data-dir ./bloch-data \
  --listen /ip4/0.0.0.0/tcp/16110 \
  --peer /dns4/<a-reachable-node>/tcp/16110      # or --dns-seed <host>
```

Watch it catch up before doing anything else:

```
./target/release/bloch-cli getpeerinfo     # peer_count > 0 = connected
./target/release/bloch-cli getblockcount   # wait until it reaches the tip
```

THEN mine (only once synced)

```
# same command + --mine. The fork guard now enforces this if you
# configured a peer/seed: it won't mine from genesis while unconnected.
./target/release/bloch --testnet --mine --data-dir ./bloch-data \
  --listen /ip4/0.0.0.0/tcp/16110 --peer /dns4/<node>/tcp/16110
```

Running a bootstrap / hub? Raise the cap

```
./target/release/bloch --testnet --max-peers 500 \
  --listen /ip4/0.0.0.0/tcp/16110
```

Rules of thumb.

- **Always `--testnet`** — historical flag name; it selects the live mainnet-beta chain.
- **Sync before you mine.** A miner on an empty data-dir that hasn't synced will fork from genesis and help no one.
- **Give your node a reachable peer.** With `DEFAULT_SEEDS` empty by design, a node sits alone until you do. After first contact, PEX finds the rest and your node becomes a seed for the next person.
- **Hubs raise `--max-peers`; leaves keep it modest.**
- **Prefer a normal host** (VPS/bare-metal) with a routable address over same-account cloud machines that can't reach each other.

Bloch-SIS-PoW is an ownerless, post-quantum, pure-Proof-of-Work research chain. Everything here is **unaudited**; the relaxed k=4 regime is trivially forgeable; the coin carries **no value claim**. Postern Labs builds on the base as one operator among any number — it does not own, schedule, or speak for the protocol. This note describes a non-consensus operator convenience, nothing more.