Strategic Technical Roadmap: Advanced Feature Integration and Architectural Optimization for the B0t Project

1. Executive Summary

This comprehensive technical report outlines a strategic roadmap for the evolution of the B0t project, a Minecraft automation and reconnaissance tool. Based on an extensive analysis of current open-source ecosystems, server protocols, and automation methodologies, this document proposes a suite of advanced features designed to elevate the project from a standard client implementation to a high-performance, enterprise-grade automation platform. The analysis leverages data from leading projects in the ecosystem, including masscan, mineflayer, and ServerSeekerV2, to synthesize a unified architecture that addresses scalability, resilience, and intelligence.

The recommendations provided herein are categorized into five primary domains: **Network Reconnaissance & Scanning**, **Autonomous Agent Intelligence**, **Anarchy & Exploration Heuristics**, **Infrastructure & Data Management**, and **Operational Resilience & Security**.

The proposed enhancements aim to address specific technical challenges identified in the research:

1. **Scalability:** Moving beyond single-threaded operations to asynchronous, distributed scanning architectures capable of indexing the entire IPv4 address space.
2. **Resilience:** Implementing robust proxy rotation and error recovery mechanisms to maintain uptime in hostile server environments where IP bans are frequent.
3. **Intelligence:** Integrating heuristic algorithms for detecting points of interest (bases, stashes) rather than relying solely on brute-force search, utilizing deep protocol analysis to identify server software variants.
4. **Interoperability:** Establishing real-time data pipelines between the bot, external databases (MongoDB/SQLite), and notification services like Discord to create a responsive Command and Control (C2) loop.
5. **Ethics & Compliance:** Incorporating automated opt-out mechanisms and exclusion list processing to ensure the tool operates within responsible boundaries.

This report serves as both a feature specification and an architectural guide, providing the theoretical underpinnings and practical implementation details necessary to execute these improvements.

2. Strategic Context and Ethical Framework

Before diving into the technical implementation of advanced reconnaissance and automation features, it is essential to establish the strategic context of the tool. The landscape of Minecraft automation has shifted significantly from simple client-side modifications to complex, headless bot frameworks running on distributed infrastructure.

2.1. The Evolution of Minecraft Automation

Early automation relied on "hacked clients"—modifications to the actual game jar (e.g., Wurst, Meteor) that required a graphical user interface and heavy resource consumption. The current paradigm, exemplified by mineflayer and B0t, utilizes "headless" clients. These are lightweight Node.js applications that implement the Minecraft protocol directly, allowing them to run without a display or GPU. This shift enables massive scalability; a single Virtual Private Server (VPS) can host dozens or even hundreds of bot instances, provided the architecture is optimized for asynchronous I/O.
However, this scalability introduces complexity. Managing the state, inventory, and physics of a hundred bots requires a fundamental rethink of how the application handles data. It moves the project from the realm of "game modding" into "distributed systems engineering."

2.2. Ethical Scanning and Opt-Out Mechanisms

With the power to scan the entire internet comes the responsibility to do so ethically. The research indicates that advanced scanners like ServerSeekerV2 have implemented "Automatic Opting Out" features. This is a critical addition for B0t to be considered a professional-grade tool rather than a malicious utility.
Implementation of Opt-Out Protocols: To respect server administrators' privacy, the scanner must parse the Message of the Day (MOTD) for specific "do not scan" signals. A standard convention has emerged where administrators add a specific color code sequence—such as §b§d§f§d§b—to their server description. This sequence renders as invisible to players but is detectable by a protocol scanner.

- **Mechanism:** During the "Fingerprinting" stage (detailed in Section 3), the bot parses the raw MOTD string. If the exclusion sequence is detected, or if the server description contains keywords like "Private," "Do Not Scan," or "Family," the IP is immediately discarded and added to a permanent blacklist in the database.
- **Exclusion Lists:** The scanner must also respect a static exclude.txt file. This file contains CIDR blocks for entities that should never be scanned, such as government infrastructure (US-DoD blocks) or Internet Exchange Points (IXPs), to prevent abuse complaints and potential legal repercussions.

3. Advanced Network Reconnaissance and Scanning Architecture

The foundation of any advanced Minecraft utility is its ability to discover and index servers efficiently. While basic bot implementations typically connect to a single target, transforming B0t into a reconnaissance tool requires integrating high-throughput network scanning capabilities.

3.1. Integration of Masscan for High-Throughput Discovery

Standard TCP connection attempts in Node.js using the net module are insufficient for large-scale network discovery due to the overhead of the V8 engine and the operating system's standard TCP stack. To achieve scanning rates capable of indexing the IPv4 address space (approx. 3.7 billion usable IPs), integration with masscan—an asynchronous TCP port scanner—is recommended. masscan operates by bypassing the system's TCP/IP stack and injecting raw SYN packets directly into the network driver, allowing it to achieve millions of packets per second.

