ADVANCES IN CYBER-PHYSICAL SYSTEMS

Vol. 10, No. 1, 2025

USING THE RAFT ALGORITHM TO COORDINATE INTERCEPTOR DRONES IN A UAV DETECTION AND NEUTRALIZATION SYSTEM

Andrii Nyzhnyk[1], Andrii Partyka[1], Michal Podpora[2]

[1] *Lviv Polytechnic National University, 12, S. Bandery str., Lviv, 79013, Ukraine,*
[2] *University of Opole, 48, Oleska, 45-052 Opole, Poland*
Authors' e-mails: *andrii.o.nyzhnyk@lpnu.ua, andrii.i.partyka@lpnu.ua, michal.podpora@uni.opole.pl*

https://doi.org/10.23939/acps2025.01.105

Submitted on 20.04.2025

***Abstract*: This article explores the use of the Raft consensus algorithm to coordinate interceptor drones in systems designed to detect and neutralize unmanned aerial vehicles (UAVs). The modified Raft algorithm has enabled stable and synchronized drone actions, allowing for autonomous target interception. Modeling and simulation confirmed the system's fault tolerance and real-time coordination capabilities. In scenarios involving partial communication failures or drone loss, the system has successfully maintained consensus and continued operation. The proposed architecture has used the Rust programming language to ensure memory safety and concurrency management. The results have provided the effectiveness of using Raft in distributed UAV defense systems, while offering advantages such as leader re-election, log replication, and secure communication channels. The paper also discusses cryptographic enhancements and system resilience to potential cyber threats. This research confirms the applicability of the Raft algorithm for UAV interceptor swarms and gives a foundation for further improvements in autonomous defense systems.**

***Index terms*: raft consensus algorithm, interceptor drones, UAV detection and neutralization, distributed coordination, memory-safe programming, Rust language, real-time autonomous systems, cybersecurity in UAV networks.**

I. INTRODUCTION

In the current context of armed conflicts and rapid evolution of military technologies, unmanned aerial vehicles (UAV) are becoming an important threat to the security of states. From reconnaissance UAVs to barrage munitions and FPV drones, their use on the battlefield necessitates the development of effective defense and countermeasures. Traditional air defense systems, such as man-portable air defense systems, are expensive and not always effective against small and maneuverable drones. In this context, the use of interceptor drones becomes a promising solution, as they can detect and neutralize threats with high efficiency and low operating costs.

Effective coordination of interceptor drones is a critical task, as the success of operations depends on rapid response and coordination in real time. One possible way to solve this problem is to use the Raft algorithm, which ensures consensus in distributed systems [1]. Raft offers robust mechanisms for synchronizing data and actions between system nodes, even in the event of individual component failures, making it an ideal choice for coordinating interceptor drones in combat [2].

The purpose of this study is to develop and evaluate the effectiveness of a drone-interceptor coordination system based on the Raft algorithm. The paper analyzes the principles of the algorithm and presents modeling results that emphasize the effectiveness of the proposed approach in the face of real-world UAV threats.

II. LITERATURE REVIEW AND PROBLEM STATEMENT

The Raft algorithm was proposed as a modern and more robust alternative to the Paxos algorithm, due to which Raft provides a simpler implementation of leadership, voting, and log replication mechanisms than Paxos. These aspects are crucial for coordinating actions in systems with high reliability and responsiveness requirements. A study by Vora et al. in 2023 [1] showed that Raft can provide reliable consensus in systems with frequent failures and high loads, which is especially important for rapid response in the event of a UAV shootdown. Chu et al. (2024) [2] studied the importance of consensus for the coordination of autonomous vehicles, which is similar to the task of coordinating interceptor drones. The Raft algorithm can provide this consensus.

Additionally, Abdorrahimi et al. (2024) analyzed the use of the Raft algorithm in the blockchain, which confirms its reliability and efficiency in distributed systems [3]. This study emphasizes Raft's ability to maintain consensus in complex environments, which is relevant for coordinating drones in combat. In addition, Belous and Krylov (2024) explored ways to minimize network traffic in distributed databases, which also strengthens the argument for using similar approaches in interceptor drone systems [4].

