TuffXPlus Plugin

Code Review & Bug Report

Target Environment: Paper 1.21.x | ViaVersion + ViaBackwards

Report Date	May 25, 2026
Plugin Name	TuffXPlus (main branch)
Reviewed Against	v1.2.0 (latest release tag)
Target Server	Paper 1.21.x
Target Client	TuffClient (Eaglercraft 1.12-based)
Reviewer	Code Review Analysis
Report Version	1.0

1. Executive Summary

This report covers a full code review of the TuffXPlus plugin, comparing the current main branch against the previously released v1.2.0. The review focuses on correctness, thread safety, performance, and Paper API best practices. Several critical bugs and design issues remain unfixed. They are documented here in detail so that you, as the developer, can understand what the problem is, why it matters, and exactly how to fix it.

2. Issue Summary

The following table lists all issues found. Issues are sorted by severity.

Issue ID	Severity	Title
BUG-001	**CRITICAL**	ObjectOpenHashSet used across multiple threads in Y0Plugin
BUG-002	**CRITICAL**	Hardcoded packet IDs in ChunkHandler (0x20, 0x0B, 0x10)
BUG-003	**CRITICAL**	Multi Block Change packet decoded with wrong byte layout
BUG-004	**CRITICAL**	PacketEvents.getAPI().init() called three times on startup
BUG-005	**HIGH**	Trailing single-quote in config.yml breaks YAML parsing
BUG-006	**HIGH**	Reflection used to call World.getMinHeight() -- direct API exists
BUG-007	**HIGH**	tryRelease() race condition between two async callbacks
BUG-008	**MEDIUM**	Useless variable copy in Swimming.handleSwimReady()
BUG-009	**MEDIUM**	TuffActionBase uses plugin.plugin double-reference pattern

BUG-010	LOW	plugin.yml api-version is 1.18, should be 1.21

3. Critical Issues

[BUG-001] ObjectOpenHashSet accessed from multiple threads in Y0Plugin

File	y0/Y0Plugin.java
Line	43 (field declaration), 264, 295, 297, 646
Severity	CRITICAL -- can cause server crash or silent data corruption

What the code does

Y0Plugin declares its "active in below-Y0" player set like this:

```
private final ObjectOpenHashSet<UUID> aib = new ObjectOpenHashSet<>();
```

This set is read and written from at least two different threads:

- The main server thread (event handlers: PlayerJoinEvent, PlayerQuitEvent)
- The Netty I/O thread (ChunkHandler callbacks, which call isPlayerReady())

Why this is a problem

ObjectOpenHashSet from the fastutil library is not thread-safe. When two threads read and write the set at the same time without synchronization, one of two things can happen. First, a ConcurrentModificationException is thrown, which terminates the thread -- in Netty this means the player's entire network connection is killed silently. Second, the internal array of the set becomes corrupted so that contains() returns wrong results. This means a player who should receive Y0 chunks does not receive them, or a player who should not receive them does. Because no exception is logged in the second case, this bug is very hard to diagnose in production.

How to fix it

Replace ObjectOpenHashSet<UUID> with a concurrent set backed by ConcurrentHashMap. This is the standard Java approach and has no meaningful performance cost for UUID lookups.

```
// BEFORE
private final ObjectOpenHashSet<UUID> aib = new ObjectOpenHashSet<>();

// AFTER
import java.util.Set;
import java.util.concurrent.ConcurrentHashMap;

private final Set<UUID> aib = ConcurrentHashMap.newKeySet();
```

No other changes are needed. ConcurrentHashMap.newKeySet() returns a Set<UUID> and supports all the same operations (add, remove, contains, clear, iterator).

