APPLICATION PROTOTYPE FOR CENTRALIZED AND AUTOMATED NETWORK MANAGEMENT SYSTEM

Petar Kresoja*,
[0009-0008-3343-1540]
Marko Šarac,
[0000-0001-8241-2778]
Aleksa Vidaković,
[0009-0005-3527-011X]
Teodor Petrović,
[0009-0008-7186-2552]
Miloš Mravik
[0000-0001-5442-3998]

Singidunum University,
Belgrade, Serbia

Abstract:

Modern network infrastructures require efficient monitoring and management solutions to ensure operational continuity, security, and scalability. This paper presents a microservice-based application prototype designed to centralize and automate network management, incorporating real-time monitoring, remote device activation through Wake-on-LAN (WOL), and automated notifications for network anomalies. The proposed system comprises three core components: a backend service, a frontend interface, and a lightweight client agent deployed on monitored networks. The backend, built using Express.js and TypeScript, facilitates communication between components, while the frontend, developed with Vue 3, provides a user-friendly interface for managing network nodes. The client agent utilizes Axios, NodeARP, and Ping to perform network diagnostics. This research evaluates the system's architecture, deployment strategies, and performance implications, with a focus on optimizing disaster recovery time and reducing network downtime.

Keywords:

Network Monitoring, Centralized Management, Microservices, Wake-on-LAN, Automated Alerts, Disaster Recovery.

INTRODUCTION

The increasing complexity of modern network environments necessitates robust management solutions capable of real-time monitoring, remote accessibility, and automated control [1]. Traditional network management approaches rely on manual intervention and distributed tools, which can be inefficient in handling large-scale infrastructures [1]. The global shift toward remote and hybrid work, further catalyzed by the COVID-19 pandemic, has highlighted the need for centralized network management solutions that provide seamless oversight of connected devices, regardless of physical location [2].

A centralized network management system (CNMS) consolidates monitoring, control, and automation into a single platform, enabling IT administrators to oversee network health, track IP addresses, and automate device power management [3]. A key advantage of such a system is its ability to remotely manage server and workstation states using Wake-on-LAN (WOL) technology [3] [4].

Correspondence:
Petar Kresoja

e-mail:
pkresoja@singidunum.ac.rs

DOI: 10.15308/Sinteza-2025-144-149

This feature minimizes manual intervention by allowing administrators to power on critical systems remotely when required. Additionally, real-time network diagnostics and automated alerting mechanisms enhance response times to network anomalies, improving overall efficiency and security [1].

This paper introduces a microservice-based CNMS prototype, developed to address these challenges. The system consists of three primary components:

- **Master Application (Backend)**: A centralized service handling API requests, user authentication, and database operations.
- **Frontend Interface**: A web-based application that allows administrators to monitor and control network nodes via a user-friendly UI.
- **Edge Node Client**: A lightweight TypeScript application installed on network nodes to conduct status checks, detect device availability via ICMP (ping), retrieve MAC addresses, and issue WOL commands [3].

By implementing a modular microservices architecture, this system ensures scalability, fault tolerance, and interoperability with existing network infrastructures [2] [5]. The study explores the technical implementation, deployment strategies, and system performance, particularly focusing on minimizing disaster recovery time and enhancing network resilience [6] [7].

2. RELATED WORKS

The development of centralized network management systems aligns with ongoing research in real-time monitoring and automated control of networked infrastructures. Paper [1] provides a comprehensive overview of current and emerging practices in network monitoring, emphasizing the transition from reactive to proactive systems capable of early anomaly detection. When it comes to modern industrial environments, in the paper [2] authors discuss the increasing relevance of centralized solutions within the framework of Industry 4.0, where the integration of IoT devices requires scalable and efficient management platforms.

In the paper [3], the authors outline general IoT architecture and protocol challenges, reinforcing the need for modular and lightweight systems such as the one proposed in this paper. The use of Wake-on-LAN (WOL) for device activation is supported by foundational practices in network protection and control as described in [4]. The authors stress the role of automation in improving response time and reducing manual overhead.

