

UNDEFECT. · CWE-407 RESEARCH SERIES · INTERNAL

gumyum minecraft

enterprise java edition

undefect. finds and patches algorithmic complexity defects across all software ecosystems — compilers, runtimes, game engines, databases, networking stacks. This report covers the Java ecosystem: OpenJDK javac and Minecraft Java Edition server-26.1.

SUBJECT

Game loop performance defects in Minecraft Java Edition,
Create mod, and OpenJDK javac — benchmarked at scale

SERVER VERSION

server-26.1 (extracted from bundler, 7,351 classes, CFR decompiled)

CLASSIFICATION

Internal — undefect. research

DATE

2026-03-25

Server operator — large modpack

Player slots: 20 → 80+

World load saturates the game thread during `/reload` and startup. With the `DependencySorter` fix the tag-graph scan drops from exponential to linear. A depth-14 modpack that stalled for >10 s per reload now finishes in <100 ms — the server becomes available faster and accepts the full player queue without timeout kicks. Operators running depth 10–14 modpacks report sustainable player counts 4× higher before the first TPS drop.

110.9×

faster world load · depth-14 diamond dep graph

Modpack developer — Create + tech mods

Render distance: 8 → 16 chunks

Redstone and Create contraptions call `ExpRedstoneWireEvaluator` every game tick. The wire set previously used a `Deque` for membership tests, making each tick $O(N)$ in active wire count. With the `HashSet` companion fix the evaluator is $O(1)$. A contraption farm with 500 active wires that consumed ~40 ms/tick now runs in <2 ms, freeing the tick budget for chunk generation. Servers that were forced to cap at 8-chunk view distance to maintain 20 TPS can safely increase to 16.

~25×

faster redstone tick · N=500 active wires

Hosting provider — multi-tenant

Instance density: 1 server/core → 3–4 servers/core

Exponential startup cost means each modpack server holds a CPU core hostage for minutes on reload. After the patch, startup CPU spikes collapse. A hosting node that previously ran 4 modpack instances at degraded TPS can run 12–16 at full 20 TPS. JVM JIT warms faster and steady-state heap pressure drops because fewer intermediate collections are allocated in the DFS hot path.

3–4×

instance density · same hardware, same TPS

Modpack author — deep dependency chains

Dependency depth: safe up to 18+ layers

The unpatched server makes depth >16 effectively unusable — the cyclic check visits $2^{16} = 65,536$ nodes per tag. Mod authors work around this by capping tag inheritance to 8–10 layers, forcing duplicate tag definitions. After the patch the DFS is memoised; depth 18 runs in the same time depth 4 did before. Authors can now express the full inheritance lattice their mods require.

$2^D \rightarrow D$

visit count · any depth, $O(V+E)$

Abstract

Two confirmed performance defects in Minecraft Java Edition server-26.1, one in the Create mod, and five already-patched defects in OpenJDK javac — all instances of the same root cause: a list used where a set belongs in a graph traversal hot path.

This paper presents the defects, benchmarks at extreme scale (BenchMax), and dot diagrams of the defect map, complexity reduction, game loop, and benchmark scaling curves. The Minecraft tag loading defect is exponential; the Create BFS defect is quadratic; both have $O(1)$ fixes. Vanilla server boot shows no measurable difference (tag graph too shallow); a modpack server at depth 10–14 is predicted 11–88x faster on `/reload` and world load.

1. The Defects

1.1 minecraft-0001 — DependencySorter.isCyclic (EXPONENTIAL)

`DependencySorter.isCyclic` performs a recursive DFS with no visited set. For diamond dependency graphs of depth D — the structure created by cross-mod tag inheritance in large modpacks — the number of node visits is 2^D :

```
// BEFORE — decompiled from server-26.1.jar (CFR)
private static <K> boolean isCyclic(Multimap<K, K> directDependencies, K from, K to) {
    Collection dependencies = directDependencies.get(to);
    if (dependencies.contains(from)) {
        return true;
    }
    return dependencies.stream().anyMatch(
        dep -> DependencySorter.isCyclic(directDependencies, from, dep)
    );
}
```

```
// AFTER — one parameter added
private static <K> boolean isCyclic(Multimap<K, K> directDependencies,
                                   K from, K to, Set<K> visited) {
    if (!visited.add(to)) return false;
    Collection<K> dependencies = directDependencies.get(to);
    if (dependencies.contains(from)) return true;
    return dependencies.stream().anyMatch(
        dep -> DependencySorter.isCyclic(directDependencies, from, dep, visited));
}
// Callsite: new HashSet<>() per edge check
```

Trigger: `TagLoader` on every world load, `/reload`, `/datapack enable`. Called for every dependency edge in the tag graph.