[BUG-002] Hardcoded packet IDs will break on version mismatch

File	netty/ChunkHandler.java
Line	71, 77, 82
Severity	CRITICAL -- wrong packets processed, chunks silently corrupted

What the code does

ChunkHandler intercepts raw bytes in the Netty pipeline and identifies packets by their numeric ID:

```
if (packetId == 0x20) { handleChunkPacket(...); return; }
if (packetId == 0x0B && viaBlocks != null) { handleBlockChange(...); return; }
if (packetId == 0x10 && viaBlocks != null) { handleMultiBlockChange(...); return; }
```

Why this is a problem

Minecraft packet IDs are not stable. They change between minor game versions. The IDs 0x20 (Chunk Data), 0x0B (Block Update), and 0x10 (Multi Block Change) are specific to one exact server version. When the server is updated, or when ViaVersion remaps the packet stream, these checks will silently match the wrong packet or match nothing at all. The plugin will then either process the wrong data (corrupted chunk visuals) or skip valid packets entirely. Because there is no error thrown, this is extremely hard to debug in production.

How to fix it

Use PacketEvents to identify packets by type instead of raw ID. PacketEvents is already a dependency of this plugin. Replace the raw-ID check with a proper type check:

```
// BEFORE -- fragile, breaks on any version change
if (packetId == 0x20) { handleChunkPacket(...); }

// AFTER -- use PacketEvents type system
import com.github.retrooper.packetevents.event.PacketSendEvent;
import com.github.retrooper.packetevents.protocol.packettype.PacketType;

@Override
public void onPacketSend(PacketSendEvent event) {
    if (event.getPacketType() == PacketType.Play.Server.CHUNK_DATA) {
        handleChunkPacket(event);
    } else if (event.getPacketType() == PacketType.Play.Server.BLOCK_CHANGE) {
        handleBlockChange(event);
    } else if (event.getPacketType() == PacketType.Play.Server.MULTI_BLOCK_CHANGE)
{
```

```
            handleMultiBlockChange(event);
        }
    }
```

This approach does not depend on any specific numeric ID and works across all server versions that PacketEvents supports.

[BUG-003] Multi Block Change packet decoded with wrong byte layout

File	netty/ChunkHandler.java
Line	115 -- 120 (handleMultiBlockChange method)
Severity	CRITICAL -- chunk coordinates are always wrong, ViaBlocks sends bad data

What the code does

The handleMultiBlockChange method tries to read the chunk X and Z coordinates from the Multi Block Change packet like this:

```
buf.resetReaderIndex();
buf.skipBytes(varIntLen(buf));   // skip packet ID
int cx = buf.readInt();          // reads 4 bytes as chunk X
int cz = buf.readInt();          // reads 4 bytes as chunk Z
```

Why this is a problem

Since Minecraft 1.16, the Multi Block Change packet does not store chunk X and Z as two separate integers. Instead, it packs both coordinates (and the Y sub-chunk index) into a single 8-byte long value using the following bit layout:

```
bits 63..42  = chunk X  (22 bits, signed)
bits 41..20  = sub-Y    (22 bits)
bits 19..0   = chunk Z  (20 bits, signed)
```

Calling readInt() twice reads the first four bytes as X and the next four bytes as Z. This gives completely wrong chunk coordinates. As a result, every call to viaBlocks.getExtraDataForMultiBlock() uses incorrect chunk positions, and either returns no data or returns data for the wrong chunk.

How to fix it

```
// BEFORE -- reads two separate ints, wrong since 1.16
int cx = buf.readInt();
int cz = buf.readInt();

// AFTER -- read one long and unpack the bits
long chunkSectionPos = buf.readLong();
int cx  = (int)(chunkSectionPos >> 42);
int csy = (int)((chunkSectionPos << 22) >> 42);  // sub-chunk Y, may be needed
later
```

```
int cz  = (int)((chunkSectionPos << 44) >> 44);
```

After the long, you also need to skip one boolean (the "suppress light updates" flag) before reading the block count. Add buf.readBoolean() before calling readVarInt(buf) for the count.