Compared to these works, our approach integrates Raft into an interceptor drone system with enhanced fault tolerance, aiming at combat-grade reliability in contested environments. The proposed model builds upon these foundations and addresses unique constraints of aerial interception in real-time scenarios. Also, this study contributes to the cybersecurity field since the proposed approach uses a memory-safe language (Rust) to address

the common vulnerabilities associated with embedded systems and real-time coordination logic.

III. SCOPE OF WORK AND OBJECTIVES

This study focuses on the design and evaluation of a distributed coordination system for interceptor drones based on the Raft consensus algorithm. The scope of work includes simulation of drone swarms executing coordinated interception missions in hostile environments with potential communication loss or partial drone failure.

The main objectives of this research are as follows: (1) to adapt and evaluate the Raft consensus algorithm in the context of UAV threat response; (2) to implement a prototype drone coordination system using memory-safe Rust programming language; (3) to test the system's fault tolerance, leader election process, and ability to maintain consensus under varying conditions; (4) to propose security-enhanced modifications of the Raft algorithm for use in cyber-physical military systems.

The study also addresses integration challenges, implementation constraints, and real-world applicability for both military and civilian UAV countermeasure scenarios.

IV. CREATING A SYSTEM ARCHITECTURE

The system's architecture involves a distributed network of interceptor drones, each with its own sensors and communication means to interact with other drones and the central server (see Fig. 1). The main task is to quickly detect and neutralize enemy UAVs, while maintaining coherence of actions even in the event of failures or temporary loss of communication.

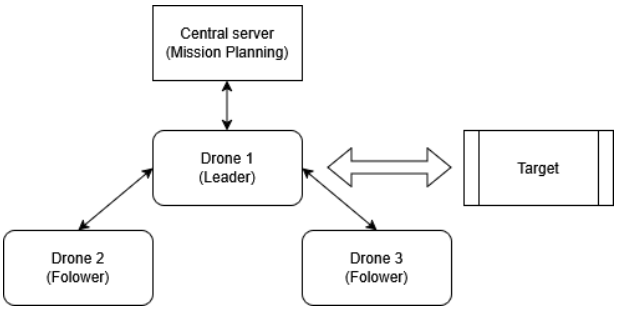

Fig. 1. System architecture of Raft-based UAV interceptor coordination system

The central server collects and processes data, providing basic information about targets and coordinating the initial planning of the operation, while interceptor drones directly detect objects in their range and apply interception means. Each drone constantly exchanges position, status, and threat information, automatically matching roles and tasks in changing environments [5].

An adapted Raft algorithm is used to coordinate the system. One of the drones in the "cluster" acts as a leader, distributing tasks and agreeing on the choice of target. If the leader loses connection with the network, the drones use timers to initiate the election of a new leader based on a majority vote mechanism. Each decision made is recorded in a log that the leader sends to other drones, reflecting the current state of the entire group (including drone positions, battery level, signal strength, etc.). In the event of critical changes, such as switching attention to another target or choosing the best one among several, a voting mechanism is used to maintain the system's stability despite the failure of individual elements. Thanks to this architecture, drones retain high autonomy and can continue to perform combat missions even in the face of significant network failures or lack of communication with the server.

A. HOW THE RAFT ALGORITHM WORKS IN DRONE COORDINATION SYSTEMS

The Raft algorithm solves the problem of achieving consensus in distributed systems, which is critical for real-time coordination of interceptor drones. It is based on a centralized control model: a leader is elected who makes decisions and sends change logs to other nodes. In case of loss of communication with the leader, an election process based on random timers is initiated: nodes that do not receive a signal become candidates and collect the majority vote. This ensures rapid recovery of the system after failures and allows drones to respond quickly to new threats or the disappearance of enemy objects [6].