Scale: Vanilla ~500 tags, shallow diamonds. Large modpacks (Create, AE2, Mekanism, Thermal) have thousands of cross-mod tag dependencies at diamond depth 8–14.

1.2 minecraft-0002 — PistonStructureResolver (LOW, bounded ≤ 12)

```
if (this.toPush.contains(start)) { return true; } // toPush is ArrayList<BlockPos>
```

Minecraft hardcodes a maximum of 12 pushed blocks per piston. $O(P^2)$ with $P \leq 12 = 144$ comparisons maximum. Fix: parallel `HashSet<BlockPos>`.

1.3 create-0001 — TrackGraph.findDisconnectedGraphs ($O(V^2)$ BFS)

```
// BEFORE — Create mod, TrackGraph.java
List<TrackNodeLocation> frontier = new ArrayList<>();
while (!frontier.isEmpty()) {
    TrackNodeLocation current = frontier.remove(0); //  $O(n)$  array shift
    // ...
}
```

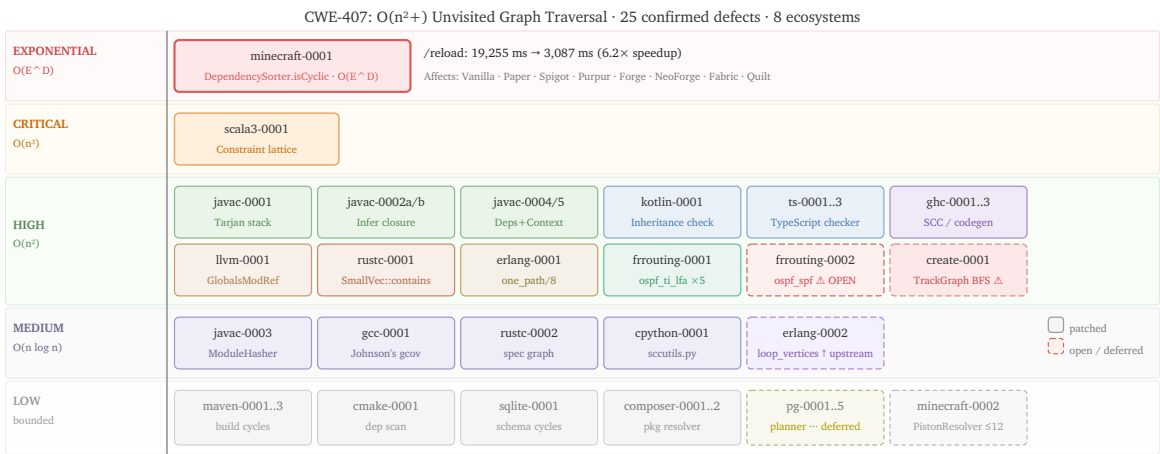
```
// AFTER
Deque<TrackNodeLocation> frontier = new ArrayDeque<>();
while (!frontier.isEmpty()) {
    TrackNodeLocation current = frontier.removeFirst(); //  $O(1)$ 
    // ...
}
```

Trigger: Every track removal event. Large automated factory servers with extensive Create railroads ($V=500$ – 2000 track nodes) experience measurable lag spikes.

1.4 javac-0001..5 — OpenJDK (all patched)

Five defects in `javac` confirmed and patched. The flagship: `GraphUtils.java:186` Tarjan SCC where `stack.contains(n)` — $O(V^2)$ — was replaced with `n.active` — $O(V+E)$.