3.1.1. Architectural Pattern: The "Spawn-and-Pipe" Model

Rather than attempting to rewrite the scanning logic in JavaScript, which would be performantly prohibitive, the recommended approach is to utilize Node.js's child_process module to spawn a masscan instance and consume its output in real-time. This allows B0t to leverage the raw packet injection speed of masscan while maintaining the logic flexibility of JavaScript for processing results.
Implementation Strategy: The masscan process should be spawned with flags optimized for machine parsing. The -oJ flag forces JSON output, or the --interactive mode can be used for line-by-line processing. The Node.js parent process must establish a stream reader on the child process's stdout.

- **Process Isolation and Worker Threads:** Given the high volume of data, the scanning module should run in a separate Worker Thread or even a distinct microservice. The parsing of the stdout stream is CPU-intensive; if run on the main event loop, it could block the bot's ability to respond to keep-alive packets from the Minecraft server, causing disconnects. The child_process.spawn method ensures the scanner runs independently, but the data consumption must be handled asynchronously.
- **Backpressure Handling and Buffering:** High-speed scanning generates data faster than most databases can write. If masscan finds 10,000 servers per second, attempting to insert 10,000 documents into MongoDB instantly will cause a bottleneck. The implementation must include a transform stream with internal buffering (e.g., using stream.Transform). This buffer accumulates discovered IPs and flushes them to the database in bulk batches (e.g., every 1,000 records or every 5 seconds). This technique, known as "batching," dramatically reduces database I/O overhead.
- **Dynamic Command Construction:** The wrapper must dynamically construct arguments to allow users to define target subnets. Users might want to scan the entire internet (0.0.0.0/0) or specific subsets like residential ranges (scan24s logic found in other scanners). The command construction must also automatically append the -e or --excludefile argument pointing to the exclude.txt mentioned in Section 2.2 to ensure compliance.

3.1.2. Verification Layer: The Two-Stage Scan Architecture

masscan detects open ports (SYN/ACK response), but it does not verify if the service listening on port 25565 is actually a Minecraft server. It could be a web server, an SSH tunnel, or a honeypot. Therefore, a "Two-Stage Scan" architecture is necessary:

1. **Stage 1 (Discovery):** masscan identifies an open TCP port. This is the "fast path."
2. **Stage 2 (Fingerprinting):** The Node.js application attempts a Minecraft-specific handshake (Server List Ping). This is the "slow path" and acts as a filter.

This architecture mirrors the design of professional scanners like ServerSeekerV2 and Mine-Scan, which separate the raw port detection from the application-layer verification to maximize throughput.

3.2. Deep Protocol Analysis and Fingerprinting

Simply detecting a server is often insufficient; advanced reconnaissance requires identifying the underlying server software to determine compatibility or vulnerability status. The Minecraft ecosystem is fragmented into various server implementations: Vanilla, Bukkit, Spigot, Paper, Forge, and Fabric. Each has unique protocol quirks.

3.2.1. Server List Ping Protocol Mechanics

The Minecraft Server List Ping protocol involves a specific handshake sequence. The client sends a Handshake packet (Packet ID 0x00) with a "Next State" field set to 1 (Status), followed by a generic Request packet. The server responds with a JSON payload containing the server's description, version, and favicon.
Advanced Data Extraction via Regex: The version.name field in the JSON response often contains more than just the version number. Server administrators or software forks frequently append identifiers. B0t should implement a robust Regular Expression (Regex) engine to parse this string.

- **Software Identification:** A regex such as /(Paper|Spigot|Forge|Fabric|BungeeCord)\s*(\d+\.\d+(\.\d+)?)/i can extract the software type. This distinction is vital:
  - **Paper/Spigot:** Indicates a plugin-based server. These often have stricter anti-cheat (NoCheatPlus, Matrix) and optimized physics, which might affect bot movement calculations.
  - **Forge/Fabric:** Indicates a modded server. These may require the bot to undergo a "Mod Handshake" (FML protocol) to join. If B0t detects a Forge server (versions 1.7–1.12), it must look for the modinfo object in the handshake to list the specific mods (e.g., "Create", "Pixelmon"). This allows users to search for servers running specific modpacks.
  - **Vanilla:** Indicates a standard Mojang server. These are often less protected and more susceptible to vanilla exploits or "unnatural block" scanning.

