UDC 004.056.55
DOI: 10.15587/2706-5448.2025.328856

**Bohdan Hinko,**
**Oleksandr Boretskyi,**
**Vitalii Marianovskyi**

# DEVELOPMENT OF AUTOMATED COLLECTION METHOD OF INITIAL DIAGNOSTIC INFORMATION FOR THE TECHNICAL SUPPORT SERVICE OF ORGANIZATION NETWORK USERS

*The object of the study is the process of collecting initial diagnostic information by the technical support service (Service Desk/Help Desk) in organizations where IT infrastructure is a key element of business processes.*

*One of the most problematic areas is the manual and inefficient collection of accurate diagnostic data from users who often lack sufficient technical knowledge. This leads to significant delays at the primary diagnostics stage, increases overall system downtime, and directly impacts employee productivity, especially when network infrastructure problems arise.*

*In the course of the study, an approach is proposed that involves optimizing and automating the collection of primary network diagnostic information directly from the user's side. This method includes automatically checking the physical connection, obtaining correct network settings (IP address, DNS, etc.), and verifying resource accessibility over the network.*

*The expected result is a significant increase in the speed and quality of the technical support service's work. This is due to the fact that the proposed automated method minimizes the need for lengthy user questioning and sequential manual checks of settings. It has a number of features, in particular, a focus on automating data collection specifically from the network infrastructure, which is the foundation for the vast majority of IT services.*

*This approach allows to automate the collection of diagnostic data in an infrastructure built using equipment from different vendors and does not depend on the specific software implementation of network services, monitoring and logging services. Compared to similar known traditional methods, this approach provides such advantages as reduced downtime, a lower risk of significant financial losses for the company, and an increase in overall user satisfaction with the quality of IT services.*

***Keywords**: automation, support, networks, diagnostics, API, Celery, Zammad, FastAPI, incident management.*

*Received: 16.04.2025*
*Received in revised form: 18.06.2025*
*Accepted: 08.07.2025*
*Published: 29.08.2025*



***How to cite***
*Hinko, B., Boretskyi, O., Marianovskyi, V. (2025). Development of automated collection method of initial diagnostic information for the technical support service of organization network users. Technology Audit and Production Reserves, 4 (2 (84)), 29–36. https://doi.org/10.15587/2706-5448.2025.328856*

## 1. Introduction

Reliable information technology infrastructure is the foundation for the effective operation of modern organizations [1]. Any failures in the IT environment, especially network problems, can significantly impact productivity, so their rapid resolution is critically important [2]. The user technical support service plays a key role in this, accepting, classifying, and resolving incidents [3]. The quality of its work directly affects user satisfaction, but effectiveness is often limited by the large volume of requests and the difficulty of obtaining accurate diagnostic information from users [4]. The primary diagnostics stage is particularly important, where delays or errors can significantly prolong downtime.

Network infrastructure underlies most IT services [5], and significant portions of the problem users face are related specifically to it. Ensuring stable network connectivity is a basic support task. Traditional manual diagnostic methods are often slow and inefficient. Therefore, optimizing and automating the collection of primary diagnostic information is a relevant task for improving the quality of the support service's work [6].

This issue is particularly significant for the extensive network infrastructure of the dormitories at Taras Shevchenko National University of Kyiv (KNU). The network is built on a hierarchical model (Fig. 1) [7], which includes core, distribution, and access components for connecting users. High-speed communication lines and redundancy mechanisms ensure the stability and reliability of the infrastructure. Logical network segmentation at the dormitory and floor levels (using VLANs and subnets) improves manageability and simplifies fault localization [8]. In addition to logical faults, a significant portion of incidents arise from hardware malfunctions, frequently caused by the unstable performance of power supply units for network equipment. These issues can manifest as a complete device failure or, more deceptively, as a partial malfunction where power indicators are lit, but the device remains non-operational – a problem particularly noted when operating on various battery backup power sources. It is important that the proposed automated diagnostic method is designed to function and gather data even under such conditions of partial equipment failure, thereby assisting in the precise identification of the hardware-related cause of the malfunction. Network access in rooms is provided through

ISSN-L 2664-9969; E-ISSN 2706-5448

a cabling system and floor switches connected to the central dormitory equipment. Technologies are used to prevent network loops and ensure rapid link recovery. Security is ensured through access control lists and authentication requirements for accessing equipment management, and all actions are journaled and linked to requests in the task management system [9].

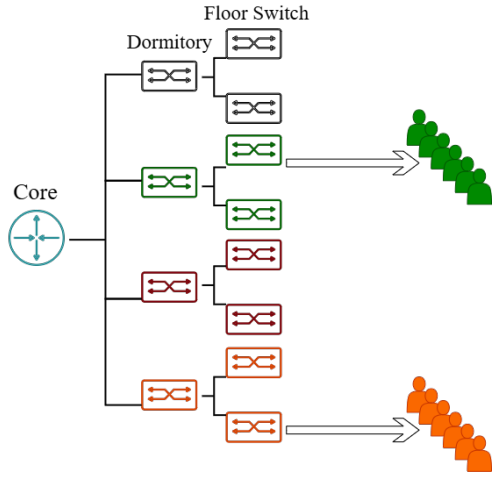


**Fig. 1.** Schematic diagram of the KNU dormitory network

The Sector for Ensuring Internet Access in University Dormitories of the Information and Computing Center is responsible for the operation of this network and user support. The support service operates on a multi-level model (L1/L2/L3) and uses ITIL practices [10] and ISO 20000 standards [11], as well as the Zammad system for managing requests [12]. The priority is the rapid restoration of services and analysis of failure causes to prevent them in the future [13], which is reflected in KPIs, particularly MTTR.