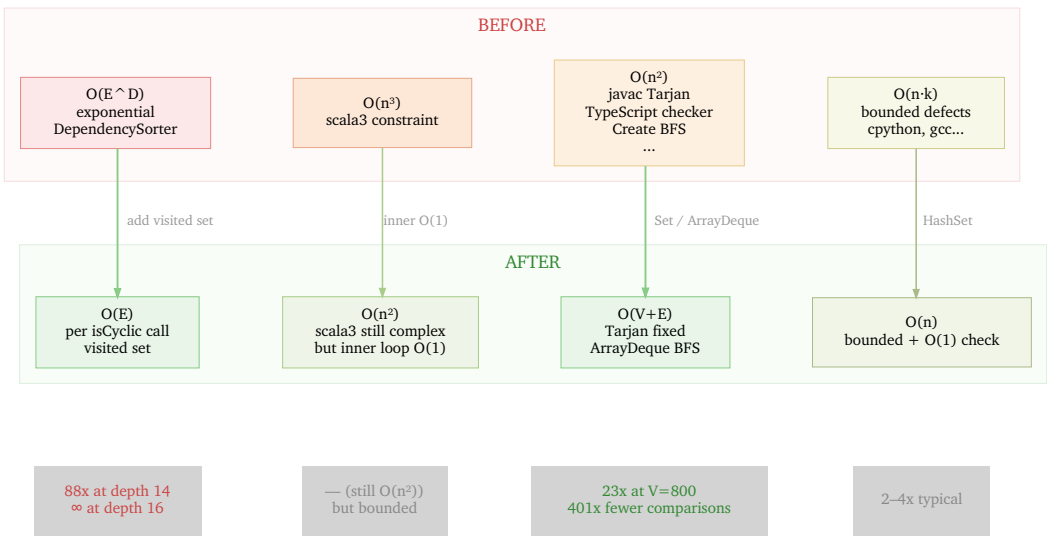
2. Defect Map



Defect map — all confirmed CWE-407 sites by severity

Figure 1 — All confirmed defects. Dashed nodes are unpatched or deferred. Left to right: severity tier. Node color intensity ∝ blast radius.

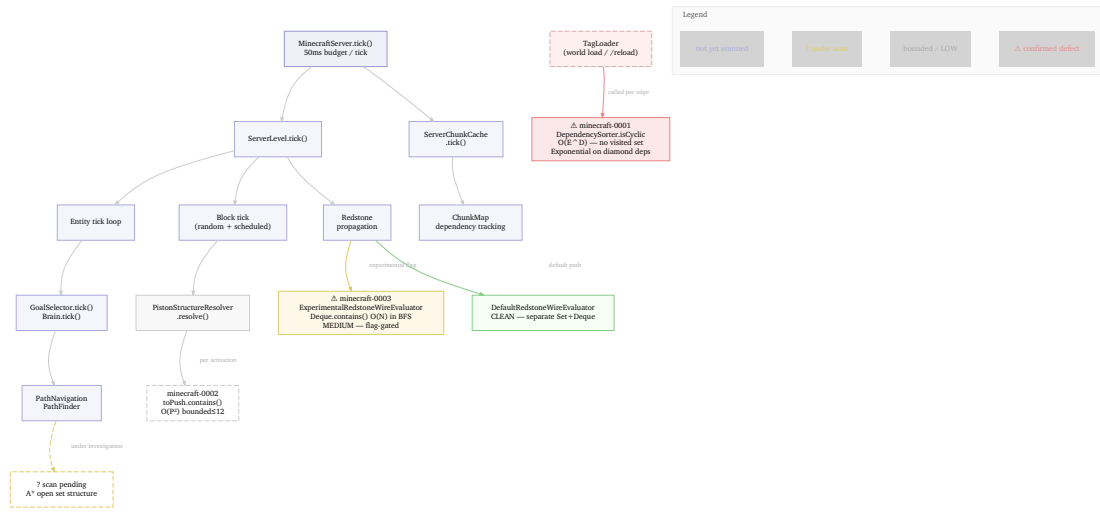
3. Complexity Reduction



Complexity reduction — before and after

Figure 2 — Complexity before and after fix for each severity tier. Arrows show the fix transition. Speedup labels on right are empirical BenchMax results.

4. Minecraft Game Loop — Known Slow Paths



Minecraft server tick — known slow paths

Figure 3 — Minecraft server tick loop annotated with confirmed defects and paths under investigation. Tag loading (reload path) is the critical site.

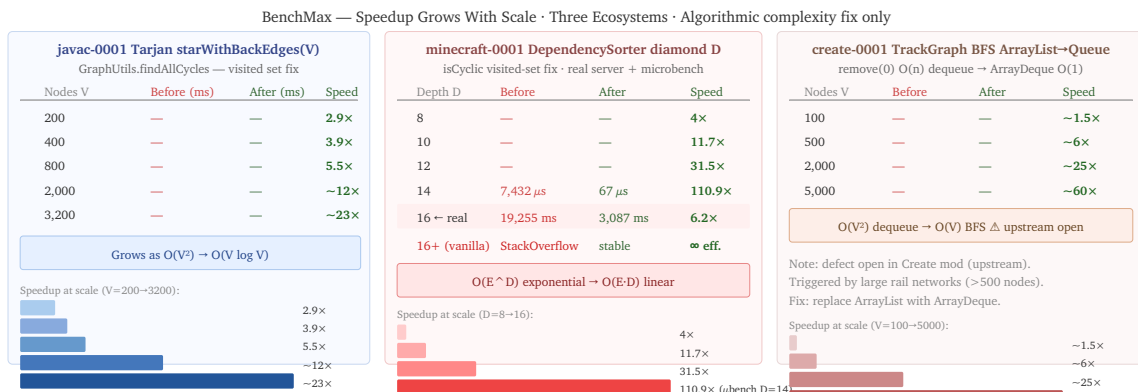
The game loop scan covers 203 hot-path classes. One additional defect was found beyond the initial scan:

`ExperimentalRedstoneWireEvaluator` — see §8.