3.2.2. Chat Component Parsing (MOTD)

The Message of the Day (MOTD) is returned as a JSON text component, which may contain complex nested structures for formatting (colors, bold, obfuscated text). A robust parser is required to convert this JSON into usable formats.

- **Raw Text for Search:** The parser must strip all color codes (§a, §l) and JSON formatting to produce a clean string for the search index. This allows users to search for "Anarchy" or "Survival" without worrying about color codes breaking the match.
- **HTML for Visualization:** For the Web UI (Section 5), the parser should convert the Minecraft color codes into HTML <span> tags with corresponding CSS classes (e.g., §4 becomes <span class="mc-dark-red">).
- **Obfuscated Text Handling:** The character §k creates "magic" text that changes rapidly. The parser *must* strip this out entirely, as indexing it would pollute the database with random characters. This parsing layer is also where the "Opt-Out" detection (checking for §b§d§f§d§b) occurs.

3.3. Vulnerability and Configuration Scanning

For a project aimed at advanced users, integrating vulnerability scanning capabilities adds significant value. This involves passively detecting misconfigurations or known exploits based on the server's protocol response.

- **Auth Type Detection (Cracked vs. Premium):** Analyzing the response for "offline mode" (cracked) vs. "online mode" (premium) servers is a high-priority feature. While the ping protocol does not explicitly state this, some server implementations leak this via the MOTD or specific error messages during a login attempt. Additionally, checking if the server enforces secure profile properties can indicate its auth status.
- **Plugin Enumeration via Query Protocol:** If the server has the enable-query=true setting in server.properties, it listens on a UDP port (usually 25565) for the GameSpy 4 protocol. This protocol is distinct from the TCP Minecraft protocol and leaks a full list of installed plugins. B0t should implement a UDP query client (using libraries like craftping or mc-server-status) to retrieve this list. This provides deep insight into the server's tech stack—for example, identifying "NoCheatPlus" or "Matrix" suggests a hardened target, while a lack of plugins suggests a vanilla target.
- **Paper Server Player Hiding:** Paper servers often use a privacy feature that hides the player sample in the ping response, showing "???" instead of names. The scanner must be programmed to recognize this specific behavior (often returned as a specific protocol response) and log it as "Hidden Players" rather than treating it as an error or zero players.

4. Autonomous Agent Intelligence and Interaction

Moving beyond the external scanner, the core bot functionality—powered by mineflayer—can be significantly enhanced. The current standard for Minecraft bots involves static scripts; the goal for B0t should be dynamic, reactive autonomy. This requires implementing complex state machines and heuristic decision-making.

4.1. Advanced Pathfinding and Navigation

Navigation is the cornerstone of any physical bot interaction. The basic mineflayer-pathfinder plugin uses A* (A-Star) algorithms, but it often struggles with complex environments or specific movement requirements.

4.1.1. Custom Movement Heuristics and Cost Functions

To improve navigation, B0t should implement custom cost functions for the pathfinder. The standard A* implementation calculates the shortest path based on distance. However, an intelligent bot must consider safety and stealth.

- **Avoidance Zones:** Users should be able to define "danger zones" (e.g., lava pools, claim borders, or high-traffic spawn regions) that the pathfinder assigns an infinite or very high traversal cost. This effectively creates a "geofence" that the bot will navigate around.
- **Parkour Modules:** Integrating logic from mineflayer-parkour allows the bot to perform difficult jumps (e.g., 3-block jumps, neo-jumps) which are necessary for navigating the rugged, griefed terrain often found in anarchy servers. This requires precise tick-based physics simulation to time the sprint and jump packets perfectly.

4.1.2. Elytra and Trident Flight Physics

For post-game content and rapid exploration, standard walking is inefficient. Implementing flight logic using Elytra requires sophisticated physics simulation. The bot must calculate pitch and yaw to maintain momentum, using prismarine-physics to simulate drag, gravity, and kinetic energy.

- **Algorithm:** The bot must periodically adjust its pitch to "swoop"—gaining speed by diving and then pulling up to gain altitude. This allows for rapid long-distance travel, which is essential for the "Base Hunting" features discussed in Section 6. The bot must also monitor durability and firework rocket usage to prevent mid-air failure.

4.2. Inventory Management and "Looting" Logic

A major utility for bots is resource acquisition. Enhancing the B0t inventory system involves moving from simple "pick up item" commands to complex inventory state management.

4.2.1. Intelligent Loot Selection and Value Density

The mineflayer-collectblock plugin provides a foundation, but an advanced implementation requires intelligent filtering. The bot should accept a configurable "wishlist" (e.g., Diamonds, Totems, Gapples) and a "trashlist" (e.g., Cobblestone, Dirt, Rotten Flesh).

