RIGA TECHNICAL UNIVERSITY

Dmitrijs Rjazanovs

ANALYSIS OF FAULT-TOLERANT ALGORITHMS FOR THE DEVELOPMENT OF COOPERATIVE MOBILE NETWORKS

Summary of the Doctoral Thesis

RTU Press
Riga 2025

RIGA TECHNICAL UNIVERSITY
Faculty of Computer Science, Information Technology and Energy
Institute of Photonics, Electronics and Telecommunication

Dmitrijs Rjazanovs
Doctoral Student of the Study Programme "Telecommunications"

ANALYSIS OF FAULT-TOLERANT ALGORITHMS FOR THE DEVELOPMENT OF COOPERATIVE MOBILE NETWORKS

Summary of the Doctoral Thesis

Scientific supervisors
Associate Professor Dr. sc. ing.
ALEKSANDRS IPATOVS
Professor Dr. habil. sc. ing.
ERNESTS PĒTERSONS

RTU Press
Riga 2025

Rjazanovs, D. Analysis of Fault-Tolerant Algorithms for the Development of Cooperative Mobile Networks. Summary of the Doctoral Thesis. Riga: RTU Press, 2025, 47 p.

Published in accordance with the decision of the Promotion Council "P- 08" of 27 June 2025, Minutes No. 42.

This research was supported by the European Social Fund within Project No. 8.2.2.0/20/I/008, "Strengthening of PhD students and academic personnel of Riga Technical University and BA School of Business and Finance in the strategic fields of specialization" of the Specific Objective 8.2.2 "To Strengthen Academic Staff of Higher Education Institutions in Strategic Specialization Areas" of the Operational Programme "Growth and Employment".

The research was supported by the Doctoral Grant programme of Riga Technical University.

Cover picture from www.shutterstock.com.

https://doi.org/10.7250/9789934372278
ISBN 978-9934-37-227-8 (pdf)

DOCTORAL THESIS PROPOSED TO RIGA TECHNICAL UNIVERSITY FOR PROMOTION TO THE SCIENTIFIC DEGREE OF DOCTOR OF SCIENCE

To be granted the scientific degree of Doctor of Science (PhD), the present Doctoral Thesis has been submitted for defence at the open meeting of RTU Promotion Council on 28 November 2025 at 11.00 AM at the Faculty of Computer Science, Information Technology and Energy of Riga Technical University, Āzenes iela 12, Room 201.

OFFICIAL REVIEWERS

Professor Dr. sc. ing. Vjačeslavs Bobrovs,
Riga Technical University

Professor Dr. sc. ing. Toledo Moreo Rafael
Polytechnic University of Cartagena (UPCT), Spain

Professor Dr. ing. Habil. Mehmet Ercan Altinsoy
Dresden University of Technology (TUD), Germany

DECLARATION OF ACADEMIC INTEGRITY

I hereby declare that the Doctoral Thesis submitted for review to Riga Technical University for promotion to the scientific degree of Doctor of Science (PhD) is my own. I confirm that this Doctoral Thesis has not been submitted to any other university for promotion to a scientific degree.

Dmitrijs Rjazanovs ……. (signature)
Date: ……

The Doctoral Thesis has been written in English. It consists of an introduction, five chapters, conclusions, 33 figures, 12 tables, and five appendices; the total number of pages is 101, not including appendices. The Bibliography contains 61 titles.

ANNOTATION

This Thesis addresses the complexity of engineering approaches for fault-tolerant mobile networks. The problem originates from conventional IT systems, where, despite the maturity of software development practices, fault tolerance often does not receive adequate attention. This perspective is extended even further to the field of cooperative mobile network solutions. The primary goal of this research is to advance efficient, fault-tolerant solutions for cooperative mobile networks and provide insights into the efficiency of fault-tolerant consensus, exploring its various forms and practical implementation strategies.

The research specifically focuses on measuring the efficiency of eventual and synchronous leader election algorithms in cooperative UAV patrol missions. Insights are provided into the characteristics of the eventual algorithm's convergence process, the effects of different timeout strategies on mission quality, the dynamic characteristics of cooperative UAV networks, and the impact of AI accuracy and performance on mission outcomes. Additionally, the consensus problem is analyzed in the context of smart vehicle cooperative crash avoidance scenarios, exploring whether the Byzantine error model is necessary in V2V communications. A method for modeling and measuring fail-stop flooding consensus performance in this scenario is developed, and the results are analyzed. To achieve these objectives, a novel, real-time, Docker-based simulation was designed and developed. A queuing network model was created to analyze the dynamic characteristics of the cooperative UAV solution, while a custom NS3 simulation was developed to evaluate consensus in V2V scenarios.

CONTENTS

ABBREVIATIONS

A
ACK – Acknowledgement
AI – Artificial Intelligence
API – Application Programming Interface

B
BFT – Byzantine Fault Tolerance
BOLD – Bio-inspired Optimized Leader Election for Multiple Drones

C
C-ITS – Cooperative Intelligent Transport Systems
CDF – Cumulative Distribution Function

D
DT – Delta Timeout

E
ETSI – European Telecommunications Standards Institute

F
FOV – Field of View

G
GNQN – Gordon–Newell Queueing Network

H
HWMP – Hybrid Wireless Mesh Protocol

I
IBM – International Business Machines
ID – Identifier
IOT – Internet of Things
IP – Internet Protocol
IS – Infinite Server

K
KPI – Key Performance Indicator

L
LMT – Latvian Mobile Telephone

M
MVA – Mean Value Analysis

N
NAT – Network Address Translation
NS-3 – Network Simulator 3

O
OS – Operating System

P
PDF – Probability Density Function
PKI – Public Key Infrastructure
p2p – Peer-to-Peer

Q
QN – Queueing Network

S
SIM – Subscriber Identity Module
SSD – Solid State Drive
STUN – Session Traversal Utilities for NAT
SV – Smart Vehicles

T
TCP – Transmission Control Protocol
TURN – Traversal Using Relays around NAT

U
UDP – User Datagram Protocol
UAV – Unmanned Aerial Vehicles
UTF – Unicode Transformation Format

V
V2V – Vehicle-to-Vehicle Communication
V2X – Vehicle-to-Everything
VANET – Vehicular Ad Hoc Network

W
WAN – Wide Area Network

GENERAL DESCRIPTION OF THE DOCTORAL THESIS

This Thesis explores the challenge of identifying and implementing sufficiently robust fault-tolerant algorithms in the context of cooperative mobile networks. Both fault-tolerant algorithms and cooperative mobile networks encompass a vast range of use cases; therefore, this research focuses on selected scenarios with the aim of deriving general insights applicable to other contexts as well. The Thesis presents both theoretical and practical contributions, with a primary emphasis on practical implementation. The first scenario examines cooperative UAV perimeter patrol, including simulation and a leader election algorithm. The second scenario addresses the problem of achieving consensus during emergency situations in smart vehicular networks. The goal of this chapter is to briefly introduce the research problem, provide contextual background, and explain its significance.

Traditionally, the application of fault-tolerant algorithms was discussed and developed in the context of database clusters. There are limits to how far a single database server can be scaled vertically. At some point, multiple servers may be needed to share the load of requests on the database. This introduces the problem of database state replication, which is a form of consensus with numerous variations and implementations [1]. Since the 1970s, many algorithms and practical systems have been developed to address this problem. The theory and practice behind these systems are quite mature; nevertheless, research in this area continues [2].

Distributed systems are a fundamental component of modern technology, supporting a wide range of applications, such as cloud computing, autonomous vehicles, PageRank, smart power grids, state estimation, UAV coordination, multi-agent control, load balancing, and blockchain. Originating from early work on database replication, distributed systems have evolved to support increasingly complex and decentralized architectures. This evolution has significantly influenced both technological development and societal interaction with digital systems. While they enable many of today's autonomous and distributed technologies, distributed systems also continue to present both theoretical and practical challenges [3]. This Thesis aims to examine how core concepts – such as consensus – can vary in implementation and behavior depending on the specific application context, whether in vehicular networks, UAV operations, or permissionless peer-to-peer systems.

Collaborative mobile networks consist of autonomous, mobile agents – such as vehicles, drones, or robots – that interact and coordinate to achieve shared objectives without centralized control. These networks operate in dynamic environments with changing topologies, unreliable communication links, and limited resources. To function effectively, they require decentralized algorithms capable of handling failures, making decisions collaboratively, and adapting to real-time conditions, creating a unique intersection between mobility, autonomy, and distributed fault-tolerant computing.

Autonomous smart vehicles are one of the big challenges for both industry and academia. The operation of autonomous vehicles already relies on distributed consensus mechanisms for critical tasks such as platooning, collision avoidance, and cooperative decision-making, highlighting their role as components of a collaborative network rather than isolated entities [4]. These mechanisms ensure real-time coordination, traffic efficiency, and safety, yet face

significant challenges. Privacy concerns arise from the extensive data sharing required between vehicles, while performance demands necessitate robust, low-latency processing for dynamic conditions. Furthermore, reliability is crucial to prevent system failures that could lead to catastrophic outcomes. Overcoming these challenges is essential to advancing the implementation of secure and scalable distributed systems in interconnected SV ecosystems.

For the past ten years, personal electric UAVs have gained widespread popularity globally. The primary catalyst for this surge is the advancements in the lithium polymer battery industry [5], [6]. Initially, UAVs could only sustain flight for a few minutes; however, the current standard exceeds 20 minutes [7]. Cooperative UAV systems rely heavily on distributed consensus mechanisms to enable critical operations such as formation flying, coordinated search and rescue missions, and dynamic task allocation [8], [9]. These mechanisms ensure real-time coordination, collision avoidance, and efficient resource utilization across a network of UAVs operating in dynamic and unpredictable environments.

Despite the critical role that fault-tolerant primitives play in modern IT and cyber-physical systems, there remains a notable gap in the practical understanding of their theory and application. Even in typical enterprise systems, which are increasingly distributed, basic design mistakes are often made, leading to poor fault tolerance, reduced system reliability, and higher development and maintenance costs [10]. There are several reasons for this. One is that these primitives are usually abstracted away by frameworks and libraries, allowing engineers to use them without understanding the underlying assumptions and limitations. However, in practice, such abstractions rarely provide complete, end-to-end fault tolerance. A common example is the misuse of message brokers. Engineers may assume that posting a message to a broker after a transaction is sufficient, overlooking the fact that network failures can prevent message delivery. Without additional mechanisms, such as the outbox pattern [11], data consistency and message reliability cannot be guaranteed. Another contributing factor is that systems are often designed with a focus on the “happy path,” while failure scenarios are overlooked, especially in later development phases when budget and time are limited.

Aim and Tasks of the Doctoral Thesis

Considering the above-mentioned facts, **the objective of the Doctoral Thesis** has been set. The objective of this research is to investigate existing consensus algorithms, assess their applicability in advanced cooperative mobile network scenarios, and develop simulations to evaluate their performance under harsh conditions for potential development in real deployments.

To accomplish this goal, the following main tasks were defined:

1. Conduct an in-depth study of distributed systems' fundamental principles, including fault-tolerant algorithms, network models, and boundary results in consensus theory.
2. Investigate the latest advancements in distributed systems, including blockchain-based consensus algorithms, and evaluate their applicability in mobile network environments.

3. Research and apply queueing theory as a mathematical foundation for evaluating network performance and algorithm efficiency in mobile networks.
4. Explore the theoretical foundations and practical implementations of UAV-based mobile networks, including mission planning, communication, and cooperative scenarios.
5. Investigate V2X communication, covering both theoretical foundations and practical technologies used in smart vehicular networks.
6. Evaluate existing simulation frameworks and methods for modeling cooperative UAV missions, focusing on flexibility, scalability, and network reliability.
7. Design and develop a simulation environment for cooperative UAV missions, capable of testing various fault conditions, including UAV crashes, unstable networks, and delayed communication.
8. Develop a methodology to compare synchronous and partially synchronous consensus algorithms in the context of cooperative UAV perimeter patrol.
9. Design and implement a simulation model for consensus in vehicular networks (V2V), focusing on fault-tolerant agreement under crash-stop conditions.
10. Conduct extensive experiments using the developed simulations, analyze the results, and derive conclusions regarding the practical applicability of consensus algorithms in real-world deployments.

Research Methodology

During the completion of the defined tasks, the theoretical correctness, performance, assumptions, and guarantees of distributed algorithms were thoroughly analyzed. This analysis was combined with software design principles to develop simulations and create various scenarios for quantitatively comparing the effectiveness of distributed systems. Mathematical calculations were conducted to support the simulation results, and field experiments were carried out to validate network behavior under real-world conditions.

