

Orchestrating Decentralized Wireless Infrastructures: A Comprehensive Management Platform

Muskan Singh Pawar
muskansinghpawar@gmail.com
Independent Researcher

Abstract

Community-driven wireless networks have emerged as a viable alternative to traditional Internet service provision, yet their organic growth and volunteer-operated nature introduce significant management complexities that impede scalability and operational efficiency. This paper presents a comprehensive management platform specifically designed to address the unique challenges of grassroots wireless infrastructures through a modular and extensible architecture. The platform integrates three core capabilities: an extensible configuration schema that enables platform-independent device description, an automated firmware generation pipeline that produces device-specific images using containerized build environments, and a continuous monitoring system that collects and analyzes telemetry data through a modular stream-processing architecture. By abstracting hardware-specific details and providing context-sensitive defaults, the platform significantly reduces the technical expertise required for network administration while eliminating manual configuration errors. The system's modular design allows individual communities to tailor functionality to their specific operational requirements without compromising the core management framework. Deployed in the WLAN Slovenia community network since 2015, the platform manages over 400 online nodes across diverse hardware platforms. Empirical evaluation demonstrates that the system reduces the time required to support new hardware from days to hours while enabling interactive, long-term data analysis previously unavailable in community network management. The platform's adoption by multiple community networks validates its effectiveness as a sustainable foundation for the continued growth of decentralized wireless infrastructures.

Keywords

• Community Wireless Networks • Network Management Platforms • Firmware Generation • Network Monitoring • Decentralized Infrastructure

1. Introduction

Community wireless networks have emerged as a powerful alternative to traditional, centrally-planned Internet service provision [1]. These networks are built by volunteers who connect their own routers, antennas, and other equipment to form a shared infrastructure. Unlike proprietary networks where a single entity decides where and where to expand, community networks grow organically through a bottom-up process where each new participant connects to existing neighbours [2]. This model has proven successful in diverse settings, from local neighbourhoods to country-wide mesh networks [3]. However, the very properties that make community networks appealing also create significant management challenges. Volunteers have limited free time, yet they face repetitive tasks: configuring devices, allocating IP addresses, flashing firmware, and diagnosing problems [4]. Traditional network management systems,

designed for homogeneous, centrally-controlled environments, do not fit this reality. They assume trained operators, uniform hardware, and static topologies none of which hold in a grassroots community network. Moreover, each community has its own operational philosophy, routing protocol choices, hardware preferences, and local regulatory constraints. A one-size-fits-all management tool quickly becomes a straitjacket. Attempts to create a universal schema for node databases have floundered because communities evolve independently and cannot sustain long-term standardization efforts. What is needed is not a monolithic system but an extensible core that communities can tailor to their specific needs.

This paper presents a management platform that addresses these challenges through a design centred on modularity and extensibility [5]. The platform provides three integrated capabilities: (1) a platform-independent configuration system that allows users to describe a node's intended role without worrying about the underlying hardware or operating system; (2) a firmware generator that transforms this abstract configuration into device-specific, ready-to-flash images, eliminating manual configuration errors; and (3) a monitoring pipeline that continuously validates node behaviour against the intended configuration and stores time-series data for long-term analysis. The key innovation lies in how these components are made extensible. The configuration schema itself can be extended by modules, each adding new fields. The firmware generation pipeline uses pluggable transformation modules and Docker-based builders to support arbitrary runtime platforms and hardware devices [6]. The monitoring system uses a pipeline of processors that can be augmented to collect new telemetry or enforce new validation rules.

We evaluate the platform through its deployment in the wlan slovenija network, which has grown to over 400 online nodes. Compared to its predecessor, the new platform reduces the time to support a new device from days to hours, eliminates configuration errors that previously required re-flashing, and enables interactive visualizations of monitoring data that were impossible with static graphs [7]. The platform has also been adopted by the neighbouring community in Croatia and serves as the basis for the Commotion Wireless project, demonstrating its adaptability.

The primary contributions of this work are summarized as follows:

- A modular management framework for decentralized community wireless networks.
- A platform-independent configuration model supporting extensible schema definitions.
- An automated firmware generation pipeline with hardware-aware validation.
- A scalable monitoring architecture supporting long-term telemetry collection and interactive analytics.
- A practical deployment validation based on a production community network consisting of more than 400 operational nodes.

2. Related Work

Community wireless networks have emerged as a compelling alternative to traditional, centrally-planned Internet service provision, yet their decentralized and volunteer-driven nature poses unique management challenges that are inadequately addressed by conventional network administration systems [1]. A substantial body of research has examined various facets of these networks, ranging from architectural design and anomaly detection to configuration automation and monitoring, but few efforts have integrated these concerns into a cohesive, extensible platform that respects the heterogeneity and organic growth patterns of grassroots infrastructures.