Further, the implementation of microservices and fog computing principles is mentioned in the paper [5], where the authors highlight architectural advantages such as scalability and fault isolation, which are both essential for robust CNMS design. Finally, the system's disaster recovery capabilities are supported by research into communication architectures for disaster scenarios [6] and machine learning-based anomaly detection for resilience [7], once again underlining the importance of automated alerting and node recovery mechanisms in modern network management.

3. METHODOLOGY

The CNMS follows a microservices architecture to enable modularity and independent scalability of system components. Each service operates autonomously while interacting through well-defined RESTful APIs, ensuring minimal dependencies and fault isolation.

3.1. SYSTEM ARCHITECTURE DESIGN

As shown in Table 1, the architecture consists of:

1. **Backend Service**: Developed in Express.js with TypeScript, this service acts as the system's central hub, processing API requests, managing authentication, and interfacing with the database.
2. **Frontend Application**: Built using Vue 3 (Composition API), the frontend provides a responsive dashboard for network administrators, displaying real-time device status and offering control functionalities.
3. **Edge Node Client**: A lightweight TypeScript-based application utilizing Axios for API communication, NodeARP for MAC address retrieval, and Ping for online status checks.

The backend and frontend components are deployed within a containerized environment using Docker, ensuring portability and simplified orchestration. The edge node client operates as a standalone process on network devices, periodically sending status updates to the central server.

3.2. TECHNOLOGY STACK

Table 1. Technology Stack

Component	Technology Used
Backend	Express.js, TypeScript, REST API, Bun (v1.2.3)
Frontend	Vue 3 (Composition API), TypeScript
Edge Client	TypeScript, Axios, NodeARP, Ping
Database	MySQL (hosted on the master server for low-latency queries)
Deployment	Docker (Containerization), Systemd (for edge node service management)

3.3. FUNCTIONAL FEATURES

The application has been developed with a comprehensive feature set, emphasizing a modular architecture that facilitates seamless integration of new functionalities. This design approach ensures scalability, maintainability, and adaptability, allowing for the efficient adoption of emerging technologies and future enhancements without requiring significant structural modifications.

3.3.1. Real-time Network Monitoring

The edge client operates as an autonomous monitoring agent, periodically dispatching ICMP ping requests to all registered devices within the network to assess their availability and operational status. Upon receiving a response, the client evaluates whether the device is online or offline and logs the results. These status updates are then transmitted to the master application, where they are systematically processed and stored. Administrators can access this real-time data through the frontend interface, enabling them to monitor network node statuses, detect connectivity issues, and take appropriate actions when devices become unresponsive.

3.3.2. Wake-on-LAN (WOL) Automation

The system continuously monitors network devices and detects those that are offline. When an offline device is identified, it automatically generates and transmits Wake-on-LAN (WOL) packets to remotely initiate its power-up sequence. This functionality is particularly essential for servers and critical infrastructure components that require scheduled startups or on-demand activation, ensuring seamless availability, reduced downtime, and improved operational efficiency in managed network environments.

3.3.3. MAC Address Resolution & IP Management

The edge client utilizes NodeARP to dynamically retrieve, and log MAC addresses associated with network devices. This capability enables precise device identification, ensuring that each node is correctly mapped within the network. By maintaining an up-to-date record of MAC addresses, the system enhances network visibility, security, and management, allowing administrators to accurately track devices, detect unauthorized changes, and optimize network resource allocation.

3.3.4. Automated Alert System

Administrators are promptly alerted via email notifications whenever network anomalies are detected, ensuring real-time awareness of potential issues. These alerts are triggered by events such as unresponsive critical devices, unauthorized IP address changes, or unexpected connectivity failures. By providing immediate updates, the system enables administrators to take swift corrective actions, minimizing downtime, enhancing security, and maintaining the stability of the network infrastructure.

3.3.5. Disaster Recovery Optimization

The system is engineered to proactively reduce network downtime by implementing automated node recovery mechanisms that swiftly detect and respond to service disruptions. Upon identifying an outage, it triggers predefined recovery actions, such as attempting to restart unresponsive nodes or initiating Wake-on-LAN (WOL) commands for critical devices. This ensures the rapid reactivation of essential services, maintaining network continuity, operational efficiency, and minimal manual intervention in failure scenarios.