To evaluate NAT traversal in the field, the following technologies were employed: the C# programming language, Azure cloud virtual machine services, an Ubuntu-based laptop, a Windows-based laptop, the UDP network stack, and SIM cards from multiple mobile operators. This setup enabled comprehensive testing of NAT traversal and peer-to-peer connectivity under different network conditions.

For the implementation of the cooperative UAV patrol mission simulation, Docker was utilized for containerized deployment, while the simulation logic was developed using C# and .NET 8, with the XUnit testing framework ensuring reliability and repeatability of results. The simulation modeled network behavior using a log-normal probability distribution to represent transmission delays, and a sample size calculation formula was used to determine the minimum number of simulation runs required for statistically significant results. Core cooperation mechanisms were built based on eventual and perfect failure detectors, as well as eventual and perfect leader detectors, which provided a robust foundation for UAV coordination.

Python, in combination with the Storybook and VS Code environment, was used for processing simulation results and calculating the dynamic characteristics of the UAV network. The mathematical foundation was based on a closed Gordon–Newell queueing network (QN) model, which defined the UAV network's key components. Buzen's convolution algorithm was employed to calculate throughput, utilization, and other essential properties of the QN. Furthermore, a custom Python script was developed to calculate the probability distribution of QN response times using the QN normalization constant.

The NS-3 framework was applied to simulate all-to-all custom UDP and HWMP-based broadcast algorithms within an 802.11s mesh network. Different connectivity graph diameters and topologies were tested to determine the average all-to-all broadcast delay. The collected data was subjected to polynomial regression analysis, resulting in a third-degree polynomial approximation of the delay function, which depended on the number of nodes and graph diameter. The flooding consensus algorithm was employed to model and calculate the expected value of consensus delay in the presence of node crashes.

Research Results and Scientific Novelty

Contributions of the Doctoral Thesis

1. A 2D, extensible, real-time simulation platform for cooperative UAV missions was developed, supporting multiple control modes and designed for stress testing system resilience under conditions such as network instability, node crashes, and delayed communication. The platform is modular and can be reused as a basis for practical solutions.
2. A method for quantifying the impact of eventual algorithms using key performance indicators (KPIs) was proposed and applied. This approach allows for a direct comparison of algorithm performance based on measurable metrics.
3. The performance of a perfect leader detection algorithm with a conservative timeout was compared to an eventually perfect leader detector in the context of a cooperative UAV perimeter patrol mission, providing data on their reliability and convergence behavior.
4. A closed Gordon-Newell queueing network (QN) model was developed to assess the trade-off between centralized AI service response time and onboard AI accuracy in cooperative UAV patrol missions. The model allows for a direct comparison of system performance under different configurations.
5. A custom probabilistic UDP and HWMP-based all-to-all broadcast protocol was designed and simulated in the NS-3 environment. Simulation results were used to model the behavior of a flooding consensus algorithm under conditions of missing communication links.

Main Conclusions of the Doctoral Thesis

1. When measuring the impact of fault-tolerant algorithms, it is advisable to first develop optimal decision logic, as this significantly influences mission KPI, even in scenarios with extensive faults.
2. Real-time simulation is invaluable for identifying bugs that would otherwise appear during real-world deployment, where fixing them can be significantly more costly. The primary drawback of real-time simulation is its execution time. The hybrid approach – using discrete simulation for overall performance testing and real-time simulation for advanced corner cases in later project phases – can help achieve maximum fault tolerance and resilience.
3. Simulation data indicate that the cumulative number of false crash detections caused by the eventual leader detector algorithm during simulation follows a logarithmic pattern. As the number of deployed UAVs increases, the time until the last false crash detection also increases significantly. With more than five UAVs and a larger detection timeout (DT), false crash detections can continue for up to 10 minutes. Nevertheless, due to the logarithmic nature of this behavior, most false crashes occur quickly, and only a small portion of total false negatives affects KPIs.
4. In cooperative UAV perimeter patrol mission simulations, quantitative KPI metrics are valuable for comparing leader election algorithm performance and for determining whether eventually consistent algorithms are suitable for cooperative UAV scenarios.
5. In the perfect leader detector algorithm, extending the crash detection timeout from 15 seconds (ideal) to 60 seconds results in a 12.1% KPI reduction during a five-minute simulation with a 30 % UAV crash rate. The eventually perfect leader detector, configured with a 5-second base crash detection timeout and a 3-second delta, shows a 4.1 % KPI loss compared to the perfect algorithm with the ideal timeout. Notably, the eventually perfect algorithm outperforms the perfect algorithm under realistic conditions (60 s timeout), achieving a 9.7 % KPI improvement at a 30 % crash rate.
6. A UAV surveillance network employing AI-based threat detection can be modeled using a closed Gordon-Newell queueing network. While improvements in onboard AI accuracy reduce time lost to false positive monitoring linearly, reducing the central AI's response time has a greater impact on minimizing lost monitoring time under increasing system load.
7. The Byzantine error model is widely considered in avionics and space engineering. However, this term is never mentioned in ETSI ITS technical reports. A literature review of ITS reveals that Byzantine fault-tolerant algorithms proposed for V2X systems primarily focus on privacy, blockchain-based architectures, and reputation mechanisms.
8. A custom, randomized, UDP-based, reliable, all-to-all broadcast algorithm using 802.11s mesh, tested with NS3 simulation, shows an average delay of 280 ms for 33 nodes with a connectivity graph diameter of 10. For smaller diameters and node counts, average delay is below 100 ms.

9. Mathematical modeling shows that the fail-stop flooding consensus algorithm, based on the aforementioned broadcast algorithm, can theoretically achieve consensus in under 500 ms for $D = 8$, $N = 11$, with up to 30 % node crashes, and under 200 ms for smaller diameters. The shape of the reliable broadcast delay curve depends on the communication graph's diameter: for $D = 2$, delay increases with node count, while for $D \geq 7$, the growth rate decreases. A more linear trend is observed around $D = 4$.

Practical Value of the Doctoral Thesis

1. UDP NAT traversal software has been developed to establish autonomous UAV networks within cellular networks. Field experiments to test NAT traversal across various cellular network providers have been conducted.
2. The author designed and implemented a UDP-based fault-tolerant communication middleware for cooperative UAV missions. A 2D, real-time, extensible simulation framework was built for cooperative UAV missions, focusing on fault tolerance testing.
3. The author discovered through simulations that the convergence time of the eventual leader detector algorithm exhibits a logarithmic relationship with the delta parameter and the number of UAVs. The relationship between mission KPIs and the number of UAVs was identified through simulation analysis.
4. The author analyzed how different configurations of eventual and perfect leader detector algorithms impact mission KPIs, offering guidance for selecting suitable solutions for cooperative UAV missions.
5. A mathematical model to inform the design of software and hardware setups for cooperative UAV networks has been designed.

The results of the Thesis scientific research can be used for the development of cooperative UAV and other mobile node mission and service solutions.

Theses to be Defended in the Doctoral Thesis

1. Using a novel real-time 2D simulation of a cooperative UAV-based perimeter threat search system, with a UDP-based communication stack and an unstable network modeled using log-normal delays ($\mu = 6$, $\sigma = 1.5$), it is possible to evaluate the efficiency of leader election strategies. Experiments show that eventual election with an adaptive crash detection timeout improves mission KPI by 9.7 % compared to static conservative timeouts.
2. Using a closed Gordon–Newell queueing network model of cooperative UAV threat search missions, it is possible to show that reducing the response time of the remote AI service minimizes wasted UAV time from misclassifications more effectively than improving onboard AI accuracy.

3. Using a novel NS-3 simulation and a UDP-based probabilistic reliable broadcast, experiments show that the fail-stop flooding consensus algorithm can achieve consensus in 99 % of cases in under 500 ms on an 802.11s mesh with 8 hops and 11 nodes, with up to 30 % node crashes, and under 200 ms for smaller diameters.

Approbation of the Research Results

The main results of the Doctoral Thesis have been presented at 10 international scientific conferences, as well as reflected in 2 publications in scientific journals and in 8 publications in full-text conference proceedings.

Publications in full-text conference proceedings and scientific journals

1. **Rjazanovs, D**., Pētersons, E., Ipatovs, A., Juškaite, L., Yeryomin, R. "Byzantine Failures and Vehicular Networks." 2021 IEEE Microwave Theory and Techniques in Wireless Communications (MTTW). Latvia, Riga, 2021. pp. 30–34.
2. **Rjazanovs, D**., Petersons, E. "Decentralized Byzantine Fault-Tolerant Proof of Location." PoEM Workshops, 2020. pp. 1–10.
3. **Rjazanovs, D**., Ratkuns, A., Kārkliņš, T., Nagla, I., Ipatovs, A. "Making Ping Feint to Avoid Service State Desynchronization." 2023 Workshop on Microwave Theory and Technology in Wireless Communications (MTTW). 2023. pp. 34–38.
4. **Rjazanovs, D**., Grabs, E., Pēterson,s E., Aleksandrovs-Moisejs, D., Chen, T., Čulkovs, D., Aleksandrovs, M., Ipatovs, A. "Drone Cooperation, NAT Traversal, and Performance." 2024 Photonics & Electromagnetics Research Symposium (PIERS). 2024. pp. 1–6.
5. **Rjazanovs, D**., Grabs, E., Pētersons, E., Lahs, A., Kopats, A., Kārkliņš, T., Aleksandrovs-Moisejs, D., Titovics, J., Chen, T., Čulkovs, D., Aleksandrovs, M., Ipatovs, A. "Analysis of Leader Election Algorithms in the Context of Cooperative UAV Mission Simulation." 2025 Photonics & Electromagnetics Research Symposium (PIERS). 2025.
6. Aleksandrovs-Moisejs, D., Ipatovs, A., Grabs, E., **Rjazanovs, D**. "Evaluation of a Long-Distance IEEE 802.11 ah Wireless Technology in Linux Using Docker Containers." Elektronika ir Elektrotechnika, Vol. 28, No. 3, 2022, pp. 71–77.
7. Chen, T., Grabs, E., Petersons, E., Efrosinin, D., Ipatovs, A., Bogdanovs, N., **Rjazanovs, D**. "Multiclass Live Streaming Video Quality Classification Based on Convolutional Neural Networks." Automatic Control and Computer Sciences, Vol. 56, No. 5, 2022, pp. 455–466.
8. Aleksandrovs-Moisejs, D., Ipatovs, A., Grabs, E., **Rjazanovs, D**., Sinuks, I. "Arduino-based Temperature Sensor Organization and Design." 2023 PhotonIcs & Electromagnetics Research Symposium (PIERS). 2023. pp. 1408–1415.
9. Andrejevs, D., Grabs, E., Čulkovs, D., Jeralovičs, V., Juškaite, L., Chen, T., **Rjazanovs, D**., Ipatovs, A. "Development of Laser Communication Algorithm for

Moving Objects." 2024 Photonics & Electromagnetics Research Symposium (PIERS). 2024. pp. 1–4.

10. Klūga, J., Grabs, E., Chen, T., Ancāns, A., Stetjuha, M., **Rjazanovs, D**., Ipatovs, A. "Motion Sensors Data Fusion for Accurate Measurement in AHRS Systems." 2024 Photonics & Electromagnetics Research Symposium (PIERS). 2024. pp. 1–4.

Scope and Structure of the Doctoral Thesis

The total number of pages of the Doctoral Thesis is 101. It contains five chapters, conclusions, a bibliography, and five appendices.

Chapter 1 begins with a brief introduction to distributed systems theory, covering its history, fault tolerance, fundamental building blocks, and complexity. It then explores the phenomenon of how applications and theories have evolved over recent decades, naturally leading to the emergence of mobile network-based distributed systems. The chapter emphasizes the critical importance of these systems in the modern world, highlights the significantly higher complexity of mobile networks compared to conventional IT systems, and discusses the paradox of low-quality fault tolerance implementation in standard engineering practices. Finally, the chapter provides all the necessary foundations for the following sections, including a review of existing literature and the problem formulation.

Chapter 2 is entirely devoted to the simulation developed for the main experiments of this Thesis. It begins by discussing the requirements defined specifically for this research, covering both functional and non-functional aspects, as well as exploring options for reusing existing simulation platforms. Next, it addresses communication-related topics, including NAT traversal, communication protocols, and network delay simulation. The chapter then delves into the system architecture, focusing on layering and modularity aspects. Finally, it presents the testing strategy, statistical analysis, and conclusions.

Chapter 3 focuses on the problem of leader election algorithms in the context of cooperative UAV perimeter patrol. Specifically, two algorithms are analyzed and compared: perfect leader detection and eventual leader detection. The primary objective is to quantify the impact of using the eventual algorithm on mission performance. Various experiments are conducted using the simulation platform described in the previous chapter. The results are presented with an emphasis on the effects of the eventual algorithm's convergence time and identifying which algorithm performs better in specific scenarios.