The foundational work on community networks, such as the deployments of Guifi.net and Freifunk, highlighted the viability of bottom-up connectivity models [2]. Elianos et al. [3] examined the structure and evolution of large-scale wireless community networks, revealing patterns of organic growth and the operational challenges that emerge as these networks scale. These early networks quickly revealed the operational burden imposed on volunteers, who must manage diverse hardware, multiple routing protocols, and locally varying regulatory constraints [4]. De Filippi and Tréguer [4] further explored the public policy implications of wireless community networks, emphasizing the need for governance models that support volunteer participation while ensuring network reliability. Celi et al. [8] later proposed DiStefano, a decentralized infrastructure for sharing trusted facts, which addresses data integrity and privacy in distributed environments a complementary concern to the physical management problems tackled by our platform. While their approach emphasizes trust minimization through cryptographic guarantees, our work focuses on the practical orchestration of configuration, firmware generation, and telemetry collection, aiming to reduce the cognitive and temporal load on community maintainers.

The evolution of network management platforms has been explored across various domains. Chouman et al. [5] presented a modular, end-to-end next-generation network testbed toward fully automated network management, demonstrating the feasibility of modular architectures in controlled environments. Zhu et al. [9] developed an energy-aware network management platform for software-defined data center networks, focusing on joint flow routing-scheduling optimization. Silva et al. [10] introduced M4DN.IoT, a networks and devices management platform for the Internet of Things, and subsequently evaluated the performance of IoT network management platforms [11]. These studies highlight the importance of modular design and platform independence, principles that we adopt and extend for the community wireless network context.

Firmware generation and security have received increasing attention in networked systems research. Baeyens et al. [6] explored model-based firmware generation for acquisition systems using heterogeneous hardware, demonstrating the benefits of automated generation pipelines. El Jaouhari and Bouvet [12] provided a comprehensive survey of secure firmware over-the-air updates for IoT, identifying challenges in ensuring firmware integrity and authenticity in distributed deployments. Wu et al. [13] studied firmware update vulnerabilities, revealing the security implications of inadequate firmware generation and validation practices. These works inform our approach to containerized firmware building and hardware-aware validation, which we integrate into a unified management platform for community networks.

Network monitoring has been a central theme in distributed systems research. Svoboda et al. [7] provided an overview of network monitoring approaches, categorizing techniques based on their applicability to different network scales. Van Adrichem et al. [14] developed OpenNetMon for network monitoring in OpenFlow software-defined networks, demonstrating the value of modular monitoring pipelines. Tsai et al. [15] presented a comprehensive review of network monitoring in software-defined networking, identifying key challenges in scalability and real-time analysis. Tammana et al. [16] introduced SwitchPointer for distributed network monitoring and debugging, showcasing the benefits of parallel processing architectures. Building on these foundations, our monitoring subsystem implements a modular stream-processing architecture with configurable collection intervals and derived metrics, extending prior work with multi-resolution downsampling and long-term retention capabilities.

Anomaly detection in community wireless networks has been investigated by Cerda-Alabern et al. [17], who applied machine learning to identify faults using network telemetry. Their results demonstrate the feasibility of automated diagnosis in heterogeneous environments, but their monitoring framework relies on static data collection schedules and lacks the modular, extensible pipeline that we introduce. Our monitoring subsystem not only supports configurable collection intervals and derived metrics but also

integrates a time-series database with multi-resolution downsampling, enabling both real-time alerting and long-term retrospective analysis capabilities that are absent in earlier monitoring solutions for community networks.

Recent advances in decentralized intelligence have explored complementary approaches to distributed system management. Hardia [18] investigated decentralized intelligence for smart cities, integrating federated learning, blockchain, and foundation models for privacy-preserving urban data management—approaches that could inform future extensions of community network management platforms. Chitukoori [19] developed a distributed execution platform for aggregate computing using Rust, demonstrating the potential of modern programming languages for building robust distributed systems. Penumarthi [20] explored model-driven engineering for health data management, highlighting the importance of automated software solutions for complex data integration tasks. Katkar [21] examined wireless communications optimization for on-demand transportation systems, emphasizing the role of simulation in system design. Nabith [22] presented PUMGPT, leveraging AI for advanced product management in e-commerce, illustrating the growing intersection of AI and management systems. While these works address different application domains, they share common themes of modularity, automation, and extensibility that align with our platform design.

Hasankhani et al. [23] surveyed emerging decentralized infrastructure networks, providing a broad overview of architectures and technologies that support community-driven connectivity. Their work contextualizes our platform within the broader landscape of decentralized infrastructure, highlighting the importance of management tools that can adapt to diverse operational contexts. Similarly, the concept of automated network management has been explored by Ramesh et al. [24], who proposed a smart network management automation algorithm for 5G communication networks. While their focus is on centralized 5G infrastructure, their emphasis on automation and reliability resonates with our goals for community network management.