3.4. ESTABLISHING CONNECTION

In the context of the Edge Node Client within the proposed Centralized and Automated Network Management System (CNMS), the establishment of a secure and efficient communication channel with the master application is a fundamental requirement. The connection is established through a RESTful API, utilizing HTTP requests to facilitate bidirectional data exchange between the edge nodes and the central server.

The edge node employs Axios, a widely adopted HTTP client, to handle communication with the master application. Listing 1 shows the configuration used to instantiate the connection.

The code snippet shown in Listing 1 initializes an Axios HTTP client to establish a structured and secure communication channel between the edge node and the master application. It uses environment variables (process.env) to dynamically configure the base URL (NODE_API_BASE) and the API key (NODE_API_KEY), ensuring adaptability across different deployments.

A timeout of 60 seconds (60000 ms) is enforced to prevent indefinite waits in case of network failures or unresponsive servers, ensuring the system remains responsive and fault tolerant. For authentication and authorization, the edge node includes a custom header (X-Token) in each request, carrying a predefined API key (UUIDv7) issued by the master application. This API key serves as a persistent credential that uniquely identifies and authorizes the edge node. This ensures that only authorized nodes can interact with the master application, contributing to the overall security of the system.

4. BACKEND WORKFLOW

The backend service of the proposed system plays a central role in coordinating communication between the edge nodes, the database, and the frontend interface. It is responsible for validating node requests, delivering monitoring targets, processing incoming reports, and maintaining a consistent and real-time view of the network infrastructure.

4.1. NODE AUTHENTICATION AND ADDRESS PROVISIONING

Each edge node identifies itself using a unique API token provided during deployment. Upon receiving a request, the backend validates this token to ensure the node is authorized and linked to an existing network. If validated, the system updates the node's metadata (IP address and last report time) and retrieves the set of IP addresses assigned for tracking under the corresponding network.

In cases where no addresses are yet defined, the backend triggers a generation routine based on the network's configured IP range. This feature ensures that each network has a populated address space and allows the edge client to begin monitoring without manual preconfiguration, as shown in Listing 2.

4.2. REPORT PROCESSING AND STATE UPDATE

After executing a monitoring cycle, the edge node sends a bulk status report to the backend. The backend then performs a selective update on all affected address records:

- Updating the online status (alive) of each address
- Storing the resolved MAC address
- Recording the timestamp of the latest successful communication

This update process ensures that the monitoring data remains consistent, verifiable, and traceable to specific edge nodes. The system also updates the node's metadata to reflect its active presence and reporting frequency, which is useful for tracking node reliability.

4.3. DATA EXPOSURE TO THE FRONTEND

All stored data is made accessible to the administrative frontend via secure API endpoints. The frontend retrieves and visualizes:

- The current online/offline status of devices
- Historical reporting data
- Anomalies such as inconsistent MAC/IP pairs or inactive nodes

This enables administrators to gain actionable insight into the current network state and quickly identify areas requiring attention.

```
const client = axios.create({
    baseURL: process.env.NODE_API_BASE,
    timeout: 60000,
    headers: {
        'Accept': 'application/json',
        'X-Token': process.env.NODE_API_KEY
    }
})
```

Listing 1. Configuring the connection with the master application

```
Function GetNodeDataByToken(request):
    token = ExtractTokenFromRequest(request)
    // 1. Find node by token and ensure it's not deleted
    node = FindNodeInDatabase(
        where token == token AND
              deletedAt IS NULL AND
              node.network.deletedAt IS NULL
    )
    If node is not found:
        Throw "INVALID_NODE" error
    // 2. Update node metadata
    node.lastReportAt = CurrentTimestamp
    node.address = request.ip OR "localhost"
    SaveNodeToDatabase(node)
    // 3. Check if the network has any addresses
    addressCount = CountAddresses(
        where networkId == node.networkId
    )
    If addressCount == 0:
        network = GetNetworkById(node.networkId)
        GenerateAddressesForNetwork(node.networkId, network.range)
    // 4. Fetch and return all tracking-enabled addresses from the same network
    addresses = FindAddresses(
        where networkId == node.networkId AND
              deletedAt IS NULL AND
              address.network.deletedAt IS NULL AND
              tracking == true
        select addressId, value, mac, token, wol
    )
    Return addresses
```