Chapter 4 focuses on a specific aspect of cooperative UAV patrol – the use of a hybrid AI threat detection approach and how improvements to each component impact mission KPIs. It presents the proposed closed Gordon–Newell QN model, detailing the system components, parameters, and the approach used to calculate the response time probability distribution.

Chapter 5 addresses the smart vehicle emergency consensus problem and examines whether the Byzantine error model is applicable in V2X scenarios. It begins with a survey of existing literature on the application of the Byzantine model in both V2X and other domains. The second part focuses on calculating the fail-stop flooding consensus execution time in emergency crash avoidance V2V scenarios. It provides a detailed description of the NS-3 simulation setup, which

includes an all-to-all 802.11s mesh network and UDP-based probabilistic reliable broadcast. The broadcast execution time results are approximated and used to model the consensus execution time.

The Conclusions of the Thesis summarize the main findings of the research and define future research directions. The appendices include lists of conferences, publications, and projects, present the results obtained in experimental measurements and provide the simulation codes.

CONTENT OF THE DOCTORAL THESIS

Chapter 1

The goal of the first chapter is to briefly introduce the research problem, provide contextual background on the fault tolerance and mobile networks, and explain its significance.

Section 1.1 summarizes classifications from distributed systems theory. Engineers naturally focus on the primary use case, addressing the goals and core functionalities that the system must achieve. Once the primary functionality is implemented, there is frequently little time or inclination left to consider negative scenarios and corner cases. Despite the extensive theoretical knowledge about faults, this tendency to overlook negative scenarios is highlighted by the continued relevance of "The Fallacies of Distributed Computing" [12]. These fallacies demonstrate the persistent underestimation of failure modes in distributed systems.

There are five classes of errors as classified by [13]: crash faults, omission faults, crash with recovery, eavesdropping and Byzantine faults. The Thesis research focuses on crash faults and Byzantine faults. Based on that, the following algorithm classes are derived: fail-stop, fail-silent, fail-noisy, fail-recovery, fail-arbitrary and randomized. The research will use fail-stop, fail-noisy and randomized algorithms.

Section 1.2 discusses leader election and consensus algorithms in existing literature. Leader election is a foundational problem in distributed systems, concerned with selecting one distinguished process (or node) among a set of distributed and potentially identical processes [14]. To be effective, leader election protocols must ensure that all correct processes eventually agree on the same leader, and that this leader is unique and active. The classical formulation of the problem includes three desirable properties: **uniqueness** (only one leader at a time), **agreement** (all correct processes agree on the leader), and **termination** (a leader is eventually elected) [15]. To make leader election feasible in partially synchronous systems [16], researchers introduce **failure detectors**, which act as abstract modules providing (possibly unreliable) information about process failures. Two well-known classes of failure detectors are the **perfect failure detector (*P*)** and the **eventually perfect failure detector (*◇P*)** [13]. Two leader election strategies are investigated:

1. A leader election algorithm based on a **perfect failure detector (*P*)**, which ensures accurate detection of process crashes.
2. A leader election algorithm based on an **eventually perfect failure detector (*◇P*)**, which may initially suspect correct processes but stabilizes over time.

All leader election algorithms can be split into two main categories: those that assume a synchronous network and those that operate in a partially synchronous network. Each category contains various implementations. Leader election has its roots in network components, such as IBM's token ring [17], where computers are connected in a physical ring or star topology.
A classic example is the Bully algorithm [18], which assumes a general topology and a synchronous network, relying on static timeouts to detect a dead server. Another example in database clusters is the Raft algorithm [19], where a leader is elected to ensure efficient state machine replication (total order broadcast). Unlike the Bully algorithm, Raft assumes a partially

synchronous network, but it requires that more than half of the nodes be online for the algorithm to function correctly.

In [20], the author proposed a Raft-based leader election and clustering scheme that is resilient to UAV failures. Similarly, the authors of [21] present a simulation-based study of a voting-based leader election scheme for UAV swarms in a lead-follow configuration under constrained communication. Mousavi et al. [22] addressed coalition formation in UAV networks using a multi-objective optimization algorithm based on quantum evolutionary techniques. Their work focused on optimizing cost, trust, and reliability when assigning UAVs to task coalitions. Ganesan et al. [23] proposed the BOLD algorithm, which combines particle swarm optimization and spider monkey heuristics to elect energy-efficient cluster heads in dynamic UAV networks. Wang et al. [24] introduced a secure UAV-aided cluster head election mechanism aimed at protecting wireless sensor networks from compromised nodes.

In general, prior work on leader election in UAV systems can be divided into three broad categories: (1) optimization-based leader selection focused on energy or resource efficiency; (2) secure or supervised schemes that assume centralized control or perfect observation; and (3) consensus-based models that describe correctness but are evaluated only under ideal conditions.

Also, this section discusses existing consensus algorithms and their properties. Consensus is a fundamental problem in distributed computing, where a group of processes must agree on a single value despite failures and communication delays. Formally, a consensus algorithm must satisfy the following properties [15]: **agreement** (no two correct processes decide differently), **validity** (if all correct processes propose the same value, it must be the decided value), **termination** (every correct process eventually decides).

In vehicular networks, especially vehicle-to-vehicle (V2V) scenarios, consensus is used for safety-critical coordination such as collision avoidance or cooperative braking. These scenarios impose strict real-time requirements, typically under 500 ms, and must tolerate communication delays and unreliable links [4]. This study focuses on crash-stop consensus in synchronous systems, where message delays are bounded and failures are reliably detected, as required by the flooding consensus algorithm [13].

Chapter 2

The goal of **Chapter 2** is to comprehensively present the design and development of the cooperative UAV mission functionality and its corresponding real-time simulation framework. Moreover, the functionality was developed with reusability in mind, allowing it to be integrated into actual UAV software.

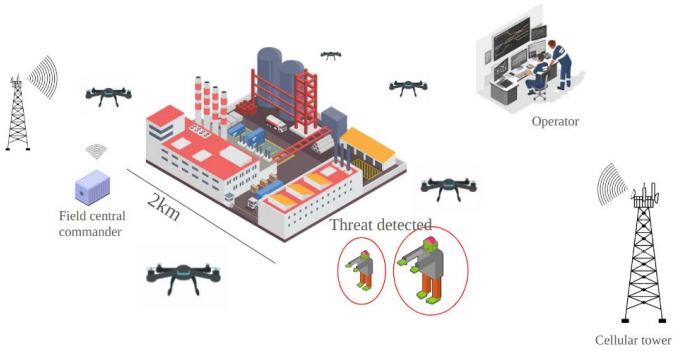

Fig. 1. Cooperative UAV patrol solution concept.

Section 2.1 describes the requirements for the simulation, functional description, simulation types and technological stack. The primary goal is to develop an extensible platform for experimenting with various cooperative UAV scenarios. The main focus of the simulation is to provide a framework for testing the system's fault tolerance, with the aim of enabling deployment to real-world infrastructure with minimal friction.

The section describes the use case and core functionality of the system (Fig. 1). The scenario assumes the presence of large facilities, typically located in suburban or remote areas. These may include factories, power stations, airports, ports, strategic infrastructure, military sites, or other high-value assets. A typical perimeter length ranges from 1 to 20 kilometers. The primary objective is to detect potential threats appearing along the perimeter in real time. The central idea is that a fleet of UAVs – potentially up to 50 units – can be deployed to autonomously partition the surveillance area and continuously monitor for potential threats. While the specific hardware and infrastructure required to support such a mission are beyond the scope of this research, several general assumptions are made [25], [26]. Each UAV is equipped with onboard AI capable of real-time object classification. Additionally, a centralized processing unit is assumed to be available for more advanced AI-based analysis, allowing for higher-confidence threat detection and classification.

The core principle guiding this research is that reliability and security must be built into the system architecture from the outset through fault-tolerant design. While different architectural approaches will be discussed later in this section, the focus of this research is on a *leader-based* architecture. In this model, one UAV is elected as the leader among all participating units. The leader is responsible for: collecting location data from follower UAVs; processing event reports (such as threat detection or disappearance); calculating the optimal mission strategy; decomposing it into specific tasks; and multicasting those tasks to the appropriate UAVs for execution. For example, the leader must divide the perimeter into segments and assign each segment to the most suitable UAV for patrol. The leader-based architecture strikes a balance between fault tolerance and mission efficiency. However, as with any UAV, the leader itself is susceptible to failure. In such cases, the remaining UAVs must elect a new leader. The challenge lies in ensuring that all UAVs agree on the same leader – otherwise, the mission may become

fragmented or inefficient. This constitutes a classic *consensus* problem, previously introduced, and like all distributed algorithms used in this context, the leader election mechanism must be fault-tolerant.

Further, the chapter analyzes the choice between deterministic and real-time simulation. A deterministic approach offers three key advantages: (1) it guarantees reproducibility (running the simulation with the same seed will always result in the same sequence of actions and outcomes); (2) the simulation can be executed significantly faster than real time, limited only by available hardware resources; and (3) debugging is greatly simplified. However, this type of simulation also has several limitations: (1) the granularity of simulation steps must be defined manually, (2) integration with real network components is not feasible due to the deterministic nature of the simulation.

The second type of simulation is a real-time system with a real communication stack. In this mode, the program operates as if actual UAVs are deployed in the field, capturing realistic parallelism and network behavior. This approach enables the simulation of true concurrency, allowing issues that would arise in a real-world deployment to be detected during testing – thereby significantly reducing development and quality assurance costs. Real-time, Docker-based and .NET 8 (C#) simulation approach was chosen, because it allows for comprehensive testing of fault-tolerant middleware and re-use program modules in real deployment further (Table 1).

Table 1

Comparison of the Simulation Frameworks

Feature/requirement	*OMNeT++*	*NS-3*	*SimPy*	*Gazebo*	*AirSim*	*Custom C# & Docker*
Real UDP protocol support	Using plugin	No	No	No	No	Yes
Non-deterministic concurrency	No	No	No	No	No	Yes
Scalability	High	High	Medium	Medium	Low	High
Customizability	High	High	High	Medium	Medium	Very High
Repeatability	Yes	Yes	Yes	Yes	Yes	Partial
Ease of integration with hardware	No	No	No	Yes	No	Yes

Section 2.2 discusses problems related to the connection establishment between UAVs. There are several communication protocol options available for coordinating cooperative UAVs. In the envisioned real-world deployment, cellular communication is selected as the primary medium for several practical reasons. First, cellular networks provide extensive coverage, including in many rural and remote areas. Second, they offer a ready-to-use infrastructure with high throughput, eliminating the need for dedicated communication systems.

To initiate a mission, the operator must configure each UAV with a unique identifier and an associated SIM card. However, a key limitation of cellular networks is that they do not support direct device-to-device communication by default, due to the absence of public IP addresses for client devices. While many studies treat the cellular network as a primary or supplementary communication channel, few delve into the challenge of establishing p2p communication in this context [26], [27], [29]. Cellular infrastructure is not designed for each device to act as a server. Despite this limitation, many smartphone applications – including messaging platforms – successfully enable peer-to-peer communication. This is achieved through the use of NAT traversal techniques, which allow devices behind private IPs to establish direct communication channels (Fig. 2).

There are different NAT types: (1) full cone NAT, (2) restricted cone NAT, (3) port-restricted cone NAT, and (4) symmetric NAT. The last type is the most restrictive and complex. It generates a unique mapping for each combination of internal source and external destination address and port. The only way to establish a p2p connection when both devices have symmetric NAT is through the usage of a relay server [28]. For all other types, the UDP hole punching technique can be used.

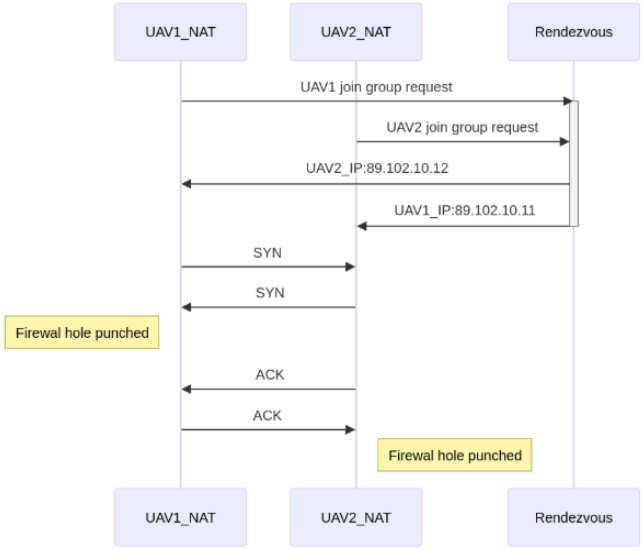

Fig. 2. UDP NAT traversal algorithm sequence diagram.

A special “Rendezvous” server was implemented within the simulation. The goal is to allow UAVs to establish a p2p connection between each other. It implemented UDP hole punching and can be deployed in a real environment straightaway.