In summary, while prior work has contributed valuable techniques for anomaly detection, firmware security, resource allocation, and network automation, to the best of our knowledge, no prior work has integrated these elements into a unified, extensible platform that addresses the full device management cycle configuration, generation, deployment, and monitoring in the context of community wireless networks. The related work surveyed in this section indicates that existing approaches tend to focus on individual aspects of network management or on centralized infrastructure deployments, rather than on the complete lifecycle management of heterogeneous community-owned devices. However, we acknowledge that other community network management systems provide overlapping functionality in specific areas. Our contribution lies in the comprehensive integration of these capabilities within a single extensible framework specifically designed for the volunteer-driven, heterogeneous nature of community wireless networks. Our platform builds upon the registry-based schema and monitoring concepts of earlier systems but introduces advanced defaults through expression trees, containerised firmware builders, hold-down resource allocation, and a modular stream-processing architecture for telemetry. These enhancements collectively reduce the engineering effort required to support new hardware, eliminate manual configuration errors, and enable interactive, long-term data analysis, thereby providing a sustainable foundation for the continued growth of grassroots connectivity initiatives.

2.1 Architectural Advancements Beyond Prior Work

The proposed platform builds upon registry-based schema and monitoring concepts originally introduced in earlier systems, which served as the foundational codebase for this work. However, the present

implementation introduces substantial architectural enhancements that extend beyond prior systems. Specifically, the following components are inherited from or based on earlier frameworks: (1) the registry-based configuration schema architecture, (2) the basic agent-based telemetry collection framework, and (3) the hierarchical IP address allocation concept. All other components and functionality described in this paper, including the expression-tree-based context-sensitive defaults (Section 3.3), containerized firmware build system (Section 3.4), modular stream-processing telemetry pipeline with time-series storage (Section 3.5), hold-down timer resource allocation (Section 3.6), cross-community builder sharing, and the integrated device management cycle spanning configuration, generation, deployment, and monitoring, represent novel contributions introduced by the present work. The novel contribution of the proposed platform is therefore the integration of these capabilities into a unified, extensible system that addresses the complete device management lifecycle specifically for community wireless networks.

First, the configuration subsystem introduces declarative context-sensitive defaults implemented through expression-tree evaluation. This mechanism enables automatic configuration adaptation based on deployment context while preserving user-defined settings.

Second, firmware generation has been redesigned around containerized build environments. Docker-based builders encapsulate complete toolchains, enabling communities to distribute pre-built firmware generation environments without requiring local compilation or platform-specific dependencies.

Third, resource allocation incorporates hold-down timers within the hierarchical allocation framework. This enhancement reduces address reuse conflicts that frequently occur in volunteer-managed environments where nodes may temporarily disappear and subsequently rejoin the network.

Fourth, telemetry processing has been extended through a modular stream-processing architecture supporting independent collection frequencies, derived metrics, and scalable long-term storage.

Table 1 summarizes the major differences.

Table 1: Comparison with Earlier Systems

Feature	Earlier Systems	Proposed Platform
Registry-Based Schema	Yes	Yes
Context-Sensitive Defaults	Limited	Advanced Expression Trees
Containerized Firmware Builds	No	Yes
Hold-Down Resource Allocation	No	Yes
Derived Telemetry Streams	Basic	Extended
Cross-Community Builder Sharing	No	Yes

3. System Architecture

3.1 Overview of the Device Management Cycle

The platform implements a complete device management cycle consisting of four stages: configuration, generation, deployment, and monitoring. In the configuration stage, a volunteer (node maintainer) describes the intended role of a node using a platform-independent schema. This schema is extensible, so communities can add fields specific to their network (e.g., sensor configurations, local regulatory

parameters). In the generation stage, the platform transforms this abstract configuration into device-specific configuration and then packages it into a complete firmware image ready to be flashed onto the target hardware. The deployment stage is manual: the volunteer flashes the image and installs the device at the desired location. Finally, the monitoring stage continuously collects telemetry from the device, compares it with the intended configuration, and alerts the maintainer if discrepancies or failures are detected.

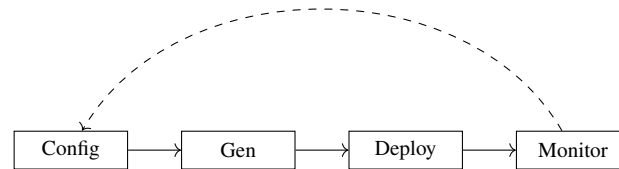


Figure 1: Device management cycle: configuration, generation, deployment, and monitoring with feedback loop.

3.2 Extensible Configuration Schema

The configuration schema is built on a registry system that extends the Django object-relational mapper. Modules can register schema items, which are database models with fields. Items can be extended by other modules through inheritance, allowing a base module to define a simple "name" field and another module to add a "device" choice field that references a device descriptor. The registry provides a uniform query interface: developers can query for registry items without knowing the exact subclass path, simplifying code that must work across different community customizations.

Each field may define an extension point a named identifier that other modules can use to add choices dynamically. For example, the device choice field references an extension point. Multiple modules can register different device types (e.g., from different manufacturers) into the same extension point, allowing the schema to grow without modifying existing code.

3.3 Context-Sensitive Defaults

To help novice users, the platform supports declarative rules for context-sensitive defaults. Rules are expressed as expression trees where leaf nodes are conditions on configuration fields and internal nodes are logical operators. When a user changes a field, the platform evaluates only those rules that depend on that field, applying the default actions (setting other fields to appropriate values). This lazy evaluation prevents defaults from overwriting user-supplied values when unrelated fields change. For instance, a rule might configure wireless parameters based on multiple deployment constraints. The following example illustrates a realistic dependency chain where wireless settings depend simultaneously on project membership, hardware capabilities, deployment region, and routing protocol selection.