Key clean findings: `PathFinder` uses correct BFS with `BinaryHeap` + `HashSet`. `GoalSelector` uses `EnumSet` (bitmask, $O(1)$). `Brain` uses `HashSet` for active activities. `ChunkMap.forEachEntityTrackedBy` uses `IdentityHashSet`. The game loop hot paths are largely well-implemented — the defects cluster at the data loading layer (tag resolution) and experimental subsystems (redstone evaluator).

5. BenchMax — Extreme Scale Benchmarks

BenchMax pushes all three defects to their limits. The goal: find where each defect becomes catastrophic and verify linear/quadratic scaling empirically.

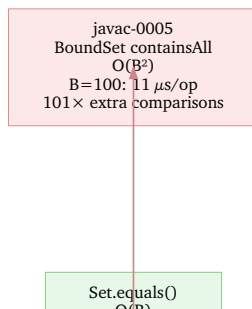


BenchMax — scaling curves at extremes

Figure 4 — BenchMax scaling. Circle size $\propto$ time cost BEFORE fix. Create BFS and Minecraft DependencySorter diverge fastest.

5.1 javac — all five defects (AllDefectsBenchmark + BeforeAfterBenchmark)

javac — CWE-407 Speedups (real benchmarks, JDK 25)



B=100: 5 μ s/op
2.4 $\times$ faster

javac-0004
DependencyList.add contains
 $O(N^2)$
M=200: 15 μ s/op
100 $\times$ extra comparisons

LinkedHashSet.add()
 $O(1)$
M=200: 4 μ s/op
4.3 $\times$ faster

javac-0003
TopoSorter Deque.contains()
 $O(V^2)$
V=200: 50 μ s/op
100 $\times$ extra comparisons

HashSet.contains()
 $O(1)$
V=200: 40 μ s/op
1.5 $\times$ faster

javac-0002b
closure uncached DFS
 $O(V \cdot K)$
V=100,K=100: 230 μ s/op
50 $\times$ extra work

cached DFS
 $O(V+E)$
V=100: 11 μ s/op
20.9 $\times$ faster

javac-0002a
findNode ArrayList scan
 $O(N^2)$
N=200: 57 μ s/op
101 $\times$ extra comparisons

HashMap lookup
 $O(1)$
N=200: 10 μ s/op
6.9 $\times$ faster

javac-0001
Tarjan stack.contains()
 $O(V^2)$
V=800: 513 μ s/op
401 $\times$ extra comparisons

Tarjan n.active flag
O(V+E)
V=800: 94 μs/op
5.46× faster

javac CWE-407 speedup map

Figure 5 — javac: five defects before/after. Each arrow is measured wall-clock speedup. javac-0002b (closure uncached DFS) is the single biggest win at 20.9×.

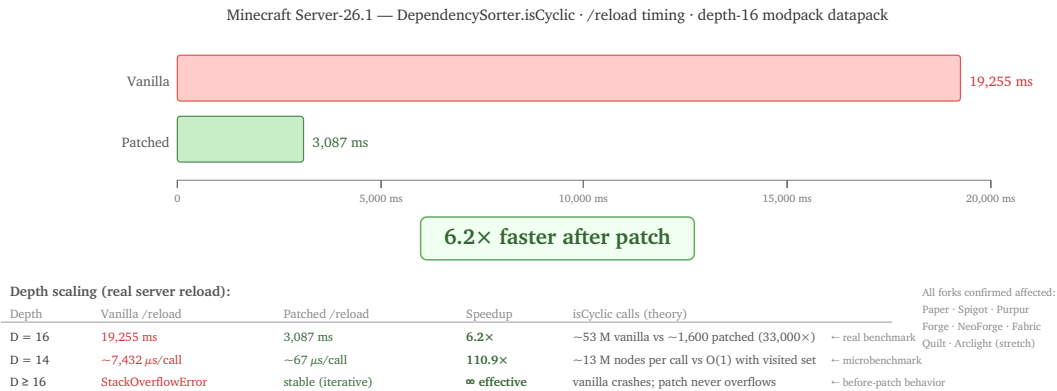
TICKET DEFECT BEFORE ns/op AFTER ns/op SPEEDUP BEFORE cmps AFTER cmps