- **Value Density Algorithm:** Upon opening a chest or killing an entity, the bot scans the window items. It calculates the "value density" of its current inventory slots. If the inventory is full, it performs a value comparison: checking if the item in the chest is of higher value than the lowest-value item in its inventory. If so, it drops the trash item and loots the valuable one. This ensures the bot maximizes the utility of its limited inventory space during long autonomous runs.

4.2.2. Container Interaction (The "Chest Stealer")

In the context of anarchy servers or raiding, speed is paramount. A "Chest Stealer" module automates the extraction of items from a container faster than a human can.

- **Technical Implementation:**
  1. Bot detects windowOpen event.
  2. Iterates through slots 0 to window.inventoryStart (the chest slots).
  3. Checks item IDs against the wishlist.
  4. Issues window.click packets with shift + click (Mode 1) to instantly move items to the bot's inventory.
- **Anti-Cheat Evasion (Critical):** Sending packets too fast will trigger server-side anti-cheat plugins (like NoSlowDown or InventoryMove). To bypass this, the bot must implement a stochastic delay. Instead of a fixed loop, it should use a randomized timeout (e.g., Math.random() * 100 + 50 ms) between clicks. This variance mimics human reaction time imperfections, bypassing heuristic detection that looks for machine-perfect timing.

4.3. Combat and Self-Preservation (PVP)

To survive in hostile environments, B0t needs defensive capabilities that react faster than human reflexes.

- **Auto-Totem:** The bot must monitor its health and offhand slot. If a Totem of Undying is used (popped), the bot must immediately locate another totem in the inventory and move

it to the offhand slot. This requires high-priority packet handling. The windowClick packet for the totem swap should be prioritized in the outgoing queue to ensure it arrives before the next damage tick.
- **Crystal Aura:** For advanced combat (End Crystal PVP), the bot needs to calculate explosion damage. It scans for valid obsidian placement blocks, calculates the damage to the target entity versus self-damage, and places/breaks crystals accordingly. This requires raycasting to check line-of-sight and utilizing the mineflayer-pvp plugin as a base. The bot must also predict enemy movement to place crystals where the enemy *will be*, not just where they are.

5. Infrastructure, Data Management, and Visualization

As B0t scales from a simple script to a complex tool, the infrastructure handling its data must evolve. Storing thousands of server IPs or millions of chat logs requires professional-grade database solutions.

5.1. Database Architecture: SQLite vs. MongoDB

The choice of database depends heavily on the scale of the intended deployment. The research highlights a clear trade-off between these two systems.

5.1.1. Local/Single-User: SQLite

For users running B0t locally for personal use, **SQLite** is the optimal choice. It requires no external server process and stores data in a single file (.db), making it highly portable.
- **Schema Design:** Relational tables are suitable for inventory tracking or simple coordinate logging.
  - Table: Servers (IP, Port, MOTD, Version, LastSeen).
  - Table: Players (UUID, Username, LastSeenServer).
- **Performance Characteristics:** SQLite performs exceptionally well for read-heavy workloads or single-threaded writes. However, its single-file architecture limits scalability. It locks the entire database file during a write operation. If the masscan process is attempting to write 5,000 records per second, SQLite will become a bottleneck due to lock contention.

5.1.2. Enterprise/High-Volume: MongoDB

For high-throughput scanning (e.g., indexing the entire internet) or distributed bot swarms, **MongoDB** is the superior choice.
- **Document Model Flexibility:** Server ping responses are JSON objects. MongoDB is a document store, meaning it can save the ping response *exactly* as received without complex schema mapping. This is crucial because Minecraft server data is highly variable; some servers return simple string MOTDs, while others return complex JSON objects with extra mod lists or player samples. MongoDB handles this polymorphism natively.
- **Write Throughput and Sharding:** MongoDB supports high concurrent write operations, which is essential when masscan is piping thousands of results per second. It uses

document-level locking (WiredTiger engine) rather than file-level locking. Furthermore, for massive datasets (terabytes of server history), MongoDB supports sharding—distributing data across multiple servers—allowing the database to scale horizontally.
- **Geospatial Indexing:** MongoDB has built-in support for 2dsphere indexes. If B0t resolves Server IPs to Geo-locations (Latitude/Longitude), MongoDB allows for efficient queries like "Find all servers within 500km of Berlin" using the $near operator. This is a powerful feature for the visualization layer.

5.2. Visualization and User Interface

Command-line interfaces (CLI) are powerful but lack intuitiveness for data visualization. B0t should expose a Web UI to make the data actionable.