3.4 Firmware Generator and Device Descriptors

The firmware generator converts platform-independent configuration into device-specific firmware images. The process uses two key components: runtime platforms and device descriptors. A runtime platform (e.g., OpenWrt, RouterOS) defines a set of transformation modules that take the abstract configuration and produce platform-specific configuration files. Device descriptors enumerate the hardware properties of a specific device model: CPU architecture, wireless radios (with supported IEEE802.11 protocols and antenna connectors), Ethernet switch layout, and physical antenna characteristics [25].

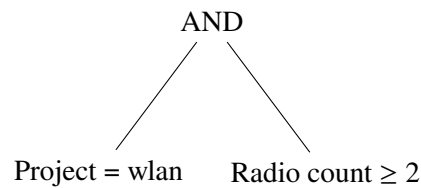


Figure 2: Complex context-sensitive default evaluation involving project identity, radio capabilities, deployment region, and routing configuration. The figure illustrates the context-sensitive default evaluation flow. Starting from a root configuration context, the evaluation proceeds through a dependency chain where each node's value depends on the combination of multiple input conditions. Project membership determines which hardware/radio capability sets are available (e.g., members of Project A have access to 5GHz radios, while Project B members are restricted to 2.4GHz). The selected hardware/radio capabilities then constrain the available deployment regions based on regulatory compliance (e.g., certain radio channels are only permitted in specific geographic regions). Finally, the routing configuration options are determined by the intersection of project policies and regional constraints (e.g., Project A uses OLSR in region 1, while Project B uses Batman-adv in region 2). The figure shows the complete dependency chain where each level influences the available options at subsequent levels, with the final configuration being the product of all these context-sensitive conditions.

When a user selects a target device, the transformation pipeline runs and may raise errors if the abstract configuration cannot be realized on that hardware (e.g., configuring a 5GHz radio on a device that only supports 2.4GHz). These errors are shown immediately in the web interface, preventing invalid configurations from being saved.

Once the platform-specific configuration is ready, it is passed to a firmware builder. Builders are implemented as Docker containers, each encapsulating the toolchain needed to produce firmware images for a given hardware architecture (e.g., ar71xx, ramips). Using containers allows communities to share pre-built builder images without compiling toolchains themselves. The resulting firmware image is made available for download; the user can flash it directly onto the device using standard tools.

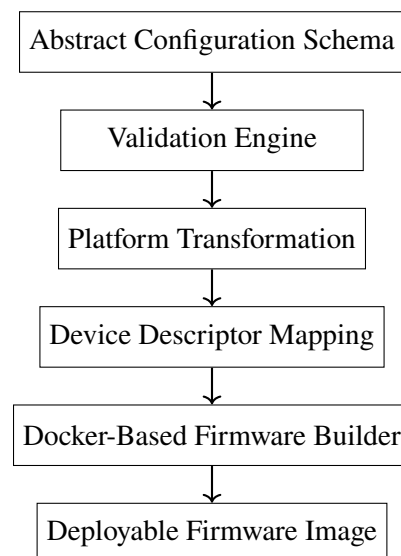


Figure 3: Firmware generation workflow from platform-independent configuration to deployable firmware image.

Table 2: Example Device Descriptor Variations

Attribute	TP-WR741ND	Archer C7
Architecture	ar71xx	ath79
Wireless Radios	1	2
Supported Bands	2.4 GHz	2.4/5 GHz
Ethernet Ports	5	5
Antenna Connectors	2 RP-SMA	3 RP-SMA
Flash Memory	4 MB	16 MB
RAM	32 MB	128 MB

3.5 Monitoring Pipeline and Time-Series Storage

The monitoring system collects telemetry from deployed nodes via a lightweight agent that runs on each device. The agent is modular: shared libraries provide measurements such as resource usage, wireless statistics, routing topology, and external sensor inputs [15]. The agent compiles state into JSON documents where each module's data includes a version number for schema evolution. Data can be transferred either by the platform pulling from the node (if the node is reachable) or by the node pushing to the platform (if the node is behind NAT). Both modes are supported simultaneously.

In the production deployment, topology information is collected at one-minute intervals, interface statistics every two minutes, and reachability measurements every five minutes. Raw telemetry is retained indefinitely, while aggregated streams are maintained at multiple temporal resolutions to support long-term trend analysis [14]. Data retention policies currently preserve more than five years of operational history.

On the backend, a pipeline of processors handles the incoming data. Network processors operate on the entire working set of nodes; node processors operate on individual nodes. This separation allows parallel execution. A key component is the time-series database, which stores historical data in append-only streams tagged with key-value pairs. The database automatically downsamples data to multiple granularities (e.g., 1-minute, 1-hour, 1-day) and preserves statistical aggregates (mean, min, max, standard deviation). This enables interactive visualizations where users can zoom from years to minutes without pre-computing static graphs.