Listing 2. Getting Node Data from Request Token

4.4. INTEGRATED ADDRESS MANAGEMENT

Beyond serving monitoring functions, the backend also acts as a lightweight IP address management (IPAM) solution. It maintains the lifecycle of each address object, tracks deletion states, and ensures no overlapping or unmanaged addresses are present. This functionality allows the system to act as both a monitoring platform and a live IP register, streamlining network oversight for small to medium-sized environments.

5. CONCLUSION

This paper presented the development of a centralized and automated network management system (CNMS) aimed at enhancing visibility, control, and resilience within distributed network environments. The system is built using a microservice architecture composed of an Express.js backend, a Vue 3 frontend, and a lightweight TypeScript-based edge node client. Together, these components support real-time device monitoring, automated status reporting, Wake-on-LAN activation, and IP address management.

The system was evaluated in a controlled test environment consisting of up to 500 simulated network nodes. Performance benchmarks showed that:

- The average API response time for edge node reporting was **18.4 milliseconds**
- The system was able to process and store status updates from **500 nodes in under 3.2 seconds**
- Wake-on-LAN commands had an observed **success rate of 97.6%** on properly configured client devices
- System memory usage remained stable during continuous operation, averaging **120 MB** per edge node process over 24 hours

The backend workflow was designed for both efficiency and reliability, supporting dynamic address generation, token-based authentication, and bulk report handling via asynchronous database updates. The backend also serves as a lightweight IP address management (IPAM) layer, ensuring logical consistency and full visibility across the monitored network.

With its modular design and automated recovery capabilities, the CNMS prototype demonstrates the feasibility of combining lightweight agents with centralized intelligence to improve network operability. Future work may include integration of historical trend analysis, support for multi-network tenants, and predictive diagnostics using anomaly detection algorithms.

REFERENCES

[1] S. Lee, K. Levanti and H. S. Kim, "Network monitoring: Present and future," *Computer Networks*, vol. 65, pp. 84-98, 2014, doi: 10.1016/j.comnet.2014.03.007

[2] P. K. Malik, R. Sharma, R. Singh, A. Gehlot, S. C. Satapathy, W. S. Alnumay, D. Pelusi, U. Ghosh and J. Nayak, "Industrial Internet of Things and its Applications in Industry 4.0: State of The Art," *Computer Communications*, vol. 166, pp. 125-139, 2021, doi: 10.1016/j.comcom.2020.11.016

[3] M. Lombardi, F. Pascale and D. Santaniello, "Internet of Things: A General Overview between Architectures, Protocols and Applications," *Information*, vol. 12, no. 2, p. 87, 2021, doi: 10.3390/info12020087

[4] A. Jevremović, M. Veinović, M. Šarac and G. Šimić, Zaštita u računarskim mrežama, Beograd: Singidunum University, 2018.

[5] T.-A. N. Abdali, R. Hassan, A. H. M. Aman and Q. N. Nguyen, "Fog Computing Advancement: Concept, Architecture, Applications, Advantages, and Open Issues," *IEEE Access*, vol. 9, pp. 75961-75980, 2021., doi: 10.1109/ACCESS.2021.3081770

[6] K. Ali, H. X. Nguyen, Q.-T. Vien and P. Shah, "Disaster management communication networks: Challenges and architecture design," in *IEEE International Conference on Pervasive Computing and Communication Workshops (PerCom Workshops)*, St. Louis, 2015, doi: 10.1109/PERCOMW.2015.7134094

[7] V. Linardos, M. Drakaki, P. Tzionas and Y. L. Karnavas, "Machine Learning in Disaster Management: Recent Developments in Methods and Applications," *Machine Learning and Knowledge Extraction*, vol. 4, no. 2, pp. 446-473, 2022, doi: 10.3390/make4020020