However, the current process for diagnosing network incidents, applied by the Sector for Ensuring Internet Access in University Dormitories, remains predominantly manual. The process starts with identifying the user and their network connection. Then, the correct network port is determined based on documentation. A connection to the switch is established via the console interface. Diagnostic commands are executed step by step in the CLI. Finally, a report is created manually and the request is updated in the task management system.

This manual procedure has significant drawbacks: it is lengthy (often taking over 30 minutes per incident), which reduces the throughput of the support service, and is vulnerable to errors due to the human factor [14]. Additionally, there is a high barrier to entry for new L1 engineers, positions often held by students who require time to master the specifics of the network and diagnostic tools.

Considering these drawbacks – lengthiness, risk of errors, and adaptation difficulties – a justified need arises for implementing an automated procedure for collecting primary diagnostic information within the KNU support service.

*The aim of research* is to develop a method and a conceptual architectural model based on it for a system that automates the primary collection of diagnostic information for network incidents. Destination infrastructure for proposed method is typical 3-level network hierarchical model which widely used in mid- and large- size companies.

In order to confirm that proposed method solve task of collecting diagnostic data from network infrastructure and can reduce time needed for finding the root cause of network failure was created set of tools that implement diagnostic data collection according to proposed method. It should allow to reduce mean time to repair (MTTR) and thereby increasing the overall productivity of the support service. As test environment used network segment that holds network infrastructure of the dormitories at Taras Shevchenko National University of Kyiv, contains bunch of network equipment, with integration of the support service's existing processes and monitoring tools, particularly the Zammad ticket management system.

## 2. Materials and Methods

*The object of the study* is the process of collecting initial diagnostic information by the technical support service (Service Desk/Help Desk) in organizations where IT infrastructure is a key element of business processes.

This work includes two main stages. First, it analyzes how network issues are currently diagnosed in KNU dormitories. Second, it proposes an automated system to fix the problems found. Understanding the current state is a necessary prerequisite for developing an effective target solution.

The analysis of the current diagnostic process (Fig. 2) began with identifying the key participants, or actors, and their roles. The main initiator of the process is the network user (student or employee), who encounters a problem and contacts the support service. The central figure in the manual process is the first-line support engineer (L1), who is tasked with user interaction, primary diagnostics, and documentation. Higher-level engineers (L2/L3) are involved indirectly when complex incidents are escalated. The analysis methodology included studying typical interaction scenarios between these actors. The sequence of actions ("workflow") of the L1 engineer was examined in detail.

The process is usually initiated by a user request through informal channels, most often the Telegram messenger or email. Upon receiving the request, the L1 engineer manually creates a ticket in the Zammad request management system, recording the initial information and contact details. Next, the engineer initiates a dialogue with the user to clarify the details of the problem, its nature, and the time of occurrence. The next step is performing basic checks: the engineer may ask the user to check the connection, indicators, execute commands (e. g., ipconfig, ping), and send the results [15]. Concurrently, it is possible to consult the Grafana monitoring system [16] to check the general status of the equipment or network [17]. If the basic checks do not yield results, the engineer proceeds to diagnostics at the infrastructure level [18]. This requires identifying the specific switch port to which the user is connected, which, in the absence of an up-to-date automated mapping system, can be complex and requires consulting documentation or internal tables. After identifying the port, the engineer connects to the switch via the SSH protocol [19], using their own credentials. It is done to manually execute a set of diagnostic commands in the command line (CLI) to check the port status, MAC address table, VLAN settings, cable diagnostics, etc. [20]. Throughout the entire process, the engineer manually documents all their actions, communication with the user, and the results of command execution in the corresponding Zammad ticket. The cycle concludes with a decision: either resolving the problem at the L1 level or escalating it to a higher level with the transfer of collected information.

Analysis of this current process revealed a number of significant shortcomings. The main problems include time delays during communication with the user, manual ticket creation in Zammad, port identification, SSH connection, and sequential command execution. There is significant duplication of effort in manually documenting information in Zammad. The process is sensitive to the human factor, which can lead to errors when entering commands or interpreting their results. This creates "bottlenecks", reduces the throughput of the support service, and negatively impacts the MTTR indicator. The high barrier to entry for new L1 engineers, often students, is also a significant drawback.