Raft uses a quorum rule (more than half) to make decisions. A candidate for leadership must receive votes from at least half of the nodes plus one. The same applies to confirmation of operations: new entries in the change logs are considered valid only if they are supported by a majority. This eliminates the "bifurcation" of system states and guarantees consistency, because without a quorum, contradictory records are not moved forward [7]. Also, Raft is usually implemented on an odd number of nodes (3, 5, 7, etc.), which minimizes the risk of an equal distribution of votes [8]. This way, even if a certain number of drones fail or lose connection, the rest can continue to perform the common task.

Raft is fault tolerant: it can function correctly even if up to half of the nodes fail. This is especially important when drones operate in environments with a high risk of damage, as the algorithm allows them to maintain data integrity and maintain uninterrupted coordination. In large clusters, the leader election time can increase, which affects system performance, so it is recommended to optimize timer parameters for scalability [9]. Thanks to the election mechanisms, log replication, and quorum use, Raft is the best choice for building fault-tolerant systems, including UAV detection and neutralization systems.

B. CHOOSING A PROGRAMMING LANGUAGE AND DEVELOPMENT TOOLS

Software for the flight controller or for the control station is written using low-level programming languages. They allow you to use the limited resources of the microcontroller and the system as a whole very efficiently. The disadvantage of this approach is that the developer, having gained full control of the system, can implicitly create an error in the system, which will lead to incorrect

program execution or vulnerability [10]. The most common problem in low-level systems is the problem of proper memory management.

Dynamic memory errors remain one of the most common sources of software vulnerabilities. Attackers actively exploit them to gain unauthorized access to systems, steal data, disrupt software operations, and perform other criminal acts [11]. These types of vulnerabilities are very difficult to reproduce and quickly fix. In high-level systems, this problem is dealt with at the level of the operating system and antiviruses, but unfortunately, it is impossible to run a full-fledged operating system in microcontrollers to counteract these problems.

Despite being a long-standing issue with significant consequences, memory management vulnerabilities remain widespread even in large-scale projects. Reports by Google, Microsoft, Mozilla, and others show that up to 70 % of critical security flaws in software (both proprietary and open-source) are caused by improper memory handling (see Fig. 2).

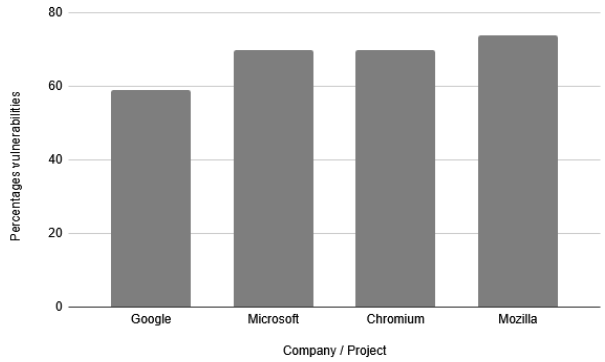

Fig. 2. Percentage of vulnerabilities related to memory management

This data shows that the problem of secure memory management remains acute and requires careful attention from developers and cybersecurity professionals. Therefore, to write software for a drone interception system, you should use Rust or similar programming languages that guarantee secure memory management in low-level systems.

Rust was chosen to write the software for the drone interceptor coordination system because it provides a high level of memory safety and prevents common errors associated with improper management of dynamic memory, such as buffer overflows or working with uninitialized variables. Rust's performance is comparable to languages like C and C++, which is crucial for resource-intensive tasks related to real-time drone coordination. In addition, its strong type system allows many errors to be detected at compile time, reducing the risk of vulnerabilities in the executable code. An important advantage is also the built-in mechanisms for safe parallelism and competition, which allow for the efficient processing of a large number of tasks simultaneously, which is critical for a system that often has to work under a significant load [12].

C. EXPERIMENT SETUP AND EVALUATION

The software development process began with the design of the high-level structure of the system, where the main components and data model for the exchange of messages between the drones and the central server were defined [13].