[BUG-004] PacketEvents.getAPI().init() called three times on startup

File	TuffX.java line 55, TuffActions.java line 62, y0/Y0Plugin.java line 179
Severity	CRITICAL -- can throw an IllegalStateException and prevent the plugin from loading

What the code does

Three different classes each call PacketEvents.getAPI().init() independently when the plugin enables:

```
// TuffX.java
PacketEvents.getAPI().init();

// TuffActions.java
PacketEvents.getAPI().init();

// Y0Plugin.java
PacketEvents.getAPI().init();
```

Why this is a problem

PacketEvents is designed to be initialized exactly once. Calling init() a second or third time while the library is already running either throws an IllegalStateException or produces undefined behavior in the packet pipeline. If the plugin is reloaded (for example via /reload or a plugin manager), multiple init() calls on the same PacketEvents instance will cause registration errors or duplicated packet listeners.

How to fix it

Move init() to exactly one place: TuffX.onEnable(), before any submodule is created. Remove the init() calls from TuffActions and Y0Plugin entirely.

```
// TuffX.java -- onEnable()
PacketEvents.getAPI().init();  // ONLY HERE

// then create submodules
this.tuffActions = new TuffActions(this);
this.y0Plugin = new Y0Plugin(this);

// TuffActions.java -- remove this line
// PacketEvents.getAPI().init();   <-- DELETE
```

```
// Y0Plugin.java -- remove this line
// PacketEvents.getAPI().init();   <-- DELETE
```

4. High Severity Issues

[BUG-005] Trailing single-quote in config.yml causes a YAML parse error

File	src/main/resources/config.yml
Line	72
Severity	HIGH -- viaentities section fails to load; max-distance is always -1 regardless of user setting

What the code does

The viaentities section of config.yml contains this line:

```
max-distance: -1'
```

Why this is a problem

The trailing single-quote character makes this an invalid YAML value. Some YAML parsers will throw an exception when loading the file and the entire viaentities section will be skipped. Others may treat the value as the string "-1'" instead of the integer -1, which will cause getInt("viaentities.max-distance") to silently return 0 (the default for a missing integer key) rather than -1. Either way, the user's setting for max-distance is never applied.

How to fix it

```
# BEFORE (broken)
max-distance: -1'

# AFTER (correct)
max-distance: -1
```

[BUG-006] Reflection used to call World.getMinHeight() -- direct API is available

File	viablocks/CustomBlockListener.java
Line	620 -- 632
Severity	HIGH -- unnecessary performance cost, crashes silently on reflection failure

What the code does

To call World.getMinHeight(), the code uses Java reflection instead of calling the method directly:

```
private int getMinHeight(World world) {
    return worldMinHeights.computeIfAbsent(world.getName(), k -> {
        try {
            Method method = world.getClass().getMethod("getMinHeight");
            Object value = method.invoke(world);
            if (value instanceof Integer) return (Integer) value;
        } catch (Exception e) {}
        return 0;
    });
}
```

Why this is a problem

World.getMinHeight() was added to the Bukkit API in version 1.16.5 and is present in all supported versions of this plugin (1.18+). There is no reason to use reflection. Reflection is significantly slower than a direct method call because the JVM cannot inline or optimize it. The try-catch block around the reflection call silently swallows any exception, meaning that if the method name ever changes or if the cast fails, the method returns 0 (sea level) instead of the actual world minimum, causing Y0 processing to miss entire underground sections. Additionally, the reflection-based approach bypasses the JVM's method access checks on every call, even though computeIfAbsent caches the result per-world.

How to fix it

```
// BEFORE -- reflection, slow and brittle
private int getMinHeight(World world) {
    return worldMinHeights.computeIfAbsent(world.getName(), k -> {
        try {
            Method method = world.getClass().getMethod("getMinHeight");
            Object value = method.invoke(world);
            if (value instanceof Integer) return (Integer) value;
        } catch (Exception e) {}
        return 0;
    });
}

// AFTER -- direct API call, clean and fast
private int getMinHeight(World world) {
    return worldMinHeights.computeIfAbsent(world.getName(),
        k -> world.getMinHeight());
}
```

You can also remove the java.lang.reflect.Method import from the top of the file if it is no longer used elsewhere.