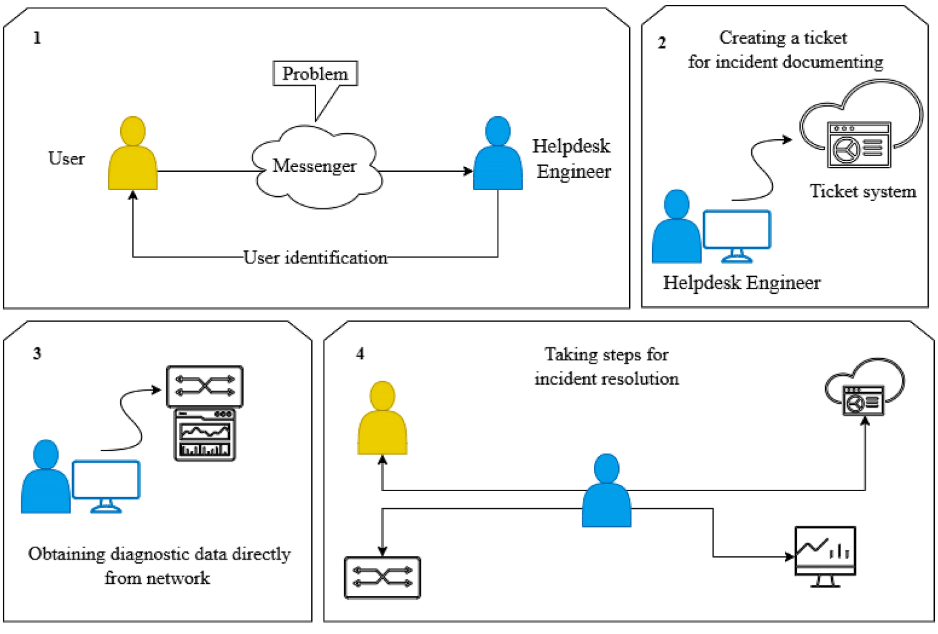


**Fig. 2.** Schematic of the manual process for providing technical support to users

## 3. Results and Discussion

Conceptual design involves creating a specialized software system that integrates into the existing workflow. The proposed method changes the sequence of actions: the L1 engineer, having received the request and created a ticket in Zammad, initiates automated diagnostics through the system's corresponding interface. After this, the automation system takes over the execution of the diagnostic scenario. The system automatically finds the needed network device using a configuration database. It connects securely using standard protocols and runs a set of diagnostic commands. If needed, it also collects extra data from the monitoring system. Then, it analyzes the results, structures the data, and checks for common issues. Finally, it creates a diagnostic report and updates the related request in Zammad with this report. The engineer receives a notification and can review the structured results directly in the ticket system to make further decisions. A visual model of the target process (Fig. 3) illustrates these data flows and interactions between the engineer, the automation system, network equipment, and other systems (Zammad, optionally Grafana). A comparison of the information collection processes shows several improvements: faster diagnostics, better standardization and reliability, fewer human errors, and easier documentation. This lets L1 engineers focus on more complex tasks.

To implement this method, a conceptual architectural model of the system was developed [21]. It is based on the modular principle and the distribution of responsibility between logical layers. The presentation layer defines the interaction points for L1 engineers, primarily through integration with the existing Zammad system. The service layer includes the main logic and key components. It has a central control service (Backend API Server) that handles requests and manages the process.

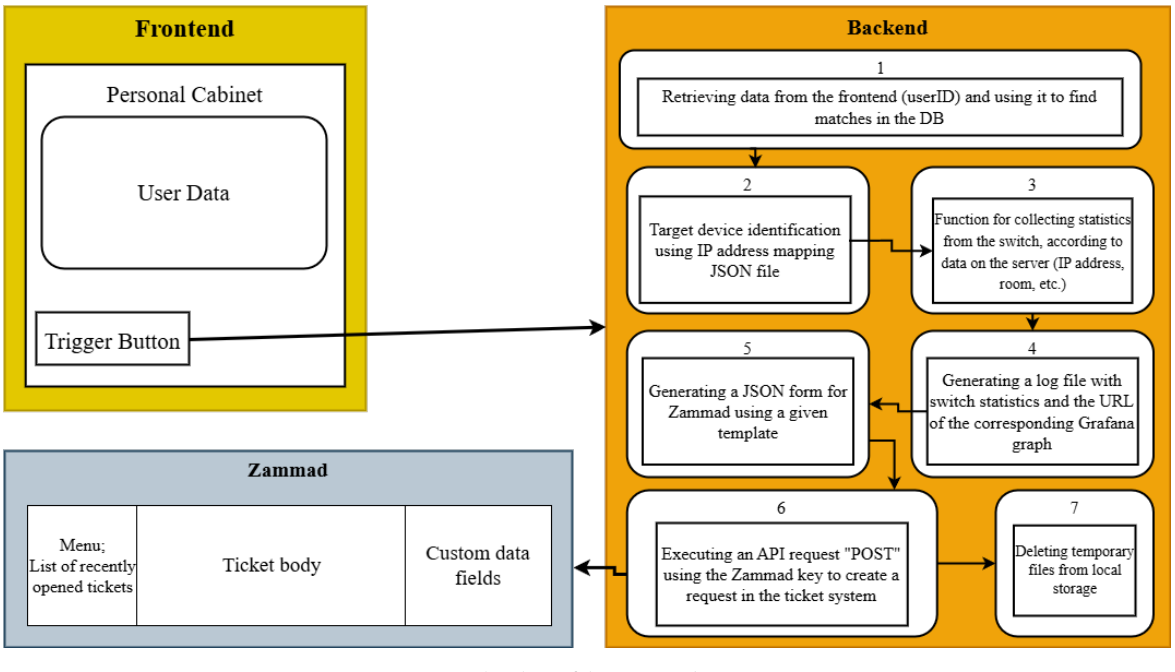


**Fig. 3.** Flowchart of the automated process

ISSN-L 2664-9969; E-ISSN 2706-5448

An asynchronous task system with message queues runs long diagnostics in the background. Diagnostic handlers (Workers) interact with devices and analyze data. There are also modules for working with management protocols, parsing data, integrating with systems like Zammad and the monitoring tool, and managing settings and credentials. The data layer describes the logical organization of configuration, operational information, and diagnostic results. The infrastructure layer defines the requirements for the server environment and network connections for deploying and operating the system.

System design (Fig. 4) was carried out considering security requirements and policy compliance. Security assurance methods included developing an approach for secure credential management for accessing network devices, protecting the system's own interfaces through authentication and authorization, and considering network security requirements when configuring access. Issues of compliance with policies for processing technical information and personal data were analyzed separately.