To evaluate the performance and feasibility of the proposed coordination system based on the Raft algorithm, a series of simulations were conducted using a prototype implementation written in Rust. The simulation environment was designed to emulate a swarm of three interceptor drones operating in a shared airspace, tasked with detecting and neutralizing multiple enemy UAV targets.

Each drone was represented as an independent node running a separate instance of the Raft protocol using the raft-lite library. Communication between drones was modeled over the loopback interface (127.0.0.1) using asynchronous message passing powered by the Tokio runtime. The drones communicated and reached consensus on which enemy target to intercept based on pre-defined conditions (e. g., prioritizing specific UAV models such as Lancet-3).

The available drones and targets list was hard-coded to simulate a realistic scenario with limited coordination time. The primary metric for evaluating the system's correctness was its ability to reach a consensus quickly. A successful consensus was recorded when all nodes decided to intercept the same high-priority target.

The simulation results demonstrated the system's ability to coordinate actions under various conditions, including partial node delays and message handling asynchrony. All three drones successfully reached consensus on the selected target, confirming the system's correctness and fault tolerance within the constraints of the simulated environment. The program code is shown in Listing 1.

```
Listing 1. Program code
use raft_lite::config::{RaftConfig,
RaftParams};
use raft_lite::raft::Raft;
use std::path::PathBuf;

#[tokio::main]
async fn main() {
    // To simplify this example, the HTTP
protocol is used.
    // The protocol is just a transport
layer, so anything suitable can be used.
    // The list of drone interceptors
available in that area
    let drones = vec![
        "127.0.0.1:1024".to_string(), //
drone 1
        "127.0.0.1:1025".to_string(), //
drone 2
        "127.0.0.1:1026".to_string(), //
drone 3
    ];
```

```
    // The list of  enemy drones that must
be intercepted
    let targets = vec![
        b"Zala-421-16".to_vec(), // some
hypothetical target
        b"SuperCam".to_vec(),    // another
target
        b"Lancet-3".to_vec(),    // one
more target
    ];

    for i in 0..drones.len() {
        let drones = drones.clone();
        let targets = targets.clone();
        let mut raft =
get_raft_instance(drones.clone(),
drones[i].clone());

        // Spawn a separate thread to
simulate that Raft can work on different
computing units like drones or servers
        tokio::spawn(async move {
            let (raft_tx, mut raft_rx) =
raft.run();

raft_tx.send(targets[i].clone()).unwrap();

            loop {
                match raft_rx.recv().await
{
                    Some(msg) => {
                        let target =
String::from_utf8(msg).unwrap();

                        if
choose_target_to_attack(&target) {

println!("Consensus is reached. Drone {}
will attack {}", i, target);
                        }
                    }
                    None => break,
                }
            }
        });
    }

tokio::time::sleep(tokio::time::Duration::f
rom_secs(5)).await;
}

fn get_raft_instance(peers: Vec<String>,
self_addr: String) -> Raft {
    let path: PathBuf =
PathBuf::from("./data/drones").join(&self_a
ddr);
    let raft_config =
RaftConfig::new(peers, self_addr,
RaftParams::default(), Some(path));
    Raft::new(raft_config)
}

fn choose_target_to_attack(target: &String)
-> bool {
    // There can be hundreds of conditions
for choosing the appropriate target
    // but let's say we are interested only
in one condition when the target is
'Lancet-3'
    *target == "Lancet-3"

}
```

The result of the program execution is shown in Listing 2.

Listing 2. Result of the program execution

```
/home/andryxaua/.cargo/bin/cargo run --
color=always --profile dev --package
test_raft_for_drones --bin
test_raft_for_drones
       Finished `dev` profile [unoptimized
+ debuginfo] target(s) in 0.20s
       Running
`target/debug/test_raft_for_drones`
Consensus is reached. Drone 1 will attack
Lancet-3
Consensus is reached. Drone 0 will attack
Lancet-3
Consensus is reached. Drone 2 will attack
Lancet-3

Process finished with exit code 0
```

D. SECURITY-ENHANCED RAFT ADAPTATION

Despite the simplicity of the current implementation, it has limitations that warrant further research. Although Raft is an elegant solution for consensus in distributed systems, it assumes non-Byzantine failures – nodes may crash but not behave maliciously. For UAV interceptor systems exposed to cyberattacks, enhanced security is essential. Incorporating cryptographic safeguards allows Raft to handle scenarios where nodes are compromised or communication channels intercepted [14].