5.2.1. Web-Based Inventory Viewer

Using mineflayer-web-inventory or prismarine-viewer, the bot can host a local web server (e.g., using express). This allows the user to view the bot's inventory, health, hunger, and armor status in a browser window in real-time. This is particularly useful for "headless" bots running on a remote VPS, giving the operator a visual check on the bot's status without needing to log into the game.

5.2.2. Map Visualization with Leaflet.js

For the server scanner component, a map interface is highly engaging and useful for identifying regional server clusters.
1. **GeoIP Resolution:** The system should use geoip-lite (for a free, local database) or the MaxMind GeoLite2 database to resolve Server IPs to latitude/longitude coordinates. While geoip-lite is faster (memory-mapped), MaxMind offers higher accuracy for city-level data.
2. **Frontend Implementation:** Implement Leaflet.js, a lightweight open-source JavaScript mapping library. It is mobile-friendly and supports clustering out of the box.
3. **Clustering:** Displaying 50,000 servers on a map will crash a browser. B0t must use the Leaflet.markercluster plugin. This groups nearby markers into a single cluster icon (e.g., a circle saying "500") which expands when clicked. This maintains UI performance while visualizing massive datasets.

5.3. Real-Time Notifications via Discord Webhooks

For a bot running autonomously, real-time alerting is critical. Integration with Discord Webhooks allows B0t to push notifications without maintaining a full Discord bot session.
- **Payload Formatting:** The bot should construct rich JSON payloads (embeds) for the webhook.
  - *Color Coding:* Red for disconnects, Green for found bases/servers, Gold for rare loot acquisition.
  - *Rich Fields:* The embed should contain fields for Coordinates (X, Y, Z), Server IP, and potentially a link to a map view.
- **Rate Limiting Strategy:** Discord enforces strict rate limits on webhooks (approximately 5 requests per 2 seconds). B0t must implement a notification queue. If the scanner finds 50 bases instantly, it cannot fire 50 webhooks. It must buffer these events and send them

sequentially or aggregate them into a single "Summary Report" webhook to avoid getting the webhook URL blocked.

6. Heuristic Exploration and Anarchy Operations

One of the most requested feature sets for Minecraft bots relates to "Anarchy" servers (e.g., 2b2t, 9b9t). These environments require specialized algorithms for finding player bases and analyzing world data, as the map is too large (60 million x 60 million blocks) to search randomly.

6.1. Base Hunting Algorithms

"Base Hunting" is the process of locating player-made structures in a vast procedural world. Random searching is mathematically futile. Therefore, B0t must employ heuristic analysis to predict where players are likely to be.

6.1.1. Chunk Analysis: The "New vs. Old" Border

Minecraft world generation algorithms change between versions. When a server updates (e.g., from 1.12 to 1.16), the terrain generation code changes.

- **The Principle:** If a chunk was generated in version 1.12, the bedrock pattern or ore distribution will follow the 1.12 algorithm. If the bot detects a chunk that matches the 1.12 pattern immediately adjacent to a chunk that matches the 1.16 pattern, it indicates a "chunk border."
- **Significance:** Chunk borders often indicate the edge of a player's exploration path. Players generate new chunks as they travel; following these trails of "New Chunks" often leads to a stash or base.
- **Implementation:** The bot uses mineflayer to read the chunk data (prismarine-chunk). It calculates the distribution of specific blocks (e.g., Bedrock at y=0 to y=5). It compares this against known seeds or generation patterns to flag "Old Generation" chunks. This effectively allows the bot to act as an archaeologist, tracing the history of the server's exploration.

6.1.2. Unnatural Block Density Scanning

A more direct method is scanning for "unnatural" blocks—items that do not spawn in the terrain generation code.

- **Density Thresholds:** The bot scans loaded chunks (16x16x256 area) for blocks such as chest, furnace, shulker_box, beacon, bed, and standing_sign.
- **Technical Nuance:** The scanner must distinguish between player-placed blocks and naturally generating structures. For example, a chest can spawn in a dungeon or village. However, naturally spawned chests are rarely grouped in high density. The algorithm should set a threshold (e.g., > 5 chests in a 3-chunk radius). It should also look for blocks that *never* spawn naturally in the Overworld, such as shulker_box or beacon.
- **Deep Anomaly Detection:** As per advanced research, the bot can also scan for "illegal" or "glitched" blocks, such as headless pistons or floating torches, which are often created by players using exploits to break bedrock or duplicate items. Detecting these highly specific block states requires reading the block's metadata/NBT tags.

6.2. World Downloading and Proxying

Users often want to save the builds they find. A "World Downloader" feature allows the bot to save the chunks it sees to a local file.