Additionally, some experiments were conducted in the field to test how various cellular network providers configure their NAT devices. **Experiment 1** involved a configuration with one LMT (Latvian Mobile Telephone) cellular client and one Azure cloud server with a public IP address. This setup was successful, confirming that outbound UDP traffic from the LMT client could reach the public server and receive responses, with NAT mappings persisting long

enough to enable stable communication. **Experiment 2** tested direct communication between two LMT cellular clients located within close physical proximity. Despite both devices receiving distinct public IP addresses from the operator, the attempt failed. This result suggests that NAT or firewall policies applied by LMT restrict peer-to-peer UDP connectivity – possibly due to symmetric NAT behavior or additional security mechanisms implemented by the operator. **Experiment 3** paired one LMT client with one BITE cellular client. Under similar conditions to the previous experiment, UDP hole punching was successful. This indicates that at least some combinations of mobile network operators support direct UDP communication, likely due to differences in NAT configuration or firewall policies between providers. As an alternative to NAT traversal, some cellular network operators offer SIM cards with publicly routable IP addresses for an additional fee. This approach simplifies peer-to-peer communication and also enables the use of TCP, which may be preferable in scenarios requiring reliable delivery and connection state management.

Further, the chapter is devoted to the modeling and implementation of the realistic network behavior – packet delays and drops. All communication between UAVs during the simulation goes through a separate custom UDP proxy container. Special headers are used by UAVs that instruct the proxy server on the original destination of the UDP packet. The proxy itself is using a configurable log-normal distribution to sample random packet delays (which implies reordering) and random drops. The throughput of the proxy was measured and reached 350 packets/sec with an average size of 250 bytes.

Implements:
EventuallyPerfectFailureDetector, **instance** $\diamond\mathcal{P}$.

Uses:
PerfectPointToPointLinks, **instance** *pl*.

upon event $\langle\ \diamond\mathcal{P}, Init\ \rangle$ **do**
$alive := \Pi$;
$suspected := \emptyset$;
$delay := \Delta$;
starttimer(*delay*);

upon event $\langle\ Timeout\ \rangle$ **do**
if $alive \cap suspected \neq \emptyset$ **then**
$delay := delay + \Delta$;
forall $p \in \Pi$ **do**
if $(p \notin alive) \wedge (p \notin suspected)$ **then**
$suspected := suspected \cup \{p\}$;
trigger $\langle\ \diamond\mathcal{P}, Suspect \mid p\ \rangle$;
else if $(p \in alive) \wedge (p \in suspected)$ **then**
$suspected := suspected \setminus \{p\}$;
trigger $\langle\ \diamond\mathcal{P}, Restore \mid p\ \rangle$;
trigger $\langle\ pl, Send \mid p, [\text{HEARTBEATREQUEST}]\ \rangle$;
$alive := \emptyset$;
starttimer(*delay*);

Fig. 3. Eventually perfect failure detector – increasing timeout algorithm [13].

Section 2.3 describes the architecture of the simulation. The main components are depicted in Fig. 4. At the core of the platform is the **World** module, which simulates a two-dimensional virtual environment. This module is responsible for updating UAV positions, injecting threats, and recording the simulation state on a frame-by-frame basis. It also produces a simplified video stream, where each frame is rendered using a UTF-16 character-based grid, enabling lightweight visualization directly in the console. This same 2D symbolic representation is used

to emulate UAV vision, wherein each UAV perceives its environment through a localized window rendered as a matrix of UTF-16 symbols.

All simulation output is written to a **simulation output file**, which can later be visualized using a **simulation recording visualizer**. This tool replays the simulation using the same UTF-16 console-based rendering engine, allowing for efficient post-experiment inspection without the need to re-run the full simulation.

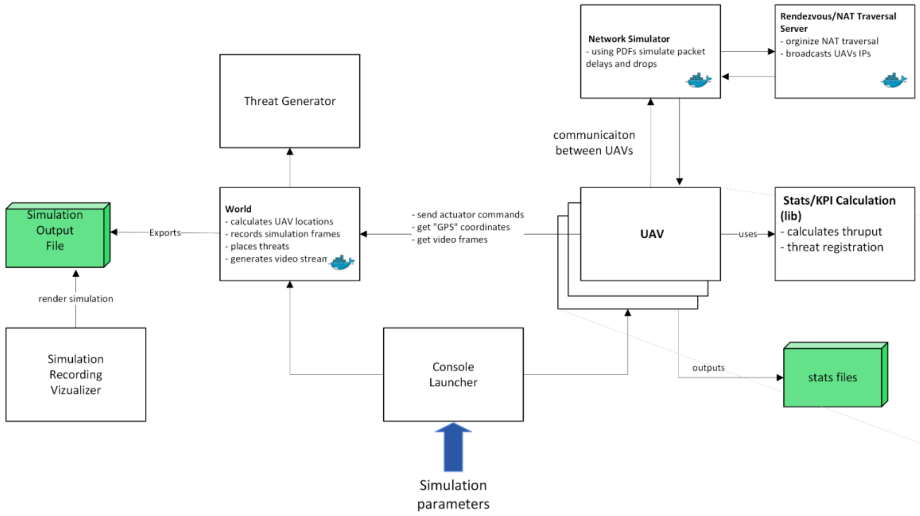

Fig. 4. Architecture of the simulation platform.

To evaluate the effectiveness of UAV behavior and coordination strategies, the simulation tracks several key performance indicators (KPIs). These metrics assess how well threats are detected, monitored, and controlled, as well as the efficiency of UAV communication.

- **T. reg. (%) – threat registration rate.** This metric represents the percentage of threats successfully registered by UAVs during the simulation. A threat is considered registered if it is detected by a UAV during patrol. The value is calculated as the number of registered threats divided by the total number of threats generated.
- **T. lifetime mon. (%) – threat monitoring time.** This KPI measures the proportion of threat lifetimes that were actively monitored by UAVs. Each threat has a predefined lifetime, and any time that is being monitored by a UAV is counted toward this metric. It reflects the effectiveness of continuous surveillance.
- **T. contr. (%) – threat control rate.** Once a threat is registered, the system may assign two UAVs to monitor it simultaneously – a condition referred to as "control." This metric captures the percentage of threats that, at any point during the simulation, reached this controlled state.
- **T. lifetime contr. (%) – controlled lifetime.** This KPI measures the percentage of the total threat lifetimes during which all threats were under control, meaning each had at

least two UAVs assigned. It indicates how consistently the system maintained full coverage over active threats.

Fig. 5 depicts an example simulation run with a small perimeter.

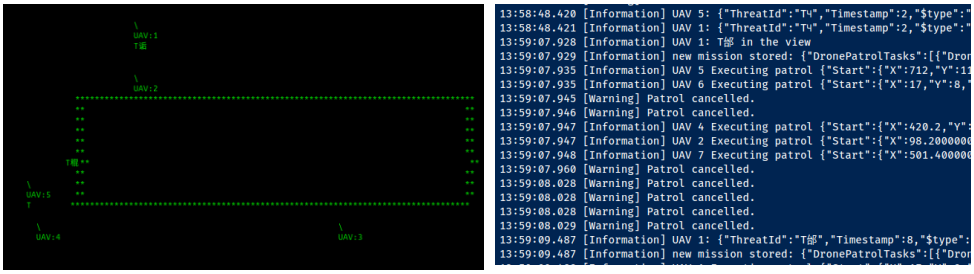

Fig. 5. Example of the simulation visualizer (left) and real-time logs (right).

The simulation platform presented enables realistic testing of UAV coordination strategies, bridging simulation and real-world deployment. Unlike NS-3, which is suited for protocol testing, this platform allows seamless transition from simulation to physical UAVs, ensuring consistent development. Key insights include the critical impact of concurrency issues, the importance of real-time execution for uncovering timing bugs, and the limitations of real-time simulation due to its duration. Docker provided network realism rather than memory isolation, and fixed seeds improved test repeatability, but could not ensure full determinism. The platform effectively models delays and packet loss, making it a robust tool for UAV system testing and development.

Chapter 3

This chapter presents the results of research on how the eventual leader election influences the performance of cooperative missions. **Section 3.1** formulates the problem, describes the mission control mode used in the simulation and describes fault-tolerant primitives used in the research. One key objective is to evaluate whether eventually consistent leader election can be effectively applied in cooperative scenarios such as perimeter patrol. A central challenge is understanding how temporary leadership inconsistencies impact mission performance. To investigate this, mission effectiveness is measured using quantitative metrics (KPIs), enabling an analysis of how inconsistency affects overall mission quality.

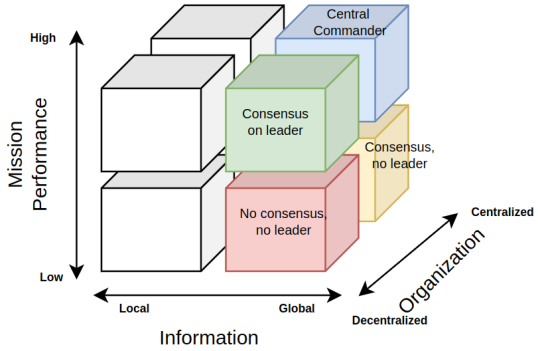

Fig. 6. Cooperative control mode classification.

Fig. 6 depicts all the UAV control modes implemented by the simulation. Each of the modes relies on different communication and fault-tolerant primitives. The mode that will be used in this research is consensus on a leader (or dynamic leader), because it shows the best balance between performance and fault tolerance [29], [30].

This research employs two types of failure detectors: the *perfect failure detector* (P) and the *eventual perfect failure detector* ($\Diamond P$), both defined in Chapter 1. It is important to note that the term "perfect" does not imply that crash detection is synchronized across all nodes. In practice, if the crash detection timeout is T, then the time difference between any two nodes detecting the failure of a process X will be less than T, and the worst-case detection time will be bounded by $2T$. This occurs because a node may respond with an acknowledgment at the beginning of a heartbeat interval, causing nearly $2T$ to pass before its absence is detected. While increasing the frequency of checks can improve detection time, it comes at the cost of higher network traffic, potentially impacting overall system performance.

This research considers exactly two leader election algorithms, each derived from one of the previously described failure detectors. For simplicity, we refer to the algorithm based on the perfect failure detector P as the *perfect* leader election algorithm, and the one based on the eventual perfect failure detector $\Diamond P$ as the *eventual* algorithm (Fig. 7). An important thing to understand about the eventual algorithm is how the underlying dynamic timeout works (Fig. 3 and Fig. 9).

The most important property of the *perfect* algorithm is **LE2** (as defined in Chapter 1), which ensures that once a leader is elected, it is guaranteed that all previously suspected leaders have indeed crashed. Although leader election does not occur synchronously due to the asynchronous nature of message delivery, the strong guarantees of P ensure that no two correct processes will simultaneously consider different leaders alive – effectively preventing *split-brain* scenarios in the perfect algorithm.

Implements:
LeaderElection, **instance** *le*.

Uses:
PerfectFailureDetector, **instance** $\mathcal{P}$.

upon event $\langle$ *le*, *Init* $\rangle$ **do**
suspected := $\emptyset$;
leader := $\perp$;

upon event $\langle$ $\mathcal{P}$, *Crash* | p $\rangle$ **do**
suspected := *suspected* $\cup$ $\{p\}$;

upon *leader* $\neq$ maxrank($\Pi \setminus$ *suspected*) **do**
leader := maxrank($\Pi \setminus$ *suspected*);
trigger $\langle$ *le*, Leader | *leader* $\rangle$;

Implements:
EventualLeaderDetector, **instance** Ω.

Uses:
EventuallyPerfectFailureDetector, **instance** $\diamond\mathcal{P}$.

upon event $\langle$ Ω, *Init* $\rangle$ **do**
suspected := $\emptyset$;
leader := $\perp$;

upon event $\langle$ $\diamond\mathcal{P}$, *Suspect* | p $\rangle$ **do**
suspected := *suspected* $\cup$ $\{p\}$;

upon event $\langle$ $\diamond\mathcal{P}$, *Restore* | p $\rangle$ **do**
suspected := *suspected* $\setminus$ $\{p\}$;

upon *leader* $\neq$ maxrank($\Pi \setminus$ *suspected*) **do**
leader := maxrank($\Pi \setminus$ *suspected*);
trigger $\langle$ Ω, *Trust* | *leader* $\rangle$;

Fig. 7. Perfect leader election algorithm (left) and eventual leader election algorithm (right).

Fig. 8 summarizes the strengths and limitations of each algorithm. The blue border highlights the focus of this research: evaluating how the *eventual* model impacts mission KPIs, and whether it is possible for it to outperform the *perfect* model under conditions where a longer timeout (adapted to observed delays) mitigates false crash detection.