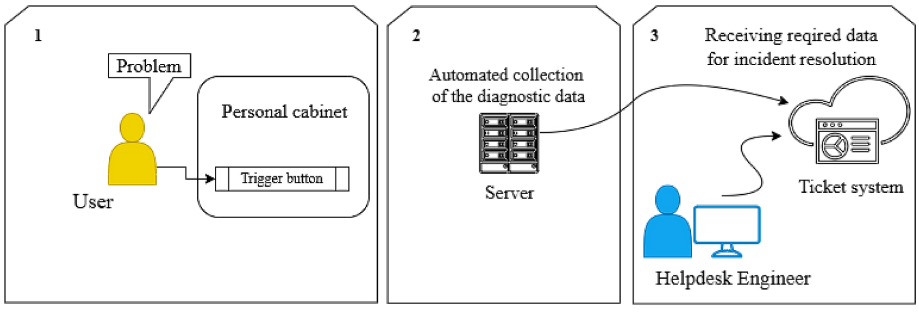


**Fig. 4.** Example workflow of the target process using the automated method

A risk assessment methodology was applied to identify potential technical (algorithm errors, interaction issues with equipment or APIs), operational (staff resistance, configuration update problems), and security (data compromise, unauthorized access) risks. For each type of risk, mitigation strategies were developed, including testing, actions journaling, documentation, staff training, configuration management procedures, and the implementation of security practices.

The choice to develop a specialized solution was justified through comparative analysis with alternative approaches. Options considered included optimizing manual processes (detailed instructions, training), using disparate general-purpose automation tools (network libraries, configuration management systems), and implementing comprehensive commercial ITSM/AIOps platforms. Analysis showed that developing a custom system is the most appropriate approach for the specific conditions and needs of the university support service, providing necessary flexibility and integration with existing infrastructure, particularly Zammad, at optimal costs.

At the conclusion of the design phase, a methodology for evaluating the effectiveness of the developed solution and specific, measurable success criteria were defined. Key indicators include reducing the average primary diagnostic time (target: at least 30–40%), increasing the L1 Resolution Rate (target: 10–15%), reducing overall MTTR, a high level of standardization and reliability in diagnostics (minimal system error rate), as well as complete and automatic documentation of results in Zammad. In addition to quantitative indicators, qualitative benefits are considered, such as lowering the barrier to entry for new employees and increasing service consistency. These criteria are used for further testing and analysis of the implemented system's results.

The practical implementation of the system is based on a carefully selected technology stack that meets the functional and non-functional requirements defined during the design phase. Python was chosen as the main programming language due to its readable syntax, large ecosystem of libraries that accelerate development, and strong support for network automation and asynchronous programming. The asynchronous web framework FastAPI was used to build the server-side (backend) and the application programming interface (API) [22]. Its choice is justified by high performance, built-in support for asynchronous operations, automatic data validation using Pydantic models, and automatic API documentation generation (based on OpenAPI) [23], which simplifies integration and testing. The Celery distributed task queue system was used to handle potentially long-running diagnostic tasks without blocking the main server [24]. This approach allows offloading diagnostic execution to separate background processes (workers), ensuring asynchronicity, scalability, and increased reliability through the use of a message broker (Redis in this case). Interaction with network equipment (switches) via the SSH protocol is implemented using specialized Python libraries, such as Netmiko or Paramiko. Integration with the Zammad ticketing system and, potentially, with the Grafana monitoring system is carried out via their REST APIs [25] using standard HTTP requests implemented with the requests or httpx libraries. Stable Linux distributions, such as Rocky Linux or Ubuntu, were considered as the operating system for deploying server components. JSON format files were used during the development phase to store configuration information, particularly the mapping of switch IP addresses. API testing was conducted using Postman tools and the built-in FastAPI documentation [26].

The practical implementation of the system is built as a typical Python project with a modular structure for ease of development and maintenance. The main code is located in packages corresponding to key functional blocks: api (FastAPI web server code), worker (implementation of asynchronous Celery tasks), core (shared business logic, SSH interaction, parsing, analysis), config (configuration management), models (Pydantic data models) [27]. Key classes and modules were defined: the FastAPI application instance (app = FastAPI()), APIRouter objects for grouping endpoints, Pydantic models for data validation; the Celery application instance and task functions (e. g., run_diagnostics_task); a module for SSH interaction with proper exception handling; an integration module with the Zammad API for updating tickets and potentially a module for the Grafana API; modules for parsing text output from commands and analyzing data based on diagnostic rules; a configuration loading module (including switch IP mapping from a JSON file). Entry points are the launch of the ASGI server (e. g., uvicorn) and commands to start Celery workers.

The central component is the backend service, implemented with FastAPI, which provides a REST API for interaction. The main endpoints were designed and implemented: POST/api/diagnostics/start to initiate a diagnostic task (accepts JSON with user/request ID, returns task ID); GET/api/diagnostics/status/{task_id} to check the status of an asynchronous task; GET/api/diagnostics/result/{task_id} to retrieve the final report after successful execution. Other endpoints (/create_ticket_mock, /cleanup) were used as auxiliary during the development phase. Using the Pydantic library ensures automatic validation of input data and serialization of output data into JSON format, increasing API reliability and consistency. FastAPI's interaction with Celery is implemented using the "producer-consumer" pattern: the FastAPI handler (/start) acts as the producer, quickly sending a task with parameters to the Celery message queue and returning a task ID to the client. Retrieving status and results occurs by querying the Celery results backend using the task ID. This ensures API responsiveness and the possibility of scaling processing.