- **Proxy-Based Approach:** While mineflayer can parse chunks, saving them to a standard Anvil format (.mca) is complex to implement from scratch. A robust solution is to use a proxy architecture. The bot acts as a Man-in-the-Middle (MITM). It connects to the server, and a real Minecraft client connects to the bot.
- **Chunk Interception:** As the server sends Map Chunk packets, the bot/proxy intercepts them and writes the raw byte array to a local storage provider (e.g., prismarine-provider-anvil). This ensures that the saved world is a byte-perfect copy of what the server sent, preserving all block data, entities, and NBT tags.

7. Operational Resilience and Security

Running a bot, especially for scanning or anarchy purposes, places the software in a hostile environment. Servers actively try to detect, block, and ban bots. Resilience is therefore a key architectural requirement.

7.1. Proxy Support and Rotation

Servers aggressively ban IPs associated with scanning or botting. To maintain operations, B0t must support SOCKS5 proxies.

- **Library Selection:** Use the socks npm package or proxy-agent. These libraries allow Node.js to tunnel its TCP connections through a third-party server.
- **Rotation Logic:** The bot should accept a list of proxies. On a connection error (e.g., ECONNRESET, Kicked: You are banned), it should automatically rotate to the next proxy in the list.
- **State Machine Integration:** The proxy rotation logic must be integrated into the mineflayer state machine. If a bot is kicked, the state machine transitions to a Reconnecting state, selects a new proxy agent, and re-initializes the createBot function. This ensures the bot can run perpetually without user intervention.

7.2. Anti-Bot Bypass Strategies

Many servers use plugins like "AntiBotDeluxe" or "GrimAC" which challenge players to perform movements (jump, rotate) or solve CAPTCHAs to prove they are human.

- **Movement Simulation:** The bot should implement "human-like" physics. Rather than sending perfect packet rotations (e.g., changing yaw from 0 to 180 in 1 tick), it should interpolate view angles over time (e.g., smoothing the yaw/pitch changes over 10 ticks). This mimics the acceleration of a mouse movement.
- **Chat Captcha Solvers:** Simple math captchas ("Please type 4 + 5") can be solved using Regex parsers on the chat event. However, some servers use image map captchas. Solving these would require integrating OCR (Optical Character Recognition) libraries like tesseract.js, which may be too heavy for a lightweight bot. A middle-ground solution is to forward the captcha image (rendered map) to the Discord C2 webhook and allow the user

to type the solution in Discord, which the bot then sends to the server.
- **Spam Avoidance:** To avoid being flagged by chat filters (like AntiSpammer), the bot must vary its message timing. Implementing a queue that sends messages with a randomized delay (jitter) prevents the "machine-gun" chat behavior that triggers bans.

8. Implementation Roadmap and Conclusion

To execute this vision, the development should be phased to manage complexity:

Phase 1: The Scanner Core

- **Goal:** Build the masscan wrapper and the Two-Stage verification system.
- **Tasks:** Implement Node.js child process spawning for masscan, build the Transform stream for buffering IPs, and create the mineflayer/node-minecraft-protocol pinger with Regex version detection. Set up the MongoDB schema.

Phase 2: The Bot Framework

- **Goal:** Enhance the mineflayer instance with autonomous capabilities.
- **Tasks:** Implement mineflayer-pathfinder with custom costs, integration of mineflayer-state-machine, and the basic Discord Webhook notification system.

Phase 3: Intelligence Modules

- **Goal:** Add advanced heuristics for anarchy and looting.
- **Tasks:** Develop the "Unnatural Block" scanning heuristic (Chest/Shulker detection), the "Chest Stealer" logic with randomized delays, and the SOCKS5 proxy rotation mechanism.

Phase 4: The Interface

- **Goal:** Make the data accessible and beautiful.
- **Tasks:** Build the Express.js API, the Web UI with Leaflet.js mapping (clustering, GeoIP), and the real-time inventory viewer.

Summary of Key Technologies

Domain	Recommended Technology	Purpose
Core	mineflayer, node-minecraft-protocol	Minecraft protocol implementation
Scanning	masscan, masscan-node	High-speed port discovery
Database	mongodb (Enterprise) / sqlite3 (Local)	Data persistence
Geo-Location	geoip-lite, leaflet	Map visualization
Networking	socks, proxy-agent	IP rotation and anonymity
Utilities	mineflayer-pathfinder, mineflayer-collectblock	Bot movement and interaction

By following this roadmap, B0t will evolve from a simple script into a formidable tool capable of large-scale data gathering and sophisticated autonomous interaction within the Minecraft ecosystem. The integration of ethical scanning practices, robust infrastructure, and intelligent heuristics will ensure it remains a valuable asset for the community.