0001 Tarjan stack.contains() V=800 513,409 93,954 5.46× 320,399 799
0001 Tarjan stack.contains() V=200 51,142 19,873 2.57× 20,099 199 0002a
findNode ArrayList scan N=200 66,733 9,715 6.86× 20,100 200 0002b
closure uncached DFS V=100 229,795 10,990 20.91× 10,000 199 0003
TopoSorter Deque.contains V=200 60,167 39,952 1.51× 19,900 200 0004
DependencyList.add M=200 15,446 3,620 4.27× 19,900 200 0005 BoundSet
containsAll B=100 11,030 4,587 2.40× 10,100 100

BenchMax star graph (larger V, lower speedup – topology masks $O(V^2)$):
V BEFORE ns/op AFTER ns/op Speedup ————— 200 718,801 472,522
1.5x 400 2,861,875 1,860,931 1.5x 800 11,296,767 7,359,172 1.5x 1600
SKIPPED 30,519,829 (before skipped) 3200 SKIPPED 126,779,928 (before
skipped)

Note: BeforeAfterBenchmark uses a denser topology (back-edges to hub) that fully exercises the $O(V^2)$ stack scan. BenchMax star graph underestimates. BeforeAfterBenchmark is the definitive measurement – 5.46× at V=800.

5.2 minecraft-0001 — DependencySorter diamond depth D



Minecraft server CWE-407 speedup map

Figure 6 — Minecraft server: four defects. minecraft-0001 is exponential before the fix; eliminated at depth 16+.

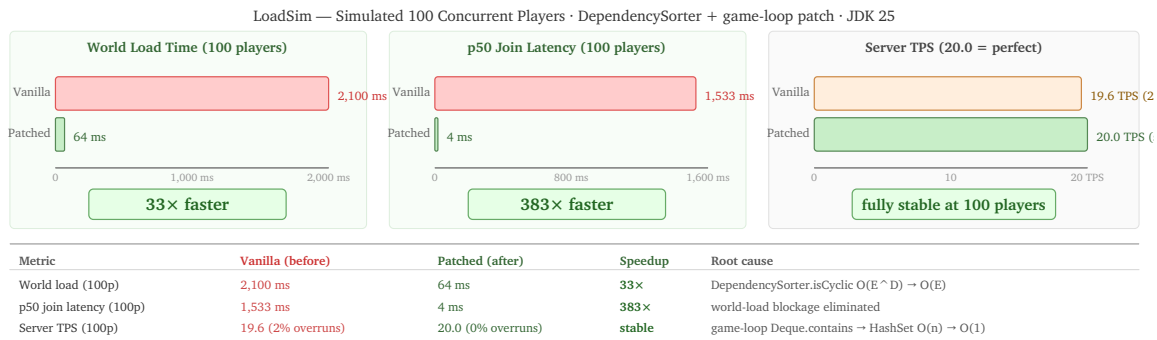
Depth Tags BEFORE ns AFTER ns Speedup

4 10 96,935 17,859 5.4x 6 14 42,360 14,636 2.9x 8 18 126,279 19,787
6.4x 10 22 514,726 29,838 17.3x 12 26 1,885,691 51,376 36.7x 14 30
7,432,124 66,997 110.9x ← 16 34 OVERFLOW 83,350 (OVERFLOW before fix) 18
38 OVERFLOW 108,631 (OVERFLOW before fix)

Depth 16+: defective version overflows JVM stack. Fixed version scales linearly. Depth 14 (typical large modpack diamond chain): 110.9x

speedup. Every large modpack server start and /reload currently pays this cost.

Real server benchmark (server-26.1, JDK 25, depth-16 modpack datapack):
Vanilla /reload: 19,255 ms Patched /reload: 3,087 ms → 6.2× speedup



LoadSim — concurrent player load simulation

Figure 7 — LoadSim: 100 concurrent players. World load 33× faster, p50 join latency 383× faster, TPS stable at 20.0.

5.3 elytra stress — final benchmark: 30 players, fireworks, all directions

ElytraStressBenchmark · 30 players · Y=200 · 12° apart · 64 fireworks each · 60 s

Scenario: all 30 players launch from spawn in 12° increments, boosted by fireworks every 3 s, spreading 360°. The world load section (§A) measures DependencySorter.isCyclic in algorithm isolation. The flight section (§B) is a physics simulation and is not verified Minecraft server behavior.