[BUG-007] Race condition in tryRelease() between two async callbacks

File	netty/ChunkHandler.java
Line	193 -- 199 (tryRelease method)
Severity	HIGH -- can release the same packet buffer twice, causing a Netty buffer use-after-free crash

What the code does

When a chunk packet is waiting for both a ViaBlocks callback and a Y0 callback, the tryRelease() method is called from whichever callback finishes last:

```
private void tryRelease(long key) {
    QueuedPacket q = queue.get(key);
    if (q != null && q.viaReady && q.y0Ready) {
        release(key);
    }
}
```

Why this is a problem

The check and the release are two separate operations. If both callbacks finish at almost exactly the same time (which is common under high load), both will read q.viaReady and q.y0Ready as true before either has called release(). This means release() is called twice for the same key. Inside release(), queue.remove(key) is the only guard, but if both threads call it simultaneously, one call sees a non-null QueuedPacket and so does the other -- remove() in ConcurrentHashMap is atomic, but two concurrent calls to remove(key) will both succeed if the first removal has not yet been reflected across CPU caches. The second release then calls q.buf.release() on an already-released buffer. In Netty, releasing a buffer whose reference count is already 0 throws an IllegalReferenceCountException, which kills the channel.

How to fix it

Use an AtomicInteger counter inside QueuedPacket to ensure release() is only called once:

```
// Add to QueuedPacket inner class
final AtomicInteger readyCount = new AtomicInteger(0);
final int waitingFor;  // set at construction time to 1 or 2

// In tryRelease()
private void tryRelease(long key) {
    QueuedPacket q = queue.get(key);
    if (q == null) return;
    if (q.readyCount.incrementAndGet() >= q.waitingFor) {
        release(key);
    }
}
```

Because AtomicInteger.incrementAndGet() is atomic, exactly one thread will see the count reach waitingFor, and only that thread proceeds to release().

5. Medium and Low Severity Issues

[BUG-008] Useless variable copy in Swimming.handleSwimReady()

File	tuffactions/swimming/Swimming.java
Line	38
Severity	MEDIUM -- no functional impact, but misleads the reader

What the code does

```
public void handleSwimReady(Player player) {
    if (!isEnabled()) return;
    Player newPlayer = player;   // <-- this line
    plugin.plugin.getServer().getScheduler().runTaskLater(plugin.plugin, () -> {
        ...
        sendSwimState(newPlayer, swimmingPlayer, true);
    }, 20L);
}
```

Why this is a problem

The line "Player newPlayer = player;" creates a second local variable that points to exactly the same object as player. It does not create a copy of the player, it does not change anything about the lambda capture, and it is not required for the lambda to compile (the parameter player is effectively final). Any developer reading this code will spend time trying to understand why newPlayer exists and whether there is some subtle reason the original player reference cannot be used. There is no such reason.

How to fix it

```
// BEFORE
Player newPlayer = player;
plugin.plugin.getServer().getScheduler().runTaskLater(plugin.plugin, () -> {
    sendSwimState(newPlayer, swimmingPlayer, true);
}, 20L);

// AFTER
plugin.plugin.getServer().getScheduler().runTaskLater(plugin.plugin, () -> {
    sendSwimState(player, swimmingPlayer, true);
}, 20L);
```

[BUG-009] TuffActionBase accesses the plugin through a double-reference chain

File	tuffactions/TuffActionBase.java (and all subclasses)
Severity	MEDIUM -- works correctly but is confusing and breaks encapsulation

What the code does

TuffActionBase holds a reference to a TuffActions object (called "plugin"), and then accesses the actual TuffX plugin through it using the pattern plugin.plugin:

```
public TuffActionBase(TuffActions plugin, ...) {
    this.plugin = plugin;
    plugin.plugin.getConfig().addDefault(...);  // plugin.plugin
}
```

Why this is a problem

The field name "plugin" on TuffActions that refers to a TuffX instance is confusing because TuffActions itself is not a plugin -- it is a manager class. This naming collision means that from inside TuffActionBase, the sentence "plugin.plugin.getConfig()" must be read as "the TuffActions manager, then its TuffX plugin reference, then the config". A cleaner design exposes the config and logger access through helper methods on TuffActions itself, so subclasses never need to reach through two levels of indirection.