Lower timeout	Larger timeout	Dynamic timeout
Fast response	Always one leader	Faster response
Multiple leaders	Decision delays	Eventually one leader
Double coverage		Temporary inconsistency

Fig. 8. Pros and cons of the different timeout strategies.

Section 3.2 describes benchmark experiments. The simulation environment includes a wide range of parameters, such as perimeter shape and size, UAV altitude (which determines surveillance diameter), UAV speed, threat spawn rate, UAV count, and network-related settings. However, due to the real-time nature of the simulation – where a 3-minute simulation requires 3 minutes of actual execution time – it is not feasible to explore all possible parameter combinations within a reasonable timeframe. As a result, this research focuses on varying only a selected subset of parameters: network configuration, timeout settings, and UAV crash rates. The remaining parameters are held constant across all experiments, with their values summarized in Table 2.

Table 2

Baseline Simulation Parameters

Parameter name	Unit	Value
Perimeter height	Meters	500
Perimeter width	Meters	100
UAV video diameter	Meters	30
UAV speed	Meters/sec	12
UAV count	UAV	10
One simulation time	Minutes	3
Threat rate	Threat/min	5
Threat's mean lifetime	Seconds	30
Simulation time	Seconds	300
Delta timeout	ms	3000

The surveillance (video) diameter is fixed at 30 meters, calculated using (2), where the field of view (FOV) is 80° and the UAV altitude h is 20 meters. A UAV speed of 12 m/s corresponds to the average cruising speed of the DJI Matrice 300 RTK, making it a realistic parameter for real-world deployments.

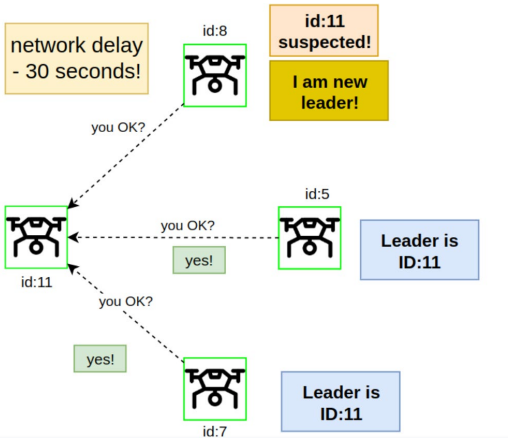

Fig. 9. Example of the eventual leader election algorithm execution. Due to partial synchrony, nodes are in temporary disagreement on who is the leader.

To estimate the required number of simulation runs necessary to obtain a reliable average for key performance indicators (KPIs), we use the following function:

$$n = \left(\frac{z \cdot \sigma}{E}\right)^2, \tag{1}$$

where

Z – Z-score, which represents confidence level (e.g. 1.96 for 95 %);

n – minimum number of simulations needed;

σ – standard deviation of the data set;
E – error margin.

For a 95 % confidence level, given the observed standard deviation and a specified margin of error of 1, it was determined that 56 simulation runs are required to ensure that the sample mean with σ = 3.82. Based on the 5-minute simulation duration, this allows for approximately two distinct parameter configurations (scenarios) to be evaluated within a 12-hour testing window.

$$h = \frac{D}{2 \times \tan\left(\frac{F}{2}\right)}, \tag{2}$$

where
F – field of view of the camera, deg;
h – UAV flight height, m;
D – meters of ground captured on video, m.

To simulate an unstable wireless network environment, a log-normal distribution was chosen to model message delays. Unlike a standard normal distribution, the log-normal distribution allows for occasional longer delays, better reflecting real-world wireless communication irregularities. The distribution is configured with parameters μ = 6 and σ = 1.5, introducing variability in delivery times. Additionally, a 1 % UDP packet drop rate is applied to further emulate unreliable network conditions.

In the naïve implementation, Node *A* would mark Node *B* as failed if an ACK is not received within a timeout threshold *X*, potentially leading to false failure detections due to packet loss or rare but valid delay outliers. To mitigate this, the retry mechanism is introduced: if no ACK is received within time *X*, Node *A* resends the heartbeat, up to n times. Each retry is modeled as an independent sample from the same log-normal delay distribution and is independently subject to packet loss.

The probability that a single retry attempt fails – either due to packet drop or delay exceeding *X* – is given by Equation (3). The cumulative distribution function (CDF) of the log-normal delay is Equation (4). Assuming independence, the probability that all n retries fail is Equation (5).

$$P_{\text{fail}}(X) = p_{\text{drop}} + (1 - p_{\text{drop}}) \cdot \mathbb{P}(D > X); \tag{3}$$

$$\mathbb{P}(D \le X) = \Phi\left(\frac{\ln(X) - \mu}{\sigma}\right) \quad \Rightarrow \quad \mathbb{P}(D > X) = 1 - \Phi\left(\frac{\ln(X) - \mu}{\sigma}\right); \tag{4}$$

$$P_{\text{total fail}}(X, n) = [p_{\text{drop}} + (1 - p_{\text{drop}}) \cdot \mathbb{P}(D > X)]^n, \tag{5}$$

where Φ is the CDF. The theoretical model shows that 4 retries with a delay threshold *D* = 3500 ms yield a 99.999 % probability that no false crash detection will occur. More than 200 simulation runs, each 5 minutes long, confirm that with a perfect timeout of 15 seconds and a retry interval of 3500 ms, no false crash detections were observed.

Experimental results showed that both the *lifetime monitor* KPI and the *threat registration time* (t. reg.) KPI exhibit a logarithmic dependency on the number of UAVs (Fig. 10). Given a 1200-meter perimeter, deploying 10 UAVs implies that each UAV is responsible for monitoring approximately 100 meters. Increasing the number of UAVs beyond 10 provides only marginal improvements in performance while significantly increasing network traffic.

First, an ideal network experiment was conducted with a packet drop rate of 0 and no delay. Subsequently, the degraded network model described in the previous section was applied. As shown in the table below, the unreliable network conditions can lead to a KPI reduction of up to 7.2 % (Table 3).

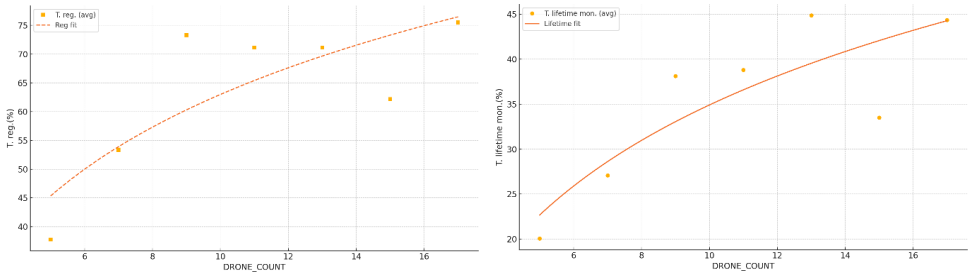

Fig. 10. Approximated function based on simulation data of KPI dependency on drone count (left – T. Reg, right – T. lifetime mon.).

Table 3

Comparison of Ideal Network and Baseline Parameters (Bad Network)

KPI	Ideal network	Bad network
T. reg. (%)	71	71
T. contr. (%)	52	46
T. lifetime mon. (%)	64	60
T. lifetime contr. (%)	47	40
Sum	**234**	**217**

In **Section 3.3**, the author analyzes the split-brain state – examining the conditions under which it may occur, its expected duration based on system parameters, and its impact on mission KPIs.

Not all false crash detection events lead directly to a split-brain state. For example, if a false crash is detected between two UAVs with low IDs – neither of which is the current leader – the system's functionality remains unaffected. Both UAVs will continue to receive commands from the current leader and send event updates to it. However, problems arise when a false crash is detected between a UAV with ID equal to the *current leader ID – 1* and the current leader itself.

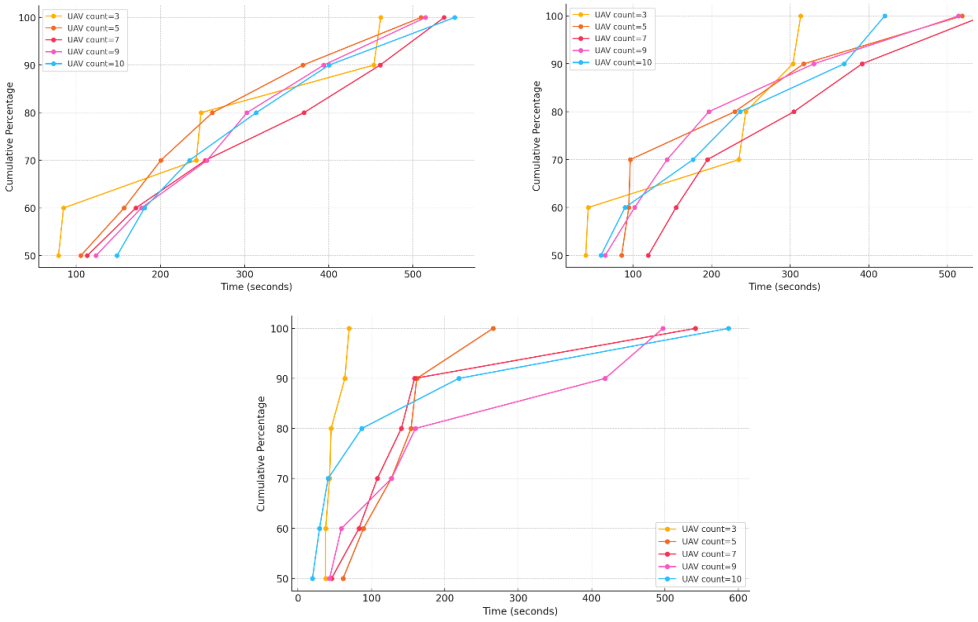

Fig. 11. Cumulative number of false crashes during simulation depending on delta timeout: 1 s (left), 2 s (right), 3 s (center).

In this set of experiments, the goal was to determine the point in time after which no further timeout adjustments occur, i.e., when all UAVs have stabilized, and no false crash detections persist. As expected, this strongly depends on the network model. The experiments were conducted using the previously described degraded network model. The results demonstrate that the *delta timeout* (DT) parameter significantly influences convergence time (Fig. 11). Experiments showed that even with a large DT of 3 seconds, and UAV group sizes greater than 5, false positives can persist for up to 10 minutes. Nevertheless, most false crash detections occur within the first 3 minutes.

In **Section 3.4**, the author compares eventual and perfect algorithms' KPIs. In the final experiment, the author evaluates how all three configurations perform under varying crash rates and degraded network conditions (Table 4). With a 30 % crash rate, the eventual algorithm exhibits only a 2 % KPI loss compared to the ideal timeout. It is worth noting that some simulations resulted in a timeout shorter than the ideal 15 seconds. This behavior is expected and may partly explain the minimal difference observed between the two algorithms. At a 50 % crash rate, the largest discrepancy occurs, with the eventual algorithm showing a 3.7 % loss relative to the ideal. At a 70 % crash rate, the difference becomes negligible, with less than 1 % deviation between eventual and ideal configurations.

The original goal of this research was to answer the question: *Can eventual leader election outperform perfect leader election in the context of expected UAV crashes?* This research demonstrated that eventually consistent distributed algorithms – specifically, eventual leader election – can be effectively applied in cooperative UAV missions such as perimeter patrol.

Although the core control logic of UAVs has a greater influence on mission success than occasional inconsistencies, the eventual approach exhibited strong resilience in crash-prone environments and, under certain conditions, even outperformed the perfect leader election strategy. Key findings revealed that mission performance is influenced by network instability, UAV count, and convergence time in non-linear ways. Additionally, incorporating logic to handle inconsistencies, such as the presence of multiple leaders, can further enhance system robustness. Overall, eventual algorithms offer a practical, fault-tolerant alternative to idealized coordination models, particularly in real-world scenarios where crashes and network disruptions are inevitable.

Chapter 4

Two types of AI are involved: onboard UAV AI and a more powerful ground-based central AI node. This research focuses on analyzing the threat detection aspect of the patrol mission. In this chapter, the UAV patrol mission is modeled as a Gordon-Newell queuing network to analyze the dynamic characteristics of the system, identify bottlenecks, model how AI performance impacts mission KPIs, and compare the effects of improving the central AI node versus the onboard AI.

In **Section 4.1,** the gap of simplified UAV vision in the simulation is emphasized. Threats in the simulation were modeled as 2D UTF-16 symbols, and the detection process was relatively simple – the 30 × 30-character array was passed to the UAV's threat detection module, and if a threat symbol was found, the threat was considered detected. While this approach poorly approximates real-world conditions, it was sufficient for the purposes of that study. In real-world scenarios, UAVs would be equipped with video cameras and would capture numerous objects during patrols. Most of these objects, whether static or dynamic, would not constitute real threats. To enable automatic threat detection, deep learning models can be used. The specifics of such models lie beyond the scope of this research. Instead, based on existing literature in the field [25], [31], assumptions regarding AI accuracy, hardware requirements, and performance will be adopted. The AI service will be treated as a black box.