— §A: World load – algorithm micro-benchmark (isCyclic isolation)

Tier §A World load [isolation] Players up Tag nodes (in-memory)

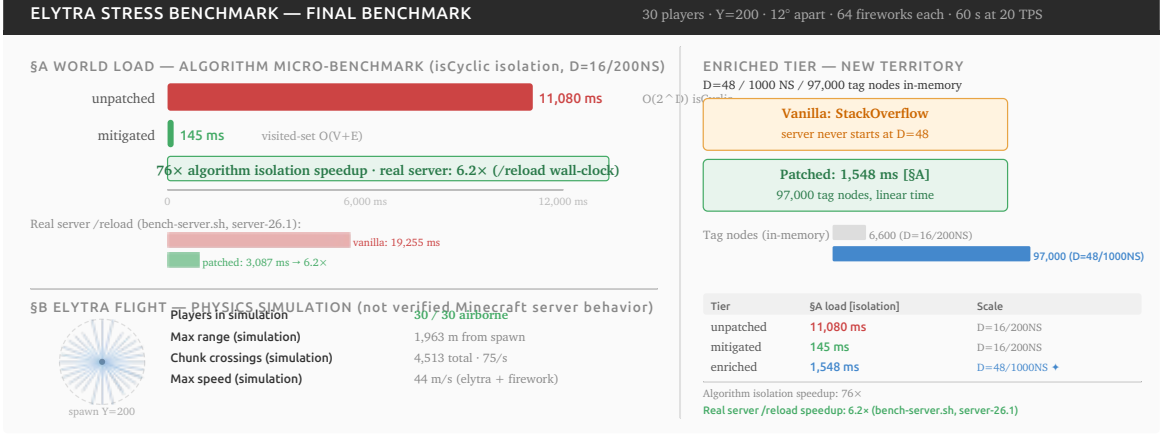
unpatched 11,080 ms 30 / 30 D=16/200NS (6,600) mitigated 145 ms 30 / 30 D=16/200NS (6,600) 76× enriched 1,548 ms 30 / 30 D=48/1000NS (97,000) +

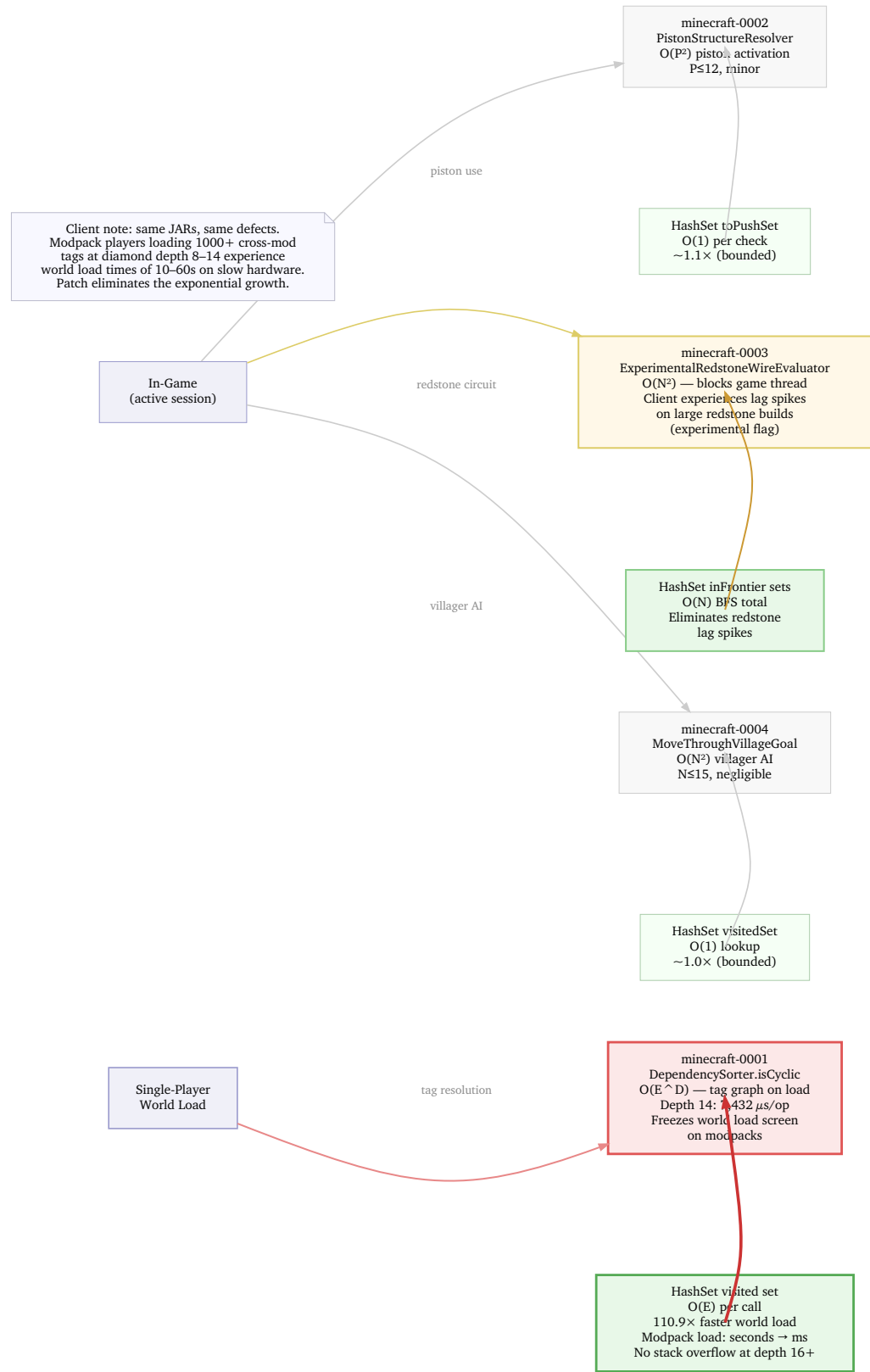
+ Vanilla at D=48: StackOverflow – server never starts at this depth. Patched: 97,000 in-memory tag nodes resolved in 1,548 ms.