Suggested refactoring

```
// In TuffActions.java -- add convenience accessors
public FileConfiguration getConfig() {
    return plugin.getConfig();
}
public Logger getLogger() {
    return plugin.getLogger();
}

// In TuffActionBase.java -- now readable
plugin.getConfig().addDefault(configPath + ".enabled", defaultEnabled);
```

[BUG-010] plugin.yml api-version is set to 1.18 instead of 1.21

File	src/main/resources/plugin.yml
Severity	LOW -- functional, but disables several Paper 1.21 optimizations

What the code does

```
api-version: 1.18
```

Why this is a problem

Paper uses the api-version field to decide which compatibility behaviors to apply to the plugin. When a plugin declares api-version: 1.18, Paper applies legacy compatibility shims designed for plugins that were written before 1.20 or 1.21 changes. These shims can mask bugs and disable certain scheduling and event optimizations introduced in Paper 1.19 through 1.21. Since this plugin explicitly targets Paper 1.21.x and uses APIs that are only available from 1.18 onward anyway, there is no reason to declare a lower version.

How to fix it

```
# BEFORE
api-version: 1.18

# AFTER
api-version: 1.21
```

6. Changes Already Improved Since v1.2.0

The following issues were identified when comparing v1.2.0 to the current main branch. They have been correctly resolved and are documented here for completeness.

- **TuffActionBase introduced:** Swimming, CreativeMenu, and Restrictions each now extend TuffActionBase. The static boolean flags swimmingEnabled, creativeEnabled, and restrictionsEnabled in TuffActions have been removed. Each module manages its own enabled state, is properly cleaned up on disable(), and the debug flag is handled uniformly.
- **Config defaults use addDefault():** Settings are now registered with addDefault() and the combination of copyDefaults(true) + saveConfig() in TuffX.onEnable() ensures that all settings appear in config.yml on first run.
- **ViaEntitiesPlugin debug default fixed:** The debug flag for viaentities now defaults to false instead of true, eliminating the log flood that occurred in the previous release.
- **Unloaded chunk crash in ChunkHandler fixed:** A world.isChunkLoaded() guard was added before calling getExtraDataForSingleBlock(). A comment in the code confirms this was discovered through a real crash in production.
- **Restrictions.validateConfig() removed:** The manual config validation method has been replaced by addDefault() calls, which is the correct Bukkit pattern.
- **Debug logging level corrected:** Errors during plugin message delivery are now logged at INFO level behind the debug flag instead of always logging at WARNING.
- **Restrictions command guard added:** The /restrictions command now returns an informative message when the restrictions feature is disabled, instead of silently executing with no effect.
- **config.yml viaentities section added:** The viaentities block is now present in the default config.yml, making all three settings (enabled, debug, max-distance) visible to server administrators.

7. Priority Action List

Fix these issues in this order. Issues higher in the list have greater impact on server stability.

Priority	Issue ID	Effort	Action
1	BUG-001	Low (2 lines)	Replace ObjectOpenHashSet with ConcurrentHashMap.newKeySet()
2	BUG-004	Low (delete 2 lines)	Remove duplicate PacketEvents.init() calls from TuffActions and Y0Plugin
3	BUG-005	Trivial (1 char)	Remove trailing single-quote from max-distance: -1' in config.yml
4	BUG-007	Medium (add AtomicInteger)	Fix tryRelease() race condition with an AtomicInteger counter
5	BUG-002	High (refactor)	Replace hardcoded packet IDs with PacketEvents type checks
6	BUG-003	Medium (fix byte parsing)	Fix Multi Block Change coordinate parsing to read one long instead of two ints
7	BUG-006	Low (remove reflection)	Replace reflection-based getMinHeight() with direct world.getMinHeight() call
8	BUG-010	Trivial (1 line)	Update api-version in plugin.yml from 1.18 to 1.21
9	BUG-008	Trivial (delete 1 line)	Remove the useless "Player newPlayer = player;" line in Swimming
10	BUG-009	Low (refactor)	Add getConfig()/getLogger() helpers to TuffActions to remove double-reference

---