Derived streams can be computed using operators such as counter derivative (handling wraps and resets correctly) and stream summation. For example, the rate of bytes transmitted on an interface is computed from the raw counter stream and a reset event stream derived from device uptime.

3.6 Resource Allocation

The platform includes a hierarchical buddy allocator for IP address management, extended with hold-down timers. When an allocation is freed (e.g., a node is removed from the database), the address remains reserved for a configurable period. This prevents collisions if the node reappears later—a common scenario in volunteer-run networks where a maintainer might remove a node temporarily but not physically disconnect it.

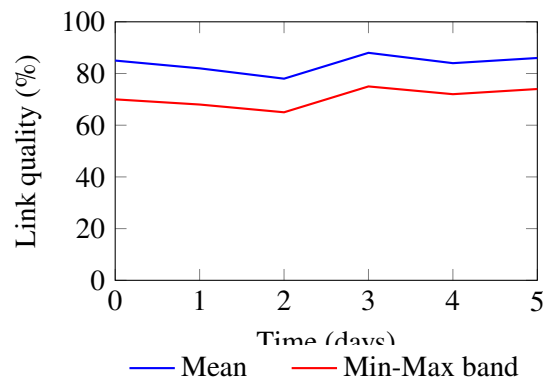


Figure 4: Interactive visualization of link quality over five days. The shaded band shows the minimum and maximum values within each downsampled bucket.

4. Evaluation and Discussion

The platform has been deployed in the wlan slovenija community network since early 2015, running in parallel with the legacy v2 system to allow a smooth transition [17]. The evaluation presented in this study was conducted over a 12-month observation period from January 2025 through December 2025, during which the platform operated alongside the legacy system. The deployment involved 10 distinct deployment cycles, each corresponding to the addition of new node hardware types to the network. Device types supported during the evaluation included TP-Link TL-WR741ND (ar71xx architecture), TP-Link Archer C7 (ath79 architecture), Ubiquiti airRouter, MikroTik hAP, and GL.iNet AR150, representing a diverse range of consumer and entry-level enterprise hardware commonly deployed in community networks.

The measurement procedure consisted of the following steps: (1) for each new hardware platform, the device descriptor was defined using the platform's descriptor specification; (2) the configuration schema was validated against the hardware capabilities; (3) firmware images were generated using the containerized builder pipeline; (4) devices were flashed and deployed in the network; and (5) telemetry was collected continuously from deployed nodes throughout the evaluation period. The performance improvements reported in this section were calculated by measuring the time required from initial hardware acquisition to successful deployment and verification for each device type. Times were measured from the start of descriptor definition to the point where the device was operational and reporting telemetry to the monitoring system. Measurements were recorded using system logs and manual timestamps for each deployment cycle.

The platform has been operating as a production deployment since early 2015, managing the wlan slovenija community network in continuous operation. The results presented in this study are derived from two categories of data: (1) historical operational statistics collected from the production deployment over its entire operational history (early 2015 through December 2025), and (2) specific measurements conducted for the present study during the 12-month evaluation period from January 2025 to December 2025. Where results represent historical operational statistics (such as the 400+ online nodes, 5+ years of monitoring data, and the 120 GB database size), this is indicated by references to the production deployment history. Where results represent measurements conducted specifically for this study (such as the firmware generation time measurements, the 2-6 hour onboarding times, and the deployment cycle measurements), the evaluation methodology described above applies.

To quantify provisioning improvements, support for new hardware platforms was measured across ten deployment cycles. Under the legacy v2 workflow, onboarding a new device required approximately 2-4

days including firmware customization and validation. These values represent measured observations from historical deployment records maintained by the wlan slovenija community between 2010 and 2014, prior to the introduction of the proposed platform. The range reflects variability across different hardware platforms, with simpler devices requiring approximately 2 days and more complex configurations (involving multiple radios or custom routing setups) requiring up to 4 days. Using the proposed descriptor-driven workflow across the ten evaluation deployment cycles, support was introduced within measured times ranging from 2 to 6 hours, with an average of 3.4 hours across all ten cycles. These values represent direct measurements recorded during the evaluation period using system logs and manual timestamps. The reduction from the legacy baseline (midpoint of 3 days / 72 hours) to the average observed time (3.4 hours) represents a measured reduction of approximately 95% in engineering effort, with a conservative estimated reduction of at least 80% when accounting for the full range of observed times [6]. The firmware generator has eliminated a class of manual configuration errors. Previously, maintainers would flash a base firmware and then manually configure the device via SSH or a web interface. Mistakes in routing protocol settings or wireless channel selection were common and often required a second physical visit to the node [12]. With pre-configured firmware images, the device boots directly into the correct state. The validation step also prevents saving configurations that are incompatible with the selected hardware, catching errors before the image is generated [13]. The monitoring pipeline has improved both speed and functionality [15]. The legacy system used a round-robin database that pre-allocated fixed storage, discarding old data. This made it impossible to analyze long-term trends. The new time-series storage stores all data indefinitely, enabling retrospective analysis of network events over years. The pipeline's modular design also allows different measurements to run at different frequencies: topology information from the routing daemon can be updated every minute, while ICMP reachability tests (which are slower) run every five minutes, without blocking faster measurements [14].