Table 4

QN Model Parameters

Node	Name	$E[S]$ (s)	V_i	D_i
1	Interval time between UAV requests	20	1	20 (3 obj/min)
2	Cellular transmission	0.03	1	0.03
3	Central commander API	0.2	1	0.2
4	Central AI	1.5	1	1.5
5	WAN transmission	0.3	0.001	0.0003
6	Human operator analysis	30	0.001	0.03

This research aims to address the structure of the system to identify its main components and potential performance bottlenecks. Second, it seeks to determine the theoretical limits of system throughput and to analyze the response time distribution using queueing theory. Finally, the study explores which aspects of the system should be optimized – such as onboard AI accuracy or central AI response time – in order to achieve the best possible mission KPIs.

In **Section 4.2,** the mathematical model is defined. Each component in this system can be represented as a queue, and the entire system can be modeled as a queuing network. The network handles a single type of job: a request from a UAV to determine whether a detected object constitutes a real threat. In the current scenario, the number of UAVs is fixed, and they behave like users interacting with a shared service, issuing requests at average intervals. Therefore, the system is best represented as a closed queuing network, as illustrated in Fig. 12. Gordon–Newell queuing networks (GNQNs), named after their inventors, represent a class of closed Markovian queuing networks, where all service times follow a negative exponential distribution. In this model, the author works with M/M/1 queues that are interconnected based on predefined routing probabilities. The average service time at queue i is given by $\mathbb{E}[S_i] = 1/\mu_i$.

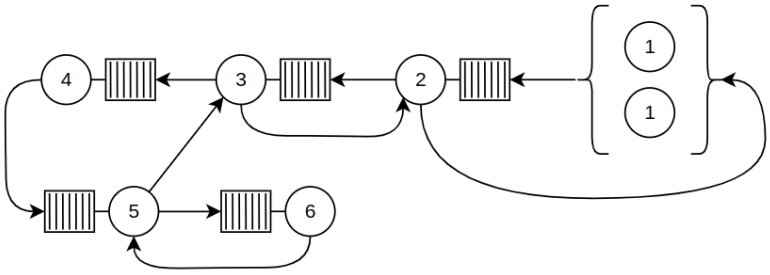

Fig. 12. Closed Gordon-Newell QN model for cooperative UAV network.

All QN parameters are summarized in Table 4.

In **Section 4.1,** the gap of simplified UAV vision in the simulation is emphasized. Threats in the simulation were modeled as 2D UTF-16 symbols, and the detection process was relatively simple – the 30 × 30-character array was passed to the UAV's threat detection module, and if a threat symbol was found, the threat was considered detected. While this approach poorly approximates real-world conditions, it was sufficient for the purposes of that study. In real-world scenarios, UAVs would be equipped with video cameras and would capture numerous objects during patrols. Most of these objects, whether static or dynamic, would not constitute real threats. To enable automatic threat detection, deep learning models can be used. The specifics of such models lie beyond the scope of this research. Instead, based on existing literature in the field [25], [31], assumptions regarding AI accuracy, hardware requirements, and performance will be adopted. The AI service will be treated as a black box.

The author uses Buzen's convolution algorithm to calculate various metrics of interest for this module. Buzen's convolution algorithm is an efficient method for solving the normalization

constant $G(K)$ in closed queueing networks with a finite number of customers K. The key idea is to recursively compute $G(K)$ using a dynamic programming approach, avoiding the direct summation over all possible states. Given service rates D_i for each queue i and the number of customers K, the normalization constant is computed as:

$$G(M,K) = \sum_{\mathbf{n}\in\mathcal{I}(M,K)} \prod_{i=1}^{M} D_i^{n_i} \tag{6}$$

where

- $G(M, K)$ – the normalization constant for a closed queuing network with M queues and K total customers;
- $\mathcal{I}(M,K)$ – the set of all possible state vectors $n = (n_1, n_2, \ldots, n_M)$;
- D_i – the service demand at queue i;
- n_i – the number of customers at queue i in state n;
- the product $\prod_{i=1}^{M} D_i^{n_i}$ evaluates the contribution of each state n to the normalization constant.

Buzen reformulated this summation as an iterative recurrence relation:

$$G(M,K) = G(M-1,K) + D_M\, G(M,K-1). \tag{7}$$

To start this recursion, we need boundary values. These can be derived as follows. When there is only 1 queue, by definition, $G(1,k) = D_1^k, \quad \text{for all } k \in \mathbb{N}$. Also, by the fact that there is only one way of distributing 0 customers over M queues, we have $G(m,0) = 1$, for all m [32].

A special case arises when analyzing a queueing network (QN) that includes a single infinite-server station. In such a network with K customers, the infinite-server station can be treated as a K-server station (Node 1 in our case). It is reasonable to assume that the network has only one infinite-server station; if multiple exist, they can be merged into a single equivalent station. The primary modification to the computational scheme involves adjusting the initialization to account for the presence of the infinite-server node. The steady-state distribution of the number of customers in this network, given a total of K customers, is provided in [32]:

$$\Pr\{N = n\} = \frac{1}{G(M,K)} \frac{D_1^{n_1}}{n_1!} \prod_{i=2}^{M} D_i^{n_i} \tag{8}$$

$$G(m,0) = 1, \quad \text{for } m = 1,\ldots,M, \tag{9}$$

$$G(1,k) = \frac{D_1^k}{k!}, \quad \text{for } k = 0,\ldots,K. \tag{10}$$

The only irregularity in the queueing network appears in the computational scheme at initialization, while the remaining computations stay unchanged (10)).

For the reconciliation of the results achieved by Buzen's algorithm, the mean value analysis approach can be used [32]. For infinite-server nodes, there is no waiting time, so the response time equals the service time:

$$E[R_j(K)] = E[S_j], \quad E[\hat{R}_j(K)] = D_j,$$
$$\mathbb{E}[\hat{R}_i(K)] = (\mathbb{E}[N_i(K-1)] + 1)\, D_i,$$
$$E[N_i(K)] = X(K) E[\hat{R}_i(K)],$$
$$\sum_{i=1}^{M} E[N_i(K)] = K, \quad X(K) = \frac{K}{E[\hat{R}(K)]}. \tag{11}$$

The expected response time, number of customers and throughput (Equation (11)). These equations extend MVA to IS nodes, keeping analytical consistency.

Table 5

Response Times and Throughput of QN Components by *K*

K	***E*[*R*$_1$(*K*)]**	***E*[*R*$_2$(*K*)]**	***E*[*R*$_3$(*K*)]**	***E*[*R*$_4$(*K*)]**	***E*[*R*$_5$(*K*)]**	***E*[*R*$_6$(*K*)]**	***X*(*K*)**
1	20	0.03	0.20	1.50	0.00	0.03	0.05
4	20	0.03	0.21	1.85	0.00	0.03	0.18
7	20	0.03	0.21	2.37	0.00	0.03	0.31
10	20	0.03	0.22	3.17	0.00	0.03	0.43
13	20	0.03	0.22	4.43	0.00	0.03	0.53
16	20	0.03	0.23	6.40	0.00	0.03	0.60
19	20	0.03	0.23	9.28	0.00	0.03	0.64
22	20	0.03	0.23	13.02	0.00	0.03	0.66
25	20	0.03	0.23	17.27	0.00	0.03	0.67
28	20	0.03	0.23	21.72	0.00	0.03	0.67

In **Section 4.3**, the author discusses key performance metrics, calculation of the response time distribution and how false AI classifications affect the mission KPI.

Table 5 presents possible performance values calculated using Buzen's algorithm, for different UAV count (*K*) and for an object rate of 3/min. The most interesting component is the system's bottleneck, which is the component with the largest service time – Node 4 (central AI). The maximum throughput of the system reached around 28 UAVs. The most important values are the system's response time, utilization and average number of nodes in queue. For reference, if *K* = 10 UAVs, the bottleneck is Node 4, with a 64 % utilization, 1.35 average queue and a response time of 3.17 s.

In Fig. 13, we can see how the fast response time grows depending on the UAV count and different mean request intervals.

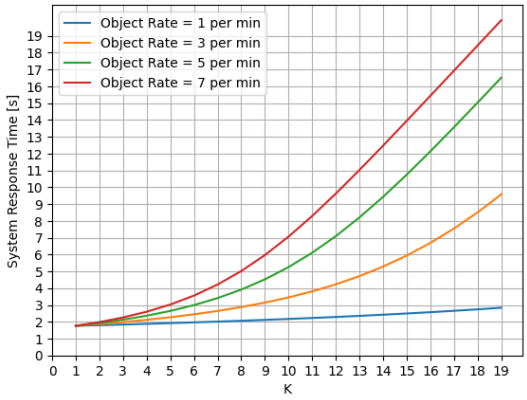

Fig. 13. System's response time depending on K (UAV count). Family of functions by the identified object rate.

The performance evaluation shows that as UAV count (K) increases, system response time grows non-linearly, especially beyond $K = 10$. The central AI (Node 4) is the system's bottleneck, reaching near-full utilization (up to 99.6 %) at high UAV counts. This causes sharp increases in response time and queue size.

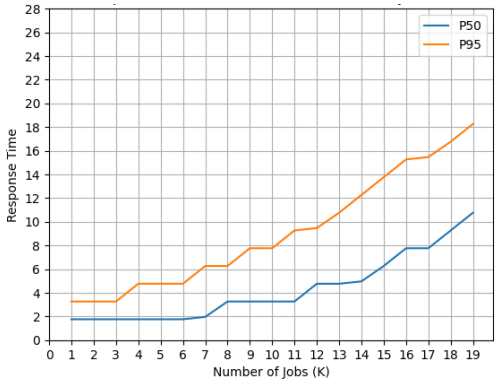

Fig. 14. Response time depending on K by percentiles 90 and 95.

Often, the average response time is not an adequate metric for non-functional system requirements, as it does not reflect the quality perceived by users. Instead, it is more informative to assess the time range within which 95 % of responses occur. With both the expected response time per state and the corresponding state probabilities available, it becomes possible to construct the probability distribution function (PDF) of the system's response time (Fig.14 and Fig. 15). This provides a more realistic view of the system's performance. Consequently, for lower values of K, the distribution plots may not appear representative. To improve interpretability, the author groups (bin) many low-probability response times into 1-second intervals, resulting in more informative and readable plots.

In this chapter, the goal is to further analyze possible scenarios of onboard AI behavior, model these scenarios, incorporate them into the broader queueing network (QN) framework, and ultimately evaluate how they impact the overall KPI.

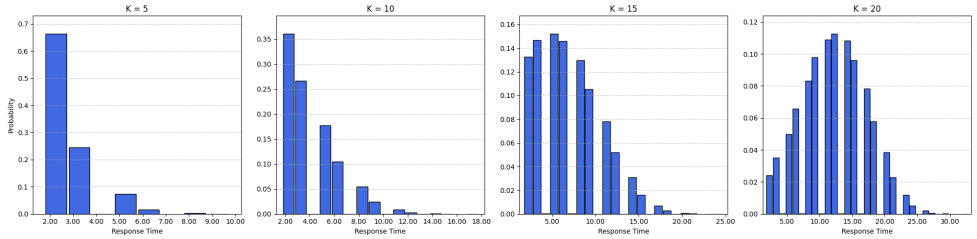

Fig. 15. Calculated response time probability distribution.

The UAV is equipped with a first layer of AI software capable of detecting unusual objects within its visual field. We assume this software operates both accurately and quickly. The next task is to classify each detected object into one of two categories: *threat* or *non-threat*. For example, a UAV may observe a bird during flight; this object should be correctly identified as non-threatening. If it is mistakenly classified as a drone, this results in a **false positive**, leading to wasted time tracking the bird instead of continuing patrol and identifying real threats.

Require: Response time PMF PMF, onboard accuracy A_0, threat ratio r, threat rate λ, simulation time T_{sim}
Ensure: Array of lost time fractions L
Set number of samples $S \leftarrow 5000$
$N \leftarrow \lfloor \lambda \cdot T_{\text{sim}} \rfloor$
$n_{\text{real}} \leftarrow \lfloor N \cdot r \rfloor$
$n_{\text{nonthreat}} \leftarrow \lfloor N \cdot (1 - r) \rfloor$
$n_{\text{fp}} \leftarrow \lfloor (1 - A_0) \cdot n_{\text{nonthreat}} \rfloor$
$n_{\text{fn}} \leftarrow \lfloor (1 - A_0) \cdot n_{\text{real}} \rfloor$
Extract (R, P) from PMF, where R is array of response times and P their probabilities
Normalize P: $P \leftarrow P / \sum P$
Sample $n_{\text{fp}} \times S$ values from R using P: FP_samples
$\text{wasted}_{\text{fp}} \leftarrow \sum$ FP_samples along each column
Sample $n_{\text{fn}} \times S$ values from R using P: FN_samples
$\text{wasted}_{\text{fn}} \leftarrow \sum$ FN_samples along each column
$\text{wasted}_{\text{total}} \leftarrow \text{wasted}_{\text{fp}} + \text{wasted}_{\text{fn}}$
$L \leftarrow \text{clip}(1 - \text{wasted}_{\text{total}} / T_{\text{sim}}, 0, 1)$
return L

Fig. 16. The algorithm calculates the percentage of wasted time out of the total simulation time. Wasted time is the time UAV spent tracking the wrong object that was incorrectly classified.