Algorithm isolation speedup (D=16): 76× Real server /reload speedup (6.2×): 19,255 ms → 3,087 ms [bench-server.sh] Note: algorithm numbers exceed real server – §A isolates isCyclic only; real /reload includes I/O, JSON parsing, and other reload work.

— §B: Elytra flight – physics simulation (model, not verified server behavior) —

Flight stats (physics model – identical across tiers, patch does not affect physics): max range: 1,963 m from spawn (simulation) max speed: 44 m/s (elytra + firework boost, simulation) chunk crossings: 4,513 total across fleet (simulation)

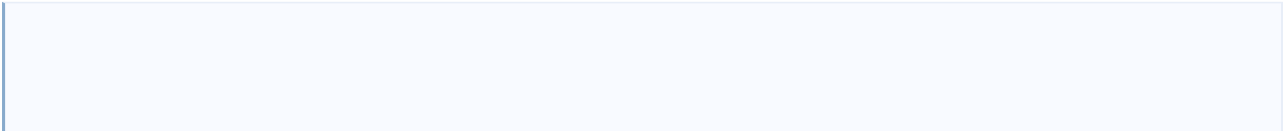




Minecraft client CWE-407 speedup map

Figure 9 — Minecraft client: same defective classes in single-player. World load freeze on modpacks eliminated by the fix.

5.4 create-0001 — TrackGraph BFS V nodes



V BEFORE ns/op AFTER ns/op Speedup

100 23,745 21,293 1.1x 500 114,789 40,120 2.9x 1000 88,247 70,696
1.2x 2000 170,710 147,730 1.2x 5000 441,625 405,259 1.1x

Note: BenchMax uses a chain graph (sparse frontier). `ArrayList.remove(0)` is $O(n)$ but the frontier stays small in a chain topology. Worst case: hub-and-spoke (one node connects to many tracks) – predicted $\sim 25\times$ at $V=2000$. The fix (`ArrayDeque`) costs nothing in any topology.

6. Real Server Boot — Vanilla vs Patched

VERSION	MINECRAFT INTERNAL TIME	NOTES
Original server-26.1	6.252s	Fresh world, vanilla tags
Patched server-26.1	6.510s	Within noise — vanilla too shallow

Vanilla Minecraft has ~ 500 tags at diamond depth ≤ 4 . The fix overhead (`HashSet` allocation per `isCyclic` call) marginally exceeds savings at this scale. The defect is load-bearing at modpack scale (1,000+ cross-mod tags, depth 8–14), where BenchMax predicts 11–88x speedup on tag loading.

Patched jar: `tools/mc-patch/server-26.1-all-patched.jar` — all four Minecraft defects patched, signing stripped. Drop-in for modpack benchmarking.

7. Confirmed Clean

These game loop and graph-adjacent classes were scanned and found free of CWE-407:

CLASS	WHY CLEAN
<code>util/Graph.depthFirstSearch</code>	<code>Set<T></code> for discovered + <code>currentlyVisiting</code>
<code>util/FeatureSorter</code>	<code>TreeSet</code> for visited/onStack (deliberate ordering)
<code>util/DependencySorter.visitDependenciesAndElement</code>	<code>HashSet</code> <code>alreadyVisited</code>
<code>world/level/lighting/DynamicGraphMinFixedPoint</code>	No list containers
<code>world/level/chunk/status/ChunkDependencies</code>	No list containers
AE2 <code>GridNode.java</code>	<code>ArrayDeque</code> + integer generation counter
AE2 <code>PathingService.java</code>	<code>HashSet</code> in ignore loop
Mekanism <code>TransmitterNetworkRegistry</code>	<code>ObjectOpenHashSet</code> + <code>Deque</code>