Quellenangaben

1. server-scanner · GitHub Topics, https://github.com/topics/server-scanner 2. Minecraft Server Scanner - Apify, https://apify.com/mr-brogrammer/minecraft-server-scanner 3. socks5 and mineflayer #1697 - Issuehunt OSS, https://oss.issuehunt.io/r/PrismarineJS/mineflayer/issues/1697 4. How to Rotate Proxies in Python (Step-by-Step Tutorial) - IPRoyal.com, https://iproyal.com/blog/how-to-rotate-proxies-in-python/ 5. Minecraft - Unnatural Blocks using Book/Chunk Dupe in Java - YouTube, https://www.youtube.com/watch?v=tcKaJdaBvUg 6. Minecraft Server Types Explained: Paper, Fabric & Forge - WiseHosting, https://wisehosting.com/blog/minecraft-server-types-explained 7. mineflayer examples - CodeSandbox, https://codesandbox.io/examples/package/mineflayer 8. Webhook Resource | Documentation | Discord Developer Portal, https://discord.com/developers/docs/resources/webhook 9. Funtimes909/ServerSeekerV2: [MIRROR] Blazingly fast Minecraft server scanner written in rust, rewrite of the original ServerSeeker with more features and way faster! - GitHub, https://github.com/Funtimes909/ServerSeekerV2 10. Scanning the Entire Internet for Minecraft Servers : r/programming - Reddit, https://www.reddit.com/r/programming/comments/u4gtnh/scanning_the_entire_internet_for_minecraft_servers/ 11. PrismarineJS/mineflayer: Create Minecraft bots with a powerful, stable, and high level JavaScript API. - GitHub, https://github.com/PrismarineJS/mineflayer 12. masscan | Kali Linux Tools, https://www.kali.org/tools/masscan/ 13. dumbasPL/masscan-node: Node.js wrapper for masscan - GitHub, https://github.com/dumbasPL/masscan-node 14. Child process | Node.js v25.2.1 Documentation, https://nodejs.org/api/child_process.html 15. Child Process Node.js v4.5.0 Manual & Documentation, https://nodejs.org/download/release/v4.5.0/docs/api/child_process.html 16. MongoDB vs SQLite - Key Differences - Airbyte, https://airbyte.com/data-engineering-resources/mongodb-vs-sqlite 17. Node.js spawn child process and get terminal output live - Stack Overflow, https://stackoverflow.com/questions/14332721/node-js-spawn-child-process-and-get-terminal-output-live 18. kgurchiek/Minecraft-Server-Scanner: A scanner to discover new Minecraft server ips. For use with https://github.com/kgurchiek/Minecraft-Server-Rescanner - GitHub, https://github.com/kgurchiek/Minecraft-Server-Scanner 19. Jael-G/Mine-Scan: Powerful multi-threaded IPv4 scanner to find Minecraft Servers - GitHub, https://github.com/Jael-G/Mine-Scan 20. Server List Ping - wiki.vg - GitHub Pages, https://c4k3.github.io/wiki.vg/Server_List_Ping.html 21. A regex for version number parsing - Stack Overflow, https://stackoverflow.com/questions/82064/a-regex-for-version-number-parsing 22. Explanation of Bukkit, Fabric, Spigot, and Paper? : r/admincraft - Reddit, https://www.reddit.com/r/admincraft/comments/n2nfhg/explanation_of_bukkit_fabric_spigot_and_paper/ 23. MCScans - Explore Minecraft Server Data, https://mcscans.fi/ 24. The list of Minecraft Server Software Hybrids that can run mods and plugins. - GitHub, https://github.com/Shawiizz/minecraft-hybrids 25. Minecraft Server Types - Shockbyte, https://shockbyte.com/help/knowledgebase/articles/minecraft-server-types 26. sfirew/minecraft-motd-parser - NPM,