The possible outcomes when a UAV detects an object are:

- **Real threat**
  - Correctly tracks the threat → KPI gained (**true positive**)
  - Fails to track the threat → KPI lost (**false negative**)
- **Non-threat**

- Correctly ignores the object → no KPI impact (**true negative**)
- Incorrectly tracks the object → KPI lost (**false positive**)

Using this classification framework, we can now derive two formulas to quantify the rate of false negatives and false positives.

$$P(FalsePositive) = (1 - A_0) * (1 - r)$$
$$P(FalseNegative) = (1 - A_0) * r \quad , \tag{12}$$

where A_0 is the accuracy of onboard AI and r is the fraction of real threats among all objects.

Our goal is to estimate the fraction of the total simulation time lost due to incorrect onboard AI classifications. By progressively improving either the onboard AI's accuracy or the central AI's performance, we can generate plots showing the fraction of lost time as a function of each. These plots (Fig. 17) will help determine whether investment should be prioritized in improving onboard AI accuracy or in reducing the response time of the central AI. The plots below compare both strategies. As the object detection rate increases, the gap between the two lines becomes more pronounced.

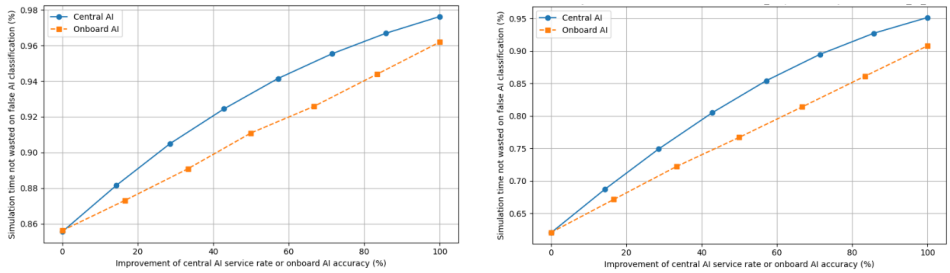

Fig. 17. Performance comparison of onboard AI accuracy and central AI processing time. Object rate is 3/min and utilization 78 % (left); object rate is 5/min and utilization 95 % (right).

This chapter introduces a performance modeling framework for cooperative UAV threat detection missions using a **closed Gordon-Newell queuing network (QN)**. The model captures the dynamic behavior of a hybrid AI architecture decision pipeline, where both onboard and central AI participate in object classification, and in rare cases, escalate to a human operator. To model the behavior of onboard AI classification, a linear model was developed to estimate the number of false detections. This model was then used to calculate the average expected time wasted by a UAV while waiting for a response from the service to correct its behavior. It was found that improving onboard AI accuracy leads to a linear reduction in wasted time. In contrast, improving the central AI service rate results in an even greater reduction, and this improvement follows a logarithmic pattern, even when central AI utilization is well below 100 %. The higher the server utilization, the greater the impact of performance improvements on reducing wasted time.

Chapter 5

This chapter explores the topic of consensus in vehicular networks. It begins with the Byzantine fault model, examining its requirements and surveying existing implementation approaches. The discussion then shifts to the more practical crash-stop consensus model, highlighting common use cases and performance challenges. Finally, a proposed method for modeling the performance of flooding-based consensus across various network topologies is presented, using NS-3 simulations supported by analytical approximations.

Section 5.1 provides a description of cooperative smart vehicle use cases where consensus can be utilized. Fig. 18 describes the cooperative crash avoidance at the intersection. To prevent a potential incident, we consider three theoretical approaches that could be used by intelligent vehicles: (1) each vehicle independently decides how to respond to the situation; (2) each vehicle makes a decision, followed by a voting process; the strategy with the most votes is selected and applied by the group; (3) a consensus algorithm is executed, allowing all vehicles in the group to agree on a single coordinated strategy.

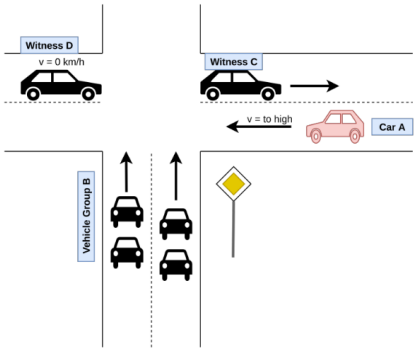

Fig. 18. Example use cases of emergency crash avoidance consensus in V2V.

The second scenario is illustrated in Fig. 19. Here, a platoon of vehicles is driving on a highway when a speeding vehicle is suddenly detected ahead. While there may be several strategies to avoid a collision, the underlying nature of the problem remains the same as in the first scenario – vehicles must make a coordinated decision based on partial observations. The key difference lies in the topology of the vehicles – how they are positioned and connected – which becomes the focus of interest in this research.

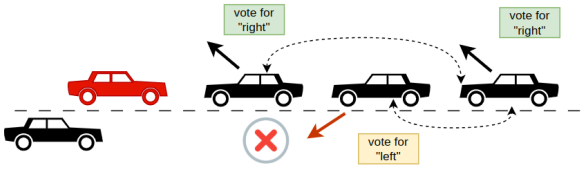

Fig. 19. Emergency crash avoidance consensus during platoon.

In **Section 5.2**, the author analyzes whether Byzantine crash failures are relevant in vehicular scenarios. There is a common misconception that Byzantine failures are purely theoretical and not worth addressing in real-world systems [33]. In recent years, there has been a growing emergence of decentralized architectures, including IoT networks and VANETs [34]. The limited attention given to BFT in V2V systems remains speculative. One possible reason is the widespread belief that encryption, digital signatures, and PKI alone are sufficient to secure a system [35]. For example, C-ITS standards focus extensively on PKI configurations but do not address fault tolerance at all [4].

In contrast, industries such as aviation not only acknowledge the importance of BFT but also actively research and integrate BFT solutions into their systems [36]. As noted in [33], one reason BFT is often overlooked in mainstream system design is the assumption that "if I haven't seen it, it doesn't exist."

The authors of [37] argue that the reliability of VANETs can be enhanced by applying BFT algorithms to decision-making processes within ad hoc networks. Additionally, they suggest that many challenges in VANETs can be mitigated using clustered topologies, which are recommended as a strategy for addressing congestion. In [38], the authors propose a protocol for VANETs that prioritizes user privacy and introduces incentive-based event message dissemination. They highlight two core challenges: "First, it is difficult to forward reliable announcements without revealing users' identities. Second, users usually lack the motivation to forward announcements." The proposed solution is a privacy-preserving announcement protocol that employs threshold signatures to protect user identities. In [39], the authors present a novel proof-of-eligibility mechanism for achieving consensus in VANETs. The core idea is to restrict participation in consensus to a subset of nodes directly involved in a specific event. This approach is intended to prevent nodes within short communication ranges from compromising the consensus process by increasing the likelihood of exceeding the one-third threshold of malicious participants, as outlined in the Byzantine fault model [40]. In [41], the authors address the problem of false data injection in vehicular network event reports. While numerous mechanisms have been proposed to enhance the security of VANETs, many remain susceptible to the injection of false information, which can be used to manipulate the behavior of other drivers. To mitigate this, the authors propose an algorithm based on the concept of *Proof of Relevance* (PoR), which aims to demonstrate that the reporting node is genuinely related to the event being reported.

Section 5.3 focuses on a few important questions related to using fail-stop consensus in V2V communication during emergencies. First, can a fail-stop consensus algorithm function effectively in urgent V2V situations where vehicles must quickly agree on an action? Second, how does the number of vehicles in the network impact the speed and efficiency of the algorithm? Third, how does the network topology influence the algorithm's performance?

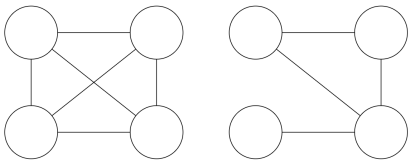

Fig. 20. Complete graph (left) and connected graph (right). The edge means there is a bounded, reliable connection, and a missing edge means either no connection at all or delayed delivery.

The flooding consensus algorithm [13] provides the following guarantees: (1) **termination** – every correct process eventually decides on some value; (2) **validity** – if a process decides on a value v, then v must have been proposed by some process; (3) **integrity** – no process decides more than once; (4) **agreement** – no two correct processes decide on different values. The algorithm operates under the assumption of a perfect failure detector. In each round, the algorithm incurs $O(N^2)$ message complexity. Additionally, after a process reaches a decision, it broadcasts $O(N^2)$ *DECIDED* messages. For each extra round triggered by a crash, another $O(N^2)$ messages are exchanged. Consequently, in the worst case, the total number of messages can grow to $O(N^3)$ [13].

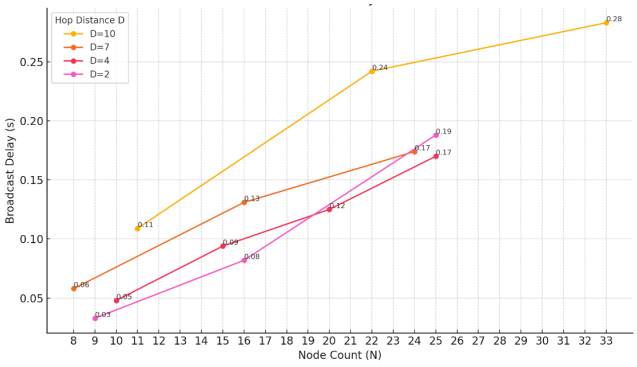

Fig. 21. Average all-to-all broadcast delay aggregated from the simulation data.

In this model, the author simulates consensus among V2V nodes by representing the network as a connected graph, where each node broadcasts messages that are flooded through the mesh. Only links with delays below a predefined real-time threshold are included; links exceeding this threshold are treated as non-existent, modeling the constraints of real-time communication (Fig. 20).

To evaluate broadcast propagation performance in wireless mesh networks, a custom simulation was developed using the NS-3 network simulator. The simulation models a grid-based mesh topology and implements a custom flooding algorithm to determine the time required for full message dissemination.

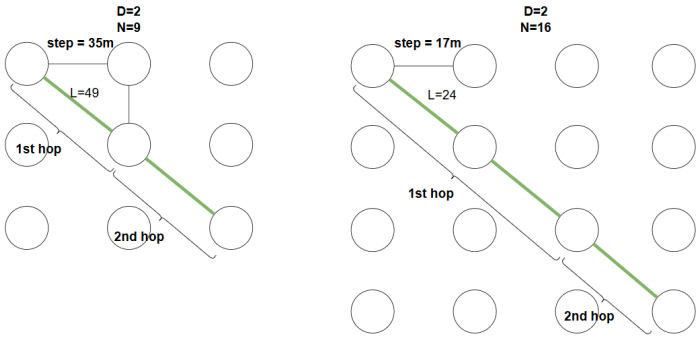

Fig. 22. Examples of how to configure different combinations of *N* and *D* using variable step. Communication range is constant (50 m).

Since UDP does not guarantee delivery, redundancy is introduced through randomized retransmissions. Each node schedules a configurable number of broadcasts, each separated by a random delay between 1 ms and a user-defined maximum. These delays help reduce interference. Fig. 22 presents example configurations that satisfy selected combinations of node count and graph diameter.

The simulation uses the UDP protocol without delivery acknowledgments. Given the strict timing requirements of emergency consensus – typically under 500 ms for full agreement – the focus of the simulation is to identify broadcast configurations that statistically guarantee delivery. In this model, when nodes execute consensus, each node broadcasts its message multiple times (e.g., three times), spaced by random delays. The expectation is that, with high probability, the message will be successfully delivered to all other nodes. If a node fails to receive a message within this period, it is treated as having crashed.

The final Table 6 presents the results of statistically reliable all-to-all average broadcast timings for various connectivity graph diameters and node counts. The corresponding raw data is visualized in Fig. 21. Due to limitations in topology configuration flexibility, only three to four distinct node counts could be tested for each diameter. The resulting analytical approximation of the all-to-all broadcast delay, based on the simulation data, is expressed by the 3rd degree polynom.