The main diagnostic logic is implemented in asynchronous tasks of Celery workers. The request processing algorithm in the worker begins with receiving the task and its parameters. Next, the following steps are executed sequentially: identifying the target switch (based on user input data and the configuration mapping file); establishing an SSH connection using appropriate credentials and libraries (Netmiko/Paramiko); sequential execution of a standardized set of diagnostic CLI commands and collection of their text output; parsing the received output to extract structured data; analysis of the structured data using a set of diagnostic rules to form conclusions about possible problems. An important part of the algorithms is error handling at each stage (device unavailability, authentication errors, command execution errors, parsing errors) with corresponding actions journaling and the ability to continue execution or correctly terminate the task marked with an error.

The system's analysis engine employs a set of diagnostic rules to identify a broad range of both hardware malfunctions and configuration errors. The implemented logic targets common network issues, including incorrect port states (e. g., "err-disabled"), VLAN misconfigurations, physical layer anomalies detected via cable diagnostics, port security violations, excessive interface error rates (CRC, giants, runts), network loops, and improper MAC address table population. Furthermore, the methodology is particularly effective in diagnosing critical hardware faults. This includes catastrophic switch hardware failure, complete power supply unit failure, and intermittent operational freezes that can be induced by the use of unsupported, non-OEM (original equipment manufacturer) backup power systems.

Close integration with existing systems is key to implementing the solution into the workflow. Integration with the Zammad ticketing system via its REST API has been implemented. A Zammad API client software module was created, using API tokens for authentication. The main functionality is automatic ticket updating: after diagnostics are complete, the system formats the report and adds it as a comment to the corresponding ticket via a POST/PUT request to the Zammad API. It is also possible to retrieve initial context from the Zammad ticket via GET requests. Integration with the Grafana monitoring system, also via its HTTP API, is planned.

This allows the automation system to request historical data or detailed metrics from relevant Grafana dashboards [28]. The obtained data can supplement the results of SSH diagnostics.

Comprehensive testing was conducted to verify functionality and evaluate effectiveness. The test environment was deployed on a virtual machine (VMware Workstation, Ubuntu) using Python venv, Redis, and tmux for process management. Software simulators ("stubs") were used for SSH interaction and the Zammad API, although Zammad and Grafana instances were deployed on a test basis to demonstrate configuration. System testing of the main workflow was carried out by sending curl requests to the /api/help-request API endpoint for several test users. Monitoring Celery worker action journaling (Fig. 5) confirmed the correct execution of all stages: receiving the task, querying the user data simulator, determining the switch IP address, simulating the execution of SSH commands on the switch, successfully saving diagnostic results to a text file, and simulating ticket creation in Zammad.

Successful creation of the diagnostic file (Fig. 6) was additionally verified using the commands ls -l /tmp/diagnostics/ and cat /tmp/diagnostics/<filename>.txt. The file content fully matched the expected format and simulated data.

The results of this testing stage confirmed the operability of the system's internal logic and the correct interaction between components when working with simulated dependencies. An attempt to test integration with the real Zammad API (uncommented code in tasks.py, parameters from.env) showed the system correctly formulated the ticket creation request (Fig. 7). However, the connection attempt failed due to a requests.exceptions.SSLError ([SSL: WRONG_VERSION_NUMBER] wrong version number), which indicates problems with the SSL/TLS configuration of the test Zammad server, rather than problems in the automation system's code.

```
[2025-05-04 15:19:35,976: WARNING/ForkPoolWorker-2] TASK: Імітація виконання: show version
[2025-05-04 15:19:36,077: WARNING/ForkPoolWorker-2] TASK: Імітація виконання: show switch
[2025-05-04 15:19:36,178: WARNING/ForkPoolWorker-2] TASK: Імітація виконання: ping 10.1.105.16
.10
[2025-05-04 15:19:36,279: WARNING/ForkPoolWorker-2] TASK: Імітація виконання: show mac address
-table address DD:EE:FF:44:55:66
[2025-05-04 15:19:36,379: WARNING/ForkPoolWorker-2] TASK: Діагностику на 10.1.5.1 завершено (і
мітація).
[2025-05-04 15:19:36,379: WARNING/ForkPoolWorker-2] TASK: Діагностичний вивід збережено: /tmp/
diagnostics/diag_10_1_5_1_65da2e81-2a8a-46ad-80fb-f323f19500e9.txt
[2025-05-04 15:19:36,379: WARNING/ForkPoolWorker-2] TASK: Генерація посилання Grafana для 10.1
.5.1...
[2025-05-04 15:19:36,379: WARNING/ForkPoolWorker-2] TASK: Посилання Grafana: http://192.168.15
4.130:3333/d/eejpdn7au1ddsb/autodiag-test1?orgId=1&from=now-6h&to=now&timezone=browser?var-swi
tch=10.1.5.1&var-user=Інший Користувач
[2025-05-04 15:19:36,379: WARNING/ForkPoolWorker-2] TASK: Створення тікету Zammad для користув
ача ID: user-456...
[2025-05-04 15:19:36,379: WARNING/ForkPoolWorker-2] TASK: Дані для Zammad API (без файлу):
{
  "title": "Авто-запит: Інший Користувач (10.1.105.16.10)",
  "group": "Dorms",
  "article": {
    "subject": "Автоматична Діагностика",
    "body": "**Деталі Користувача:**\n* Ім'я: Інший Користувач\n* Email: other@example.com\n*
IP Адреса: 10.1.105.16.10\n* MAC Адреса: DD:EE:FF:44:55:66\n* Гуртожиток: 5\n* Поверх: 16\n* К
імната: 401/1\n* Телефон: 3805044445566\n\n**Діагностична Інформація:**\n* Посилання Grafana: h
ttp://192.168.154.130:3333/d/eejpdn7au1ddsb/autodiag-test1?orgId=1&from=now-6h&to=now&timezone
=browser?var-switch=10.1.5.1&var-user=Інший Користувач\n* Результати первинної діагностики: До
дано у вкладенні.\n",
    "type": "note",
    "internal": false
  },
  "priority_id": 2,
  "custom_dorm": "5",
  "custom_floor": "16",
  "custom_grafana_link": "http://192.168.154.130:3333/d/eejpdn7au1ddsb/autodiag-test1?orgId=1&
from=now-6h&to=now&timezone=browser?var-switch=10.1.5.1&var-user=Інший Користувач"
}
[2025-05-04 15:19:36,379: WARNING/ForkPoolWorker-2] TASK: Відправка запиту на створення тікету
 Zammad (імітація)...
```