https://www.npmjs.com/package/@sfirew/minecraft-motd-parser 27. tanishisherewithhh/ImperialsBot: Minecraft bot using mineflayer to chat spam, safeguard bases and many other utility functions. 3D display using prismarine viewer of the bot - GitHub, https://github.com/tanishisherewithhh/ImperialsBot 28. 0tii/Mc-Status: Async node.js implementation of the UDP Minecraft Server Query Protocol and TCP Minecraft Server List Ping Protocol. - GitHub, https://github.com/0tii/Mc-Status 29. craftping - NPM, https://www.npmjs.com/package/craftping 30. aternosorg/craftping - GitHub, https://github.com/aternosorg/craftping 31. PaperServerListPingEvent (paper-api 1.20.6-R0.1-SNAPSHOT API) - Javadocs, https://jd.papermc.io/paper/1.20.6/com/destroystokyo/paper/event/server/PaperServerListPingEvent.html 32. PrismarineJS/mineflayer-pathfinder: Pathfinding plugin that gives bot the ability to go from A to B - GitHub, https://github.com/PrismarineJS/mineflayer-pathfinder 33. Mineflayer Tutorials Ep. 2 Pathfinding - YouTube, https://www.youtube.com/watch?v=UWGSf08wQSc 34. A simple utility plugin for Mineflayer that add a higher level API for collecting blocks. - GitHub, https://github.com/PrismarineJS/mineflayer-collectblock 35. Top 10 Examples of mineflayer code in Javascript | CloudDefense.AI, https://www.clouddefense.ai/code/javascript/example/mineflayer 36. Inventory Manipulation · Issue #156 · PrismarineJS/mineflayer - GitHub, https://github.com/PrismarineJS/mineflayer/issues/156 37. MineZBots/mzb-mineflayer: Create Minecraft bots with a powerful, stable, and high level JavaScript API. - GitHub, https://github.com/MineZBots/mzb-mineflayer 38. Finding a chest, opening it. · Issue #20 · Darthfett/mineflayer-blockfinder - GitHub, https://github.com/Darthfett/mineflayer-blockfinder/issues/20 39. Adds support for basic PVP and PVE to Mineflayer bots. - GitHub, https://github.com/PrismarineJS/mineflayer-pvp 40. MongoDB vs. SQLite: Choosing the Right Database for Your Application - Sprinkle Data, https://www.sprinkledata.com/blogs/mongodb-vs-sqlite-choosing-the-right-database-for-your-application 41. Real cases where MongoDB outshines PostgreSQL/MariaDB/MySQL ? : r/node - Reddit, https://www.reddit.com/r/node/comments/snq7v6/real_cases_where_mongodb_outshines/ 42. Web based viewer for your mineflayer bot's inventory - GitHub, https://github.com/imharvol/mineflayer-web-inventory 43. maxmind vs geoip-lite vs geoip-country | Geolocation and IP Address Lookup Comparison, https://npm-compare.com/geoip-country,geoip-lite,maxmind 44. Choose the right geolocation product - MaxMind Support, https://support.maxmind.com/knowledge-base/articles/choose-the-right-maxmind-geolocation-product 45. geoip-lite/node-geoip: Native NodeJS implementation of MaxMind's GeoIP API -- works in node 0.6.3 and above, ask me about other versions - GitHub, https://github.com/geoip-lite/node-geoip 46. Documentation - Leaflet - a JavaScript library for interactive maps, https://leafletjs.com/reference.html 47. Plugins - Leaflet - a JavaScript library for interactive maps, https://leafletjs.com/plugins.html 48. Leaflet - a JavaScript library for interactive maps, https://leafletjs.com/ 49. How to format Discord Webhook or how to use POST - CubeCoders Support, https://discourse.cubecoders.com/t/how-to-format-discord-webhook-or-how-to-use-post/1572 50. I analyzed the frequency and subchunk positions of all the blocks in 1.17 - Reddit, https://www.reddit.com/r/technicalminecraft/comments/o969tq/i_analyzed_the_frequency_and_subchunk_positions/ 51. The core implementation of worlds for prismarine - GitHub, https://github.com/PrismarineJS/prismarine-world 52. Download Minecraft worlds, extend server's render distance. 1.12.2 - 1.21 - GitHub, https://github.com/mircokroon/minecraft-world-downloader 53. Proxies with Socks5-Client not working · Issue #1834 · PrismarineJS/mineflayer - GitHub, https://github.com/PrismarineJS/mineflayer/issues/1834 54. Can't use socks5 proxy · Issue #3457 · PrismarineJS/mineflayer - GitHub, https://github.com/PrismarineJS/mineflayer/issues/3457 55. PrismarineJS mineflayer-pathfinder - Bot ignores movement rules for first few seconds of goal - Stack Overflow, https://stackoverflow.com/questions/79788739/prismarinejs-mineflayer-pathfinder-bot-ignores-movement-rules-for-first-few-se 56. Bot getting kicked by antibot plugin. · Issue #1091 · PrismarineJS/mineflayer - GitHub, https://github.com/PrismarineJS/mineflayer/issues/1091 57. bypass a botfilter 1.12.2 · Issue #1299 · PrismarineJS/mineflayer - GitHub, https://github.com/PrismarineJS/mineflayer/issues/1299 58. AntiSpammer - Paper Plugin - PaperMC Hangar, https://hangar.papermc.io/bodmedia/AntiSpammer