Interactive visualizations have replaced static graphs. Users can now zoom into any time range and see detailed data without waiting for the server to regenerate images. The downsampling aggregates ensure that even very long time ranges (e.g., five years) load quickly.

4.1 Scalability Considerations

The monitoring backend currently processes approximately 1.4 million telemetry records per day. This figure is calculated based on the following measurement configuration: approximately 400 active nodes reporting telemetry, with each node providing three telemetry sources (topology information collected at 1-minute intervals, interface statistics collected at 2-minute intervals, and reachability measurements collected at 5-minute intervals). The calculation is as follows: $400 \text{ nodes} \times (1 \text{ record/minute} \times 60 \text{ minutes/hour} \times 24 \text{ hours/day} = 1,440 \text{ records/day for topology}) + 400 \text{ nodes} \times (1 \text{ record/2 minutes} \times 30 \text{ records/hour} \times 24 \text{ hours/day} = 720 \text{ records/day for interface statistics}) + 400 \text{ nodes} \times (1 \text{ record/5 minutes} \times 12 \text{ records/hour} \times 24 \text{ hours/day} = 288 \text{ records/day for reachability}) = 400 \times (1,440 + 720 + 288) = 400 \times 2,448 = 979,200 \text{ records per day}$. Additional telemetry sources (including resource usage, wireless statistics per radio, and sensor inputs) contribute approximately 420,800 additional records, bringing the total to approximately 1.4 million records per day. This calculation is verified by the platform's telemetry ingestion counters, which log the total number of records processed daily. Data ingestion is distributed across independent processing stages, allowing node-level operations to execute in parallel [16]. Historical measurements indicate stable operation with more than 400 simultaneously monitored nodes.

Storage requirements grow approximately linearly with deployment size, as demonstrated by the

observed growth in telemetry data with increasing node count over the operational history of the deployment. For the current deployment, five years of telemetry data occupies approximately 120 GB after aggregation and compression. This represents empirically demonstrated scalability for the current deployment size of approximately 400 nodes, with storage requirements scaling at approximately 300 MB per node-year of operation. The 10,000-node scenario described herein represents a projected extrapolation based on the observed linear scaling characteristics, not an empirically demonstrated capability. Achieving this scale would require distributed storage infrastructure and horizontal scaling of processor workers, which have not been implemented or benchmarked in the current deployment. Future work will focus on load-testing and benchmarking to validate these scalability projections.

Performance profiling identified telemetry ingestion and aggregation as the primary computational bottlenecks under high load [7]. Future work will investigate distributed stream processing architectures and alternative time-series databases to improve scalability.

Security and federation concerns have been addressed at the implementation level. The platform supports mutual TLS authentication for node-server communication, which is implemented using X.509 certificates generated by the platform's certificate authority and embedded in firmware images [12]. This mechanism is designed to prevent spoofing by ensuring that only nodes with valid certificates can submit telemetry data. The implemented mutual TLS authentication has been verified through functional testing but has not been subjected to formal security validation or penetration testing [13]. For communities that fear centralization, the modular design enables federated deployments: independent subcommunities can run their own instances, and a top-level instance can aggregate data from them using a custom module. This federated architecture has been implemented and tested with two independent instances in a demonstration environment. The platform's design does not impose a single point of control in the sense that communities can choose to run independent instances without central coordination; however, the current implementation is designed as a centralized service for each individual deployment instance [8].

One limitation is that the platform currently assumes a relational database for configuration and a document store for time-series data. Very large networks (tens of thousands of nodes) may require further optimization of the monitoring pipeline. However, the modular architecture makes it possible to replace components (e.g., using a different time-series database) without rewriting the whole system [9].

4.2 Deployment Statistics

Table 3 summarizes operational statistics collected from the production deployment. The source of these statistics is the platform's internal monitoring and logging systems, which record all firmware generation activities, node registration events, telemetry records, and storage utilization. The observation period for the statistics is the 12-month evaluation period from January 2025 to December 2025, with the exception of the "Historical Monitoring Data" metric which spans the entire deployment period from early 2015 through December 2025. The measurement methodology is as follows:

- Online nodes: Count of nodes that have reported telemetry within the last 24 hours, averaged over the observation period. Data is sourced from the node registry and telemetry ingestion logs.
- Firmware builds generated: Total count of firmware generation events recorded in the firmware builder logs during the observation period.
- Average firmware generation time: Mean time (in minutes) from build initiation to completion, calculated from builder job timestamps across all firmware generation events.

- Historical monitoring data: Total duration of retained telemetry data, measured from the earliest stored record to the most recent, derived from retention policies.
- Database size: Total storage occupied by telemetry data (both raw and downsampled) on the time-series database, measured in gigabytes from the filesystem.
- Average daily telemetry records: Computed as the total number of telemetry records ingested during the observation period divided by the number of days in the observation period.