**Fig. 5.** Celery worker message journal for successful user-123 diagnostics process

ISSN-L 2664-9969; E-ISSN 2706-5448

```
--- Diagnostics Report ---
Timestamp: 2025-05-04T14:37:44.773543
Target Switch: 10.1.11.1
User IP: 10.1.111.9.15
User MAC: AA:BB:CC:11:22:33
-------------------------

--- Command: show version ---
[Simulated output for 'show version']

--- Command: show switch ---
[Simulated output for 'show switch']

--- Command: ping 10.1.111.9.15 ---
[Simulated output for 'ping 10.1.111.9.15']

--- Command: show mac address-table address AA:BB:CC:11:22:33 ---
[Simulated output for 'show mac address-table address AA:BB:CC:11:22:33']

--- Diagnostics Finished ---
```

**Fig. 6.** Example contents of the generated diagnostic file

```
[2025-05-04 15:30:05,365: WARNING/ForkPoolWorker-2] TASK: Створення тікету Zammad для користувача ID:
user-123...
[2025-05-04 15:30:05,366: WARNING/ForkPoolWorker-2] TASK: Дані для Zammad API (без файлу):
{
  "title": "Авто-запит: Гінько Богдан Ігорович (10.1.111.9.15)",
  "group": "Dorms",
  "article": {
    "subject": "Автоматична Діагностика",
    "body": "**Деталі Користувача:**\n* Ім'я: Гінько Богдан Ігорович\n* Email: test@example.com\n* IP
Адреса: 10.1.111.9.15\n* MAC Адреса: AA:BB:CC:11:22:33\n* Гуртожиток: 11\n* Поверх: 9\n* Кімната: 302/
2\n* Телефон: 380501112233\n\n**Діагностична Інформація:**\n* Посилання Grafana: http://192.168.154.13
0:3333/d/eejpdn7au1ddsb/autodiag-test1?orgId=1&from=now-6h&to=now&timezone=browser?var-switch=10.1.11.
1&var-user=Гінько Богдан Ігорович\n* Результати первинної діагностики: Додано у вкладенні.\n",
    "type": "note",
    "internal": false
  },
  "priority_id": 2,
  "custom_dorm": "11",
  "custom_floor": "9",
  "custom_grafana_link": "http://192.168.154.130:3333/d/eejpdn7au1ddsb/autodiag-test1?orgId=1&from=now
-6h&to=now&timezone=browser?var-switch=10.1.11.1&var-user=Гінько Богдан Ігорович"
}
[2025-05-04 15:30:05,366: WARNING/ForkPoolWorker-2] TASK: Відправка запиту на створення тікету Zammad
(імітація)...
[2025-05-04 15:30:05,368: WARNING/ForkPoolWorker-2] TASK: ПОМИЛКА Zammad API (створення): HTTPSConnect
ionPool(host='192.168.154.130', port=443): Max retries exceeded with url: /api/v1/tickets (Caused by S
SLError(SSLCertVerificationError(1, '[SSL: CERTIFICATE_VERIFY_FAILED] certificate verify failed: self-
signed certificate (_ssl.c:1007)')))
[2025-05-04 15:30:05,369: WARNING/ForkPoolWorker-2] !!! CELERY TASK FAILED (user_id: user-123): Runtim
eError - Помилка створення тікету Zammad. !!!
[2025-05-04 15:30:05,369: INFO/ForkPoolWorker-2] Task tasks.run_full_diagnostic_flow[e27d8638-936e-40a
a-af59-a80ca1e982e2] succeeded in 0.9087401870001486s: None
```

**Fig. 7.** Celery worker error message journal when attempting a real request to the Zammad API

This test confirmed the readiness of the integration module to work with a correctly configured Zammad server (Fig. 8).