An important improvement is ensuring log integrity and authenticity. In standard Raft, the leader appends entries replicated to followers, but forged or altered logs – especially in combat – can break consensus. Therefore, log entries should be digitally signed and verified before acceptance [15]. Periodic hashing or Merkle trees can ensure past data hasn't been tampered with, significantly reducing undetected manipulation.

Leader election is normally based on timers and term numbers, therefore vulnerable to spoofing in hostile environments. Mutual signatures on voting messages ensure legitimacy and verifiable majority support, preventing a compromised node from seizing control [16].

Although Raft lacks native Byzantine fault tolerance, the anomaly detection mechanisms (such as monitoring inconsistent logs or erratic behavior) can help isolate compromised nodes. These measures enhance resilience against active interference.

Secure communication channels are also vital. Encrypting Raft traffic (e. g., via TLS or DTLS) with identity verification prevents impersonation or injection of false data, upholding confidentiality and protecting the consensus process [17].

Finally, effective key management is required. Lightweight approaches like short-lived certificates or certificate pinning support strong authentication without burdening drones' limited resources. Together, these enhancements make Raft more resilient to both reliability failures and sophisticated cyber threats.

E. LIMITATIONS AND DIRECTIONS FOR FURTHER RESEARCH

Despite the demonstrated reliability of the Raft-based coordination system, several important considerations remain for future development. First, real-time constraints and computational overhead present a delicate balance: while cryptographic operations and frequent inter-drone communications bolster security and consensus fidelity, they can also create bottlenecks in time-critical scenarios. Optimizing cryptographic routines and refining the frequency of message exchanges may alleviate delays, particularly on resource-limited UAV platforms.

Another significant factor involves managing hardware constraints. Interceptor drones must often operate under tight energy budgets, while simultaneously supporting flight stability, sensor input processing, and communication tasks. Although Rust's memory safety and asynchronous design improve efficiency, real-world trials could reveal unforeseen interactions – such as competition over CPU or battery resources. Investigating lightweight consensus variants or partial replication schemes could help resolve potential conflicts in these constrained environments.

Equally crucial is the challenge of integrating this system into broader defense and surveillance infrastructures. UAV-based interceptors seldom function in isolation, but rather in tandem with radar networks, data fusion platforms, and advanced decision-making modules. The way UAV-based interceptors work is similar to that of SCADA systems [18]. Ensuring seamless data exchange and interoperability with these external components would enable a more robust air defense ecosystem. Furthermore, because battle conditions can fluctuate dramatically, future research might incorporate adaptive swarm strategies or even self-organizing algorithms that dynamically reassign roles among drones when conditions such as GPS reliability, terrain, or threat density change.

In addition, although the proposed system incorporates basic resilience against compromised nodes, in-depth handling of Byzantine threats remains ripe for exploration. Malicious drones may behave unpredictably, sending conflicting or deceptive information that requires sophisticated anomaly detection or Byzantine fault tolerance techniques. Investigating machine-learning-driven intrusion detection or consensus protocols with partial Byzantine coverage would refine the system's capacity to cope with actively malicious behavior.

Lastly, prolonged field tests under varied environmental and electromagnetic conditions are essential to validate the current design. Laboratory simulations, while informative, cannot fully capture the unpredictability of real-world interference, GPS spoofing, or large-scale electromagnetic disruption. Conducting extended trials with multiple drones in operationally relevant settings would offer invaluable insights, revealing performance thresholds, refining fault-tolerance measures, and confirming the system's viability in genuine high-stakes environments.