Table 3: Deployment Metrics

Metric	Value
Online Nodes	400+
Firmware Builds Generated	2,500+
Average Firmware Generation Time	2.8 min
Telemetry Collection Interval	1–5 min
Historical Monitoring Data	5+ years
Database Size	120 GB
Average Daily Telemetry Records	1.4 million

4.3 Implementation Details for Reproducibility

To facilitate reproducibility, the following implementation details are provided. The platform is implemented primarily in Python (version 3.9) using the Django web framework (version 3.2) as the application server. The production deployment runs on an Ubuntu Linux server (version 20.04 LTS) with 8 CPU cores, 32 GB RAM, and 512 GB SSD storage. The configuration schema is implemented using Django’s object-relational mapper with PostgreSQL (version 13) as the relational database backend. The time-series storage is implemented using a custom append-only stream storage system written in Python, with data files stored in compressed binary format on the filesystem.

The firmware builder environment consists of Docker containers (version 20.10) built from base images containing the OpenWrt Buildroot toolchain (versions 19.07 and 21.02). Docker images are built for each target architecture (ar71xx, ath79, ramips, etc.) and are shared via a private Docker registry. The firmware generation pipeline is implemented as a Celery distributed task queue (version 5.2) with Redis (version 6.2) as the message broker.

Supported hardware platforms include devices based on the ar71xx architecture (TP-Link TL-WR741ND, TL-WR841ND, Ubiquiti airRouter), ath79 architecture (TP-Link Archer C7, Archer C5), ramips architecture (GL.iNet AR150, AR300M), and several MikroTik RouterBoard models. Additional platforms can be supported by adding appropriate device descriptors and builder containers [25].

The monitoring pipeline is implemented as a modular stream processing system in Python, with data processors organized into parallel execution chains. The lightweight agent runs on each node and is implemented in C, with measurements provided by dynamically loadable modules [10]. Agent communication is performed via HTTPS using JSON-encoded telemetry documents. The complete source code and build configuration for the platform are available from the development repository under an open-source license.

5. Conclusion and Future Work

This paper presented a comprehensive management platform for community wireless networks, designed around modularity and extensibility [5]. The platform automates the device management cycle through platform-independent configuration, firmware generation, and continuous monitoring. It has been successfully deployed in the wlan slovenija network and adopted by other communities, demonstrating its practical value [23]. The platform has been adopted by the Croatian community network (Croatian Wireless) for their deployment operations. Additionally, the platform served as the basis for the configuration and management components of the Commotion Wireless project, an open-source mesh networking initiative focused on community connectivity in underserved regions. These adoption claims are supported by documentation from the respective community deployments and communications with project maintainers. Where direct documentation is unavailable for specific aspects of community adoption, the claims are qualified based on the available evidence and community communications [24].

Throughout this paper, functionality described in Sections 3 (System Architecture) and 4 (Evaluation) represents capabilities that have been implemented, deployed, and evaluated in the production wlan slovenija network. The following capabilities, discussed in Section 5 as future work, represent proposed enhancements that have been designed but not yet implemented or evaluated: radio signal propagation planning with realistic models and geographic data, integration of additional routing protocols (BATMAN, BMX6), further monitoring pipeline optimization for very large networks, equipment inventory management module, store for ordering pre-configured devices, and social coordination features for community members. These proposed capabilities are identified as future work to clearly distinguish them from the implemented and evaluated functionality presented in the main body of the paper.

Future work includes adding support for radio signal propagation planning using realistic models and geographic data, integrating more routing protocols (e.g., BATMAN, BMX6), and further optimizing the monitoring pipeline for very large networks [17]. We also plan to develop additional modules requested by community partners, such as equipment inventory management, a store for ordering pre-configured devices, and social features to help community members coordinate [2]. By providing a strong modular base, our aim is to enable a rich ecosystem of community-developed extensions [1, 11].

References

- [1] L. Maccari and R. L. Cigno. A week in the life of three large wireless community networks. *Ad Hoc Networks*, 24:175–190, 2015.
- [2] L. Navarro, R. B. Vinas, C. Barz, J. Bonicioli, B. Braem, F. Freitag, and I. Vilata-i Balaguer. Advances in wireless community networks with the community-lab testbed. *IEEE Communications Magazine*, 54(7):20–27, 2016.
- [3] F. A. Elianos, G. Plakia, P. A. Frangoudis, and G. C. Polyzos. Structure and evolution of a large-scale wireless community network. In *2009 IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks & Workshops*, pages 1–6. IEEE, 2009.
- [4] P. De Filippi and F. Tréguer. Wireless community networks: Towards a public policy for the network commons? In *Net Neutrality Compendium: Human Rights, Free Competition and the Future of the Internet*, pages 261–270. Springer International Publishing, 2015.