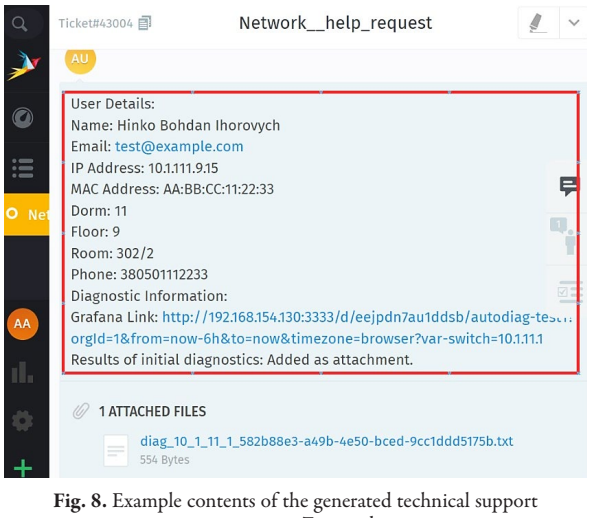


**Fig. 8.** Example contents of the generated technical support request in Zammad

Performance analysis based on the message journals of the test run with simulators showed a processing time of approximately 1.1 seconds. Applying a conservative safety factor of ×100 to account for real delays (SSH, API, load), the upper threshold for the automated process execution time was estimated at approximately 110 seconds. Comparing this time with the manual process (about 30 minutes or 1800 seconds) demonstrates a potential speedup of the primary diagnostics collection process by approximately 16 times. Testing limitations were noted (simulators, lack of load testing), and pilot testing under real conditions is recommended.

The implemented system serves as a basis for further improvements. Promising directions include expanding diagnostic capabilities: supporting more equipment models, deepening analytical functionality (more complex rules, data correlation with Grafana) [29], dynamically determining diagnostic steps. Improving integration is important: deeper integration with Zammad (knowledge base suggestions, action initiation), development of a separate web interface for engineers (history view, configuration management). Research into applying ML elements for data analysis, anomaly detection [30], or failure prediction could bring the system closer to the concept of proactive support and AIOps.

Key benefit of proposed solution is its flexibility. Proposed method of collecting diagnostic data allows to integrate big network equipment amount from different vendors and various monitoring and logging solutions. Main restrictions for network equipment is support management via CLI with remote access using SSH or Telnet protocols or SNMP or HTTP REST API which are accessible in network equipment

of enterprise level. Monitoring and logging system has interfaces for automated data export. Diagnostic tools used in solution are default network diagnostic tools widely used in almost all OS's that have network support. Processing of diagnostic data is custom solution which make sense to adopt for every user of proposed solution based on user preferences. It can be simple collecting data with processing only for selecting defined time slot and it can be complex data processing for more deep automated analysis of diagnostic information.

## 4. Conclusions

Research into the specifics of providing technical assistance to users of the university dormitory network revealed significant drawbacks in the manual collection of diagnostics, particularly considerable time delays and inconsistency, justifying the feasibility of automating this process. To address this problem, a conceptual model and a flexible modular architecture for an automated system were developed, integrating with existing ITSM tools based on first time proposed method of automation of collecting and analyzing diagnostic data.

The approaches used in the method to collect and analyze diagnostic data allow the solution to be applied in heterogeneous networks built on equipment from different vendors and do not impose additional requirements on the services that ensure the functioning of the network and monitoring and logging services.

The practical implementation of the concept resulted in a functional software prototype using a Python, FastAPI, Celery stack, with the capability for Zammad API integration. Testing the prototype in a laboratory environment confirmed its operability, the correctness of handling diagnostic scenarios with simulators, and demonstrated significant potential for accelerating the process compared to the manual method.

The developed approach proves its value in diagnosing common hardware faults, related to the improper functioning of power supply units. As a result of applying the developed method, it was possible to reduce the time required to collect diagnostic information by 16 times due to automation tools and reducing the impact of the human factor on the collection process.

## Conflict of interest

The authors declare that they have no conflict of interest in relation to this research, whether financial, personal, authorship or otherwise, that could affect the research and its results presented in this paper.

## Financing

The research was performed without financial support.

## Data availability

Manuscript has no associated data.

## Use of artificial intelligence

The authors confirm that they did not use artificial intelligence technologies when creating the current work.