V. CONCLUSION

The proposed system based on the Raft algorithm has showed high efficiency in solving the problem of coordinating distributed UAV defense systems. The use of the Raft algorithm allowed for a reliable consensus between drones, which, in turn, increases the overall efficiency and speed of the system's response to threats.

During the study, software architecture was developed that takes into account the specifics of parallel and asynchronous operations that are critical for the operation of distributed systems in real time. The use of the Rust programming language, which ensured memory safety and high performance, avoided typical problems associated with resource management and reduces the risk of errors during program code execution. The simulation results showed that integrating Raft into a drone coordination system not only improved the stability of the system, but also significantly reduced the time required to reach a consensus on target selection. This was especially important in a rapidly changing situation on the battlefield, where every second could be crucial.

Additionally, a series of tests were conducted to confirm the system's ability to operate in difficult conditions, such as partial loss of communication or physical destruction of individual drones. Results showed that the system has a high degree of reliability and can function even in the event of unforeseen circumstances.

The results obtained indicated the potential of the proposed approach for use in real-world conditions, both in the military and civilian protection against UAV threats. Further research should focus on improving the algorithms for prioritizing and optimizing the energy consumption of interceptor drones, increasing efficiency of the system.

References

[1] Vora, S., Thakkar, N., & Gor, R. (2023). A study of performance measures and throughput of Raft consensus algorithm. *International Journal of Research in Applied Science and Engineering Technology,* 11, 862–869. DOI: https://doi.org/10.22214/ijraset.2023.54751.

[2] Chu, D., Zhao, C., Wang, R., Xiao, Q., Wang, W., & Cao, D. (2024). A survey of multi-vehicle consensus in uncertain networks for autonomous driving. *IEEE Transactions on Intelligent Transportation Systems*, 25(12), 19319–19341. DOI: https://doi.org/10.1109/TITS.2024.3465046.

[3] Abdorrahimi, B., Nekouie, A., Rahmani, A., Lansky, J., Nulíček, V., Hosseinzadeh, M., & Moattar, M. (2024). Blockchain technology and raft consensus for secure physician prescriptions and improved diagnoses in electronic healthcare systems. *Scientific Reports*, 14, 2594. DOI: https://doi.org/10.1038/s41598-024-66746-y.

[4] Belous, R., & Krylov, Y. (2024). Minimisation of network traffic in the Raft-like consensus algorithm. *Municipal Economy of Cities*, 4, 2–6. DOI: https://doi.org/10.33042/2522-1809-2024-4-185-2-6.

[5] Zabunov, S., & Mardirossian, G. (2024). Four innovative drone interceptors. *Proceedings of the Bulgarian Academy of Sciences*, 77(2), 238–245. DOI: https://doi.org/10.7546/CRABS.2024.02.09.

[6] Li, Y., Fan, Y., Zhang, L., & Crowcroft, J. (2023). Raft consensus reliability in wireless networks: Probabilistic analysis. *IEEE Internet of Things Journal*, 10(14), 12839–12853. DOI: https://doi.org/10.1109/JIOT.2023.3257402.

[7] Huang, S., Yao, H., Wang, X., Mai, T., Wang, Z., & Guo, S. (2025). Graph attentional based agglomerative cluster for UAV swarm networks. *IEEE Transactions on Network Science and Engineering*, 1–18. DOI: https://doi.org/10.1109/TNSE.2025.3559215.

[8] Kumar, V., Asthana, A., & Tripathi, G. (2025). Enhancing data security in IoT-based UAV networks through blockchain integration. *Engineering, Technology & Applied Science Research*, 15(2), 21800–21804. DOI: https://doi.org/10.48084/etasr.9922.

[9] Mariani, S., Cabri, G., & Zambonelli, F. (2021). Coordination of autonomous vehicles: Taxonomy and survey. *ACM Computing Surveys*, 54(1), 1–33. DOI: https://doi.org/10.1145/3431231.