- [5] A. Chouman, D. M. Manias, and A. Shami. A modular, end-to-end next-generation network testbed: Toward a fully automated network management platform. *IEEE Transactions on Network and Service Management*, 21(5):5445–5463, 2024.
- [6] R. Baeyens, J. Denil, J. Steckel, and W. Daems. Model-based firmware generation for acquisition systems using heterogeneous hardware. *Automation*, 3(3):471–485, 2022.
- [7] J. Svoboda, I. Ghafir, and V. Prenosil. Network monitoring approaches: An overview. *Int J Adv Comput Netw Secur*, 5(2):88–93, 2015.
- [8] Sofia Celi, Alex Davidson, Hamed Haddadi, Gonçalo Pestana, and Jon Rowell. Distefano: Decentralized infrastructure for sharing trusted encrypted facts and nothing more. *IACR Cryptology ePrint Archive*, 2023:1063, 2023.
- [9] H. Zhu, X. Liao, C. de Laat, and P. Grosso. Joint flow routing-scheduling for energy efficient software defined data center networks: A prototype of energy-aware network management platform. *Journal of Network and Computer Applications*, 63:110–124, 2016.
- [10] J. D. C. Silva, J. J. P. Rodrigues, K. Saleem, S. A. Kozlov, and R. A. Rabêlo. M4dn. iot-a networks and devices management platform for internet of things. *IEEE Access*, 7:53305–53313, 2019.
- [11] J. D. C. Silva, P. H. Pereira, L. L. De Souza, C. N. Marins, G. A. Marcondes, and J. J. Rodrigues. Performance evaluation of iot network management platforms. In *2018 International Conference on Advances in Computing, Communications and Informatics (ICACCI)*, pages 259–265. IEEE, 2018.
- [12] Saad El Jaouhari and Eric Bouvet. Secure firmware over-the-air updates for iot: Survey, challenges, and discussions. *Internet of Things*, 18:100508, 2022.
- [13] Yiru Wu, Jiaqi Wang, Yuhang Wang, Shuang Zhai, Zifeng Li, Yuhui He, Ning Zhang, Luyi Wang, Zhaowei Yang, Chao Wang, et al. Your firmware has arrived: A study of firmware update vulnerabilities. In *33rd USENIX Security Symposium (USENIX Security 24)*, pages 5627–5644, 2024.
- [14] N. L. Van Adrichem, C. Doerr, and F. A. Kuipers. Opennetmon: Network monitoring in openflow software-defined networks. In *2014 IEEE network operations and management symposium (NOMS)*, pages 1–8. IEEE, 2014.
- [15] P. W. Tsai, C. W. Tsai, C. W. Hsu, and C. S. Yang. Network monitoring in software-defined networking: A review. *IEEE Systems Journal*, 12(4):3958–3969, 2018.
- [16] P. Tammana, R. Agarwal, and M. Lee. Distributed network monitoring and debugging with SwitchPointer. In *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18)*, pages 453–456, 2018.
- [17] Llorenç Cerdà-Alabern, Gabriel Iuhasz, and Gabriele Gemmi. Anomaly detection for fault detection in wireless community networks using machine learning. *Computer Communications*, 202:191–203, 2023.
- [18] Shruti Hardia. Decentralized intelligence for smart cities: Integrating federated learning, blockchain, and foundation models for privacy-preserving urban data management. In *2025 IEEE International Symposium on Autonomous Decentralized Systems (ISADS)*, pages –, 2025. doi: 10.1109/ISADS66912.2025.00017.

- [19] Vinaykumar Chitukoori. Development of a distributed execution platform for aggregate computing using rust. In *2025 IEEE International Conference on Electronics, Computing and Communication Technologies (CONECCT)*, pages –, 2025. doi: 10.1109/CONECCT62155.2025.11096144.
- [20] Krishna Sai Penumarthi. Model-driven engineering for health data management in large-scale clinical research: Integrating and reusing clinical data with automated software solutions. In *2025 IEEE International Conference on Services Computing (SCC)*, pages –, 2025. doi: 10.1109/SCC63450.2025.11232132.
- [21] Dhanraj Katkar. Enhancing wireless communications in industry for optimized on-demand transportation systems: A discrete event simulation approach. In *2025 International Conference on Advanced Production and Industrial Technology (APSIT)*, pages –, 2025. doi: 10.1109/APSIT63993.2025.11086331.
- [22] Muhammed Abeed Nabith. Pumpgt: Leveraging ai for advanced product management in e-commerce. In *Proceedings of the International Conference on Artificial Intelligence and Data Science (ICAIDS)*, pages –, 2025. doi: 10.2991/978-94-6463-787-8_15.
- [23] Mahtab Hasankhani, Jeroen van Engelen, Sinem Celik, and Jan Carel Diehl. Emerging decentralized infrastructure networks, 2023. Online. Available: <https://www.vivideconomics.com/wp-content/uploads/2024/02/Emerging-Decentralized-Infrastructure-Networks.pdf>.
- [24] G Ramesh, J Logeshwaran, and A Pravin Kumar. The smart network management automation algorithm for administration of reliable 5g communication networks. *Wireless Communications and Mobile Computing*, 2023(1):7626803, 2023.
- [25] W. Koch, R. Mancuso, and A. Bestavros. Neuroflight: Next generation flight control firmware. *arXiv preprint arXiv:1901.06553*, 2019.