### References

1. Agyemang, D. Y., Fong, P. S. W., Kissi, E. (2019). The influence of organizational infrastructure on organizational effectiveness in the construction industry. *CIB World Building Congress 2019.*
2. Abid, A., Khan, M. T., Iqbal, J. (2020). A review on fault detection and diagnosis techniques: basics and beyond. *Artificial Intelligence Review, 54 (5)*, 3639–3664. https://doi.org/10.1007/s10462-020-09934-2
3. *The official introduction to the ITIL service lifecycle* (2007). OGC-Office of Government Commerce. The Stationery Office, 238.
4. Galup, S., Quan, J. J., Dattero, R., Conger, S. (2007). Information technology service management. *Proceedings of the 2007 ACM SIGMIS CPR Conference on Computer Personnel Research: The Global Information Technology Workforce*, 46–52. https://doi.org/10.1145/1235000.1235010
5. Shen, N., Yu, B., Huang, M., Xu, H. (2021). *Campus Network Architectures and Technologies.* CRC Press. https://doi.org/10.1201/9781003143314
6. He, Y., Wong, Y.-P., Liang, Q., Wu, T., Bao, J., Hashimoto, K.-Y. (2022). Double Busbar Structure for Transverse Energy Leakage and Resonance Suppression in Surface Acoustic Wave Resonators Using 42°YX-Lithium Tantalate Thin Plate. *IEEE Transactions on Ultrasonics, Ferroelectrics, and Frequency Control, 69 (3)*, 1112–1119. https://doi.org/10.1109/tuffc.2022.3144188
7. Peterson, L. L., Davie, B. S. (2007). *Computer networks: A systems approach.* Elsevier.
8. *IEEE 802.3-2022. IEEE Standard for Ethernet.* IEEE Standards Association. Available at: https://standards.ieee.org/standard/802_3-2022.html
9. Rigney, C., Willens, S., Rubens, A., Simpson, W. (2000). *Remote Authentication Dial in User Service (RADIUS).* RFC Editor. https://doi.org/10.17487/rfc2865
10. Lubis, M., Cherthio Annisyah, R., Lyvia Winiyanti, L. (2020). ITSM Analysis using ITIL V3 in Service Operation in PT.Inovasi Tjaraka Buana. *IOP Conference Series: Materials Science and Engineering, 847 (1)*, 012077. https://doi.org/10.1088/1757-899x/847/1/012077
11. *ISO/IEC 20000-1:2018 Information technology – Service management – Part 1: Service management system requirements* (2018). International Organization for Standardization.
12. *Zammad System Documentation.* Available at: https://docs.zammad.org
13. Duman, İ., Eliiyi, U. (2021). Performance Metrics and Monitoring Tools for Sustainable Network Management. *Bilişim Teknolojileri Dergisi, 14 (1)*, 37–51. https://doi.org/10.17671/gazibtd.780504
14. Aglibar, K. D., Rodelas, N. (2022). *Impact of critical and auto ticket: Analysis for management and workers productivity in using a ticketing system.* arXiv. https://doi.org/10.48550/arXiv.2203.03709
15. Postel, J. (1981). *Internet Control Message Protocol.* RFC Editor. https://doi.org/10.17487/rfc0792
16. Turnbull, J. (2019). *The art of monitoring.* Available at: https://artofmonitoring.com
17. Chakraborty, M., Kundan, A. P. (2021). *Grafana. Monitoring Cloud-Native Applications: Lead agile operations confidently using open source software.* Berkeley: Apress, 187–240. https://doi.org/10.1007/978-1-4842-6888-9_6
18. Hoda, M. (2005). *Cisco network security troubleshooting handbook.* Cisco Press. Available at: https://www.ciscopress.com/store/cisco-network-security-troubleshooting-handbook-9781587054433
19. Choi, B. (2024). Python Network Automation Labs: SSH in Action, paramiko and netmiko Labs. *Introduction to Python Network Automation Volume II: Stepping up: Beyond the Essentials for Success.* Berkeley: Apress, 121–227. https://doi.org/10.1007/979-8-8688-0391-8_3
20. Lee, H. M., Lee, G. S., Kwon, G.-Y., Bang, S. S., Shin, Y.-J. (2021). Industrial Applications of Cable Diagnostics and Monitoring Cables via Time–Frequency Domain Reflectometry. *IEEE Sensors Journal, 21 (2)*, 1082–1091. https://doi.org/10.1109/jsen.2020.2997696
21. Ziade, T. (2017). *Python microservices development: Build, test, deploy, and scale microservices in Python.* Packt Publishing, 340.
22. Tragura, S. J. C. (2022). *Building Python microservices with FastAPI: Build secure, scalable, and structured Python microservices from design concepts to infrastructure.* Packt Publishing, 420.
23. Ponelat, J., Rosenstock, L. (2022). *Designing APIs with Swagger and OpenAPI.* Manning.
24. *Celery – Distributed Task Queue.* Available at: https://docs.celeryq.dev/en/stable/
25. Surwase, V. (2016). REST API modeling languages – A developer's perspective. *International Journal of Science Technology & Engineering, 2 (10)*, 634–637. Available at: https://www.ijste.org/articles/IJSTEV2I10199.pdf
26. *Postman documentation overview.* Available at: https://learning.postman.com/docs/introduction/overview/
27. Lubanovic, B. (2023). *FastAPI: Modern Python web development.* O'Reilly Media, Inc, 277
28. Manases, L., Zinca, D. (2022). Automation of Network Traffic Monitoring using Docker images of Snort3, Grafana and a custom API. *2022 21st RoEduNet*

ISSN-L 2664-9969; E-ISSN 2706-5448

*Conference: Networking in Education and Research (RoEduNet)*. Sovata: IEEE, 1–4. https://doi.org/10.1109/roedunet57163.2022.9921063

**29.** Taiwo Joseph Akinbolaji, G., Nzeako, G., Akokodaripon, D., Aderoju, A. V. (2024). Proactive monitoring and security in cloud infrastructure: leveraging tools like Prometheus, Grafana, and HashiCorp Vault for Robust DevOps Practices. *World Journal of Advanced Engineering Technology and Sciences, 13 (2)*, 74–89. https://doi.org/10.30574/wjaets.2024.13.2.0543

**30.** Mohammed, A. R., Mohammed, S. A., Cote, D., Shirmohammadi, S. (2021). Machine Learning-Based Network Status Detection and Fault Localization. *IEEE Transactions on Instrumentation and Measurement, 70*, 1–10. https://doi.org/10.1109/tim.2021.3094223

✉ ***Bohdan Hinko,*** *Department of Computer Engineering, Taras Shevchenko National University of Kyiv, Kyiv, Ukraine, e-mail: bohdan.hinko37@gmail.com, ORCID: https://orcid.org/0000-0001-6327-6387*

***Oleksandr Boretskyi,*** *PhD, Assistant, Department of Computer Engineering, Taras Shevchenko National University of Kyiv, Kyiv, Ukraine, ORCID: https://orcid.org/0000-0002-2300-6320*

***Vitalii Marianovskyi,*** *PhD, Assistant, Department of Computer Engineering, Taras Shevchenko National University of Kyiv, Kyiv, Ukraine, ORCID: https://orcid.org/0009-0009-4057-5689*

✉ *Corresponding author*