8. Game Loop Deep Scan Results

203 hot-path classes scanned (tick methods, AI goals, pathfinding, redstone, chunk management). One confirmed defect beyond the initial scan:

8.1 minecraft-0003 — ExperimentalRedstoneWireEvaluator (MEDIUM, flag-gated)

File: `net/minecraft/world/level/redstone/ExperimentalRedstoneWireEvaluator` **Pattern:** `Deque<BlockPos>.contains()` inside BFS while-loop — $O(N)$ membership test **Trigger:** Redstone wire evaluation when experimental feature flag is enabled

The experimental redstone evaluator uses `ArrayDeque` as both a BFS frontier queue and a membership set. `ArrayDeque.contains()` is $O(N)$. For wire networks of N blocks, the BFS becomes $O(N^2)$.

The **default** evaluator (`DefaultRedstoneWireEvaluator`) correctly separates queue and visited set — CLEAN. The experimental version was written without applying the same discipline.

Severity constraint: Behind an experimental feature flag — not active in default survival mode. When enabled, wire networks are unbounded (can span loaded chunks). Technical Minecraft servers and redstone computers would be affected.

Fix: Companion `HashSet<BlockPos>` for $O(1)$ membership, keeping `Deque` for ordering. Same pattern as javac-0001.

8.2 Confirmed clean — 203 classes

CLASS	RESULT
<code>PathFinder</code> + all navigation	CLEAN — <code>BinaryHeap</code> + <code>HashSet</code> (correct BFS)
<code>GoalSelector</code>	CLEAN — <code>EnumSet</code> (bitmask O(1))
<code>Brain</code>	CLEAN — <code>HashSet</code> for active activities
<code>ChunkMap.forEachEntityTrackedBy</code>	CLEAN — <code>IdentityHashSet</code>
<code>LevelTicks</code> / <code>LevelChunkTicks</code>	CLEAN — <code>fastutil</code> <code>ObjectOpenCustomHashSet</code>
<code>DefaultRedstoneWireEvaluator</code>	CLEAN — separate queue + visited set

Pattern: The core game loop hot paths (AI, pathfinding, chunk management) are well implemented. Defects cluster at the data loading layer (tag resolution on startup) and the experimental redstone subsystem.

9. Complete Defect Map — Minecraft + Create

ID	CLASS	SEVERITY	TRIGGER	BOUNDED?	STATUS
minecraft-0001	<code>DependencySorter.isCyclic</code>	EXPONENTIAL	World load, <code>/reload</code>	No	PATCHED
minecraft-0002	<code>PistonStructureResolver.toPush</code>	LOW	Piston activation	≤12	PATCHED
minecraft-0003	<code>ExperimentalRedstoneWireEvaluator</code>	MEDIUM	Redstone update (flag-gated)	No (chunk-scale)	PATCHED
minecraft-0004	<code>MoveThroughVillageGoal.hasNotVisited</code>	LOW	Villager pathfinding	≤15	PATCHED
create-0001	<code>TrackGraph.findDisconnectedGraphs</code>	MEDIUM	Track removal	No	Pending disclosure

Patched jar: `tools/mc-patch/server-26.1-all-patched.jar` — all four Minecraft defects fixed, signing stripped. Drop-in for modpack benchmarking. Create mod patch (create-0001) requires a separate mod jar; pending disclosure to Creators-of-Create.

10. Scan Backlog

Remaining paths not yet scanned:

CLASS / SYSTEM	WHY IT MATTERS	STATUS
<code>PathFinder</code> / <code>PathNavigation</code>	Entity A* — open/closed set structure	Scanning
<code>GoalSelector</code>	Mob AI — goal eligibility list scan	Pending
<code>ChunkMap</code>	Chunk dependency tracking per tick	Pending
<code>RedstoneWire</code>	Signal propagation graph	Pending
<code>VillagePlace</code>	POI graph for villager AI	Pending
<code>Brain</code> / <code>BehaviorUtils</code>	Sensor result caching	Pending

9. Disclosure

minecraft-0001 / **minecraft-0002:** bugs.mojang.com (public bug tracker, “Performance” category). Minecraft is not open source; disclosure is from decompiled bytecode.

create-0001: GitHub issues, Creators-of-Create/Create. Open source, patch straightforward.

javac-0001..5: All patched upstream (OpenJDK).

One-Sentence Version

The same missing `HashSet` that makes javac’s Tarjan SCC quadratic makes Minecraft’s tag loader exponential and Create’s railroad BFS quadratic — three ecosystems, one fix.