[10] Abdullayeva, F., & Valikhanli, O. (2024). A survey on UAVs security issues: Attack modeling, security aspects, countermeasures, open issues. *Control and Cybernetics*, 52(4), 405–439. DOI: https://doi.org/10.2478/candc-2023-0044.

[11] Culic, I., Vochescu, A., & Radovici, A. (2022). A low-latency optimization of a Rust-based secure operating system for embedded devices. *Sensors*, 22(22), 8700. DOI: https://doi.org/10.3390/s22228700.

[12] Oorschot, P. (2023). Memory errors and memory safety: A look at Java and Rust. IEEE Security & Privacy, 21(3), 62–68. DOI: https://doi.org/10.1109/MSEC.2023.3249719.

[13] Hai, X., Qiu, H., Wen, C., & Feng, Q. (2023). A novel distributed situation awareness consensus approach for UAV swarm systems. *IEEE Transactions on Intelligent Transportation Systems,* 24(2), 14706–14717. DOI: https://doi.org/10.1109/TITS.2023.3300871.

[14] Yuan, H., Li, F., Diao, R., & Shu, T. (2024). Double□layer Byzantine fault□tolerant grouping consensus algorithm based on Raft. *IET Blockchain*, 4(6), 555–569. DOI: https://doi.org/10.1049/blc2.12073.

[15] Yang, H., Feng, Y., & Zhang, W. (2024). LRD-Raft: Log replication decouple for efficient and secure consensus in consortium blockchain-based IoT. *IEEE Internet of Things Journal*, 12(7), 8807–8820. DOI: https://doi.org/10.1109/JIOT.2024.3506601.

[16] Cui, Y., Liang, Y., Luo, Q., Shu, Z., & Huang, T. (2024). Resilient consensus control of heterogeneous multi-UAV systems with leader of unknown input against Byzantine attacks. *IEEE Transactions on Automation Science and Engineering*, 22, 5388–5399. DOI: https://doi.org/10.1109/TASE.2024.3420697.

[17] Tian, S., Zhang, C., & Bei, G. (2023). VSSB-Raft: A secure and efficient zero trust consensus algorithm for blockchain. *ACM Transactions on Sensor Networks*, 20(2), 1–22. DOI: https://doi.org/10.1145/3611308.

[18] Nyzhnyk, A., Partyka, A., & Podpora, M. (2024). Increase the cybersecurity of SCADA and IIoT devices with secure memory management. *CEUR Workshop Proceedings*, 3800, 32–41.

Andrii Nyzhnyk was born in Lviv region, Ukraine, on October 15, 1997. He received his M.Sc. degree in cybersecurity at Lviv Polytechnic National University, where he is currently pursuing postgraduate studies at the Department of Information Security. His research interests include secure distributed systems, UAV defense technologies, and cyber-physical system resilience.

Andrii Partyka was born in Lviv, Ukraine, on April 14, 1984. He received the Specialist degree in Physical and Biomedical Electronics and the qualification of Physics Teacher at Lviv Polytechnic National University, Lviv, Ukraine, in 2006. From 2006 to 2009, he pursued postgraduate studies. In 2013, he defended his Ph. D. thesis in the specialty of Solid-State Electronics. Since 2010, he has been affiliated with the Department of Information Protection. His research interests encompass cybersecurity, cloud technologies and their protection, security models and access control to cloud resources, ethical hacking, and computer networks

Michal Podpora received an M. Sc. degree in Computer Engineering and a Ph. D. in Control Engineering and Robotics from the Opole University of Technology and in 2021 his habilitation in Cybernetics from the VSB-Technical University of Ostrava. He is currently employed as an Associate Professor with the Institute of Computer Science, University of Opole, as a Humanoid Robot Developer with Research Department at Weegree, and as AI agentic systems consultant. His main research interests include artificial intelligence in robotics, cognitive systems, multiagent knowledge processing, computer vision, smart infrastructure, embedded systems, IoT, and cybersecurity.