Table 6

All-to-All Broadcast Average Completion Time for Different Node Counts and Diameters

***D* = 10**		***D* = 7**		***D* = 4**		***D* = 2**	
N	***T* (ms)**	***N***	***T* (ms)**	***N***	***T* (ms)**	***N***	***T* (ms)**
11	109	**8**	58	**10**	48	**9**	33
N/A	N/A	**16**	131	**15**	94	**16**	82
22	242	N/A	N/A	**20**	125	N/A	N/A
33	283	**24**	174	**25**	170	**25**	188

With the analytical function modeling the delay for a single communication step (i.e., all-to-all broadcast involving $O(N^2)$ messages), we can now estimate the total consensus delay as a function of the system's crash rate. This enables the generation of delay estimates across a wide range of scenarios, from low-diameter topologies to high crash rate conditions (Fig. 23).

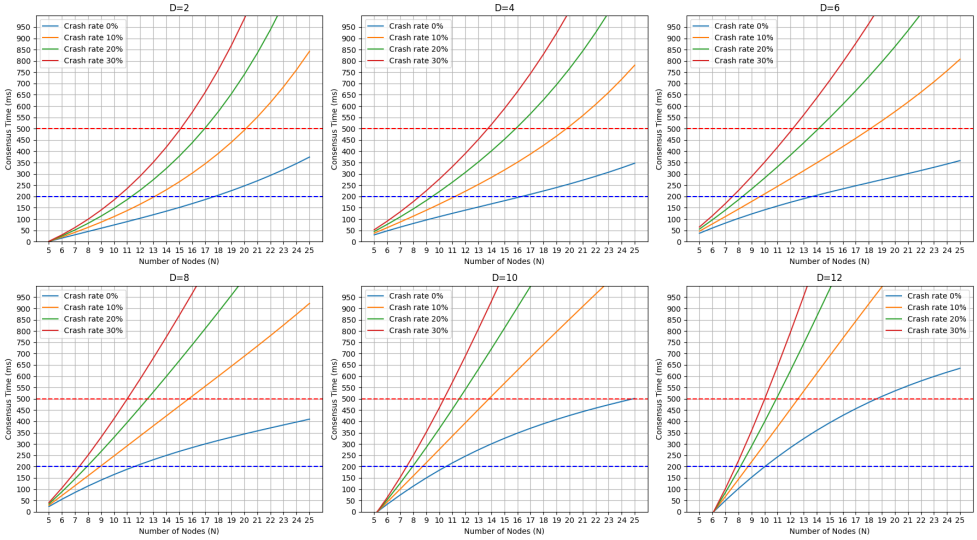

Fig. 23. Approximated consensus time depending on the number of node and graph diameter.

The findings from this research prove that it is possible to implement statistically reliable all-to-all broadcast in V2V mesh networks using NS-3 simulation, leveraging HWMP, UDP, randomized retries, and connected graph topologies. The simulations showed that, for up to 33 nodes with a connectivity graph diameter of 10, reliable broadcast can be achieved in under 300 ms. By approximating the all-to-all delay function, it becomes possible to extrapolate the expected delay for synchronous flooding-based consensus. Even in challenging conditions, for example, a platoon with a diameter of 10, nine nodes, and a crash rate of 30 %, consensus can still be reached in under 350 ms. The shape of the reliable broadcast delay curve depends on the diameter of the communication graph: with $D = 2$, the delay grows positively with node count, while for $D \geq 7$, the growth begins to decrease. A more linear trend is observed around $D = 4$. For example, with $N = 9$ and a crash rate of 20 %, the consensus delay is 120 ms for $D = 2$, and increases to 250 ms for $D = 7$, representing a 108 % increase.

MAIN RESULTS OF THE DOCTORAL THESIS

The following main conclusions were obtained during the development of the Doctoral Thesis.

1. When measuring the impact of fault-tolerant algorithms, it is advisable to first develop optimal decision logic, as this significantly influences mission KPI, even in scenarios with extensive faults.
2. Real-time simulation is invaluable for identifying bugs that would otherwise appear during real-world deployment, where fixing them can be significantly more costly. The primary drawback of real-time simulation is its execution time. A hybrid approach – using discrete simulation for overall performance testing and real-time simulation for advanced corner cases in later project phases – can help achieve maximum fault tolerance and resilience.
3. Simulation data indicate that the cumulative number of false crash detections caused by the eventual leader detector algorithm during simulation follows a logarithmic pattern. As the number of deployed UAVs increases, the time until the last false crash detection also increases significantly. With more than five UAVs and a larger detection timeout (DT), false crash detections can continue for up to 10 minutes. Nevertheless, due to the logarithmic nature of this behavior, most false crashes occur quickly, and only a small portion of total false negatives affects KPIs.
4. In cooperative UAV perimeter patrol mission simulations, quantitative KPI metrics are valuable for comparing leader election algorithm performance and for determining whether eventually consistent algorithms are suitable for cooperative UAV scenarios.
5. In the Perfect leader detector algorithm, extending the crash detection timeout from 15 seconds (ideal) to 60 seconds results in a 12.1 % KPI reduction during a five-minute simulation with a 30 % UAV crash rate. The eventually perfect leader detector, configured with a 5-second base crash detection timeout and a 3-second delta, shows a 4.1 % KPI loss compared to the perfect algorithm with the ideal timeout. Notably, the eventually perfect algorithm outperforms the perfect algorithm under realistic conditions (60 s timeout), achieving a 9.7 % KPI improvement at a 30 % crash rate.
6. A UAV surveillance network employing AI-based threat detection can be modeled using a closed Gordon-Newell queueing Network. While improvements in onboard AI accuracy reduce time lost to false positive monitoring linearly, reducing the central AI's response time has a greater impact on minimizing lost monitoring time under increasing system load.
7. The Byzantine error model is widely considered in avionics and space engineering. However, this term is never mentioned in ETSI ITS technical reports. A literature review of ITS reveals that Byzantine fault-tolerant algorithms proposed for V2X systems primarily focus on privacy, blockchain-based architectures, and reputation mechanisms. However, no research has yet proposed an emergency consensus protocol for V2V that explicitly considers Byzantine faults.

8. A custom randomized UDP-based reliable all-to-all broadcast algorithm using 802.11s mesh, tested with NS3 simulation, shows an average delay of 280 ms for 33 nodes with a connectivity graph diameter of 10. For smaller diameters and node counts, average delays are below 100 ms.
9. Mathematical modeling shows that the fail-stop flooding consensus algorithm, based on the aforementioned broadcast algorithm, can theoretically achieve consensus in under 500 ms for $D = 8$ *and* $N = 11$, with up to 30 % node crashes, and under 200 ms for smaller diameters. The shape of the reliable broadcast delay curve depends on the communication graph's diameter: for $D = 2$, the delay increases with the node count, while for $D \geq 7$, the growth rate decreases. A more linear trend is observed around $D = 4$.

The Doctoral Thesis summarizes the results of completed research and defines possible **future research directions.**

1. Testing the software developed for the perimeter patrol study on real drones using a cellular network and the NAT traversal system, also developed as part of the research.
2. Expanding the simulation's capabilities, such as enabling non-real-time simulation, implementing more complex coordination systems, and providing a more realistic network simulation.

BIBLIOGRAPHY

[1] L. Lamport, "Paxos Made Simple," *ACM SIGACT News (Distributed Computing Column) 32, 4 (Whole Number 121, December 2001),* p. 51–58, 2001.

[2] H. Ng, S. Haridi and P. Carbone, "Omni-Paxos X: Breaking the Barriers of Partial Connectivity," in *Proceedings of the Eighteenth European Conference on Computer Systems*, 2023.

[3] P. Kumari and P. Kaur, "A Survey of Fault Tolerance in Cloud Computing," *Journal of King Saud University - Computer and Information Sciences,* vol. 33, no. 10, p. 1159–1176, 2021.

[4] E. European Telecommunications Standards Institute, "Intelligent Transport Systems (ITS); Security; Threat, Vulnerability and Risk Analysis (TVRA)," ETSI, Sophia Antipolis Cedex, 2017.

[5] D. C. Gandolfo, L. R. Salinas, M. E. Serrano and J. M. Toibero, "Energy Evaluation of Low-Level Control in UAVs Powered by Lithium Polymer Battery," *ISA Transactions,* vol. 71, p. 563–572, 2017.

[6] M. Podhradský, J. Bone, A. Jensen and C. Coopmans, "Small Low-Cost Unmanned Aerial Vehicle Lithium-Polymer Battery Monitoring System," in *ASME 2013 International Design Engineering Technical Conferences and Computers and Information in Engineering Conference*, 2013.

[7] S. Jung, Y. Jo and Y.-J. Kim, "Flight Time Estimation for Continuous Surveillance Missions Using a Multirotor UAV," *Energies,* vol. 12, no. 5, p. 867, 2019.

[8] J. Hu, H. Niu, J. Carrasco, B. Lennox and F. Arvin, "Fault-Tolerant Cooperative Navigation of Networked UAV Swarms for Forest Fire Monitoring," *Aerospace Science and Technology,* vol. 123, p. 107494, 2022.

[9] D. Yu, H. Wu, Y. Sun, L. Zhang and M. Imran, "Adaptive protocol of raft in wireless network," *Ad Hoc Networks,* vol. 154, p. 103377, 2024.

[10] M. Saadoon, S. H. Ab Hamid, H. Sofian, H. H. Altarturi, Z. H. Azizul and N. N. Mohd Daud, "Fault Tolerance in Big Data Storage and Processing Systems: A Review on Challenges and Solutions," *Ain Shams Engineering Journal,* vol. 13, no. 2, p. 101538, 2021.

[11] Microsoft, "Transactional Outbox pattern with Azure Cosmos DB," 2021. [Online]. Available: https://learn.microsoft.com/en-us/azure/architecture/best-practices/transactional-outbox-cosmos.

[12] P. Deutsch, *Falacies of distributed computing.*

[13] C. Cachin, R. Guerraoui and L. Rodrigues, Introduction to Reliable and Secure Distributed Programming, Berlin: Springer, 2011.

[14] M. Herlihy and N. Shavit, The Art of Multiprocessor Programming, San Francisco: Morgan Kaufmann, 2008.

[15] N. A. Lynch, Distributed Algorithms, San Francisco: Morgan Kaufmann, 1996.

[16] L. Lamport, "The Part-Time Parliament," in *Concurrency: the Works of Leslie Lamport*, 2019, p. 277–317.

[17] N. C. Strole, "The IBM Token-Ring Network—A Functional Overview," *IEEE Network,* vol. 1, no. 1, p. 23–30, 1987.

[18] H. Garcia-Molina, "Elections in a distributed computing system," *IEEE Transactions on Computers,* vol. 31, no. 01, pp. 48-59, 1982.

[19] D. Ongaro and J. Ousterhout, "The Raft Consensus Algorithm," *Lecture Notes CS,* vol. 190, p. 2022, 2015.

[20] T. Feng, W. Fan, J. Tang and W. Zeng, "Consensus-based robust clustering and leader election algorithm for homogeneous UAV clusters," in *Journal of Physics: Conference Series*, 2019.

[21] Y. Zou, L. Yang, G. Jing, R. Zhang, Z. Xie, H. Li and D. Yu, "A survey of fault tolerant consensus in wireless networks," *High-Confidence Computing,* vol. 4, no. 2, p. 100202, 2024.

[22] S. Mousavi, F. Afghah, J. D. Ashdown and K. Turck, "Leader-follower based coalition formation in large-scale UAV networks, a quantum evolutionary approach," in *IEEE INFOCOM 2018-IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, 2018.

[23] R. Ganesan, X. M. Raajini, A. Nayyar, P. Sanjeevikumar, E. Hossain and A. H. Ertas, "BOLD: Bio-inspired optimized leader election for multiple drones," *Sensors (Basel, Switzerland),* vol. 20, no. 11, p. 3134, 2020.

[24] G. Wang, B.-S. Lee, J. Y. Ahn and G. Cho, "A UAV-Aided Cluster Head Election Framework and Applying Such to Security-Driven Cluster Head Election Schemes: A Survey," *Security and Communication Networks,* vol. 2018, no. 1, p. 6475927, 2018.

[25] H. C. Baykara, E. Bıyık, G. Gül, D. Onural, A. S. Öztürk and I. Yıldız, "Real-time detection, tracking and classification of multiple moving objects in UAV videos," in *2017 IEEE 29th International Conference on Tools with Artificial Intelligence (ICTAI)*, 2017.

[26] M. Mozaffari, W. Saad, M. Bennis, Y.-H. Nam and M. Debbah, "A Tutorial on UAVs for Wireless Networks: Applications, Challenges, and Open Problems," *IEEE Communications Surveys Tutorials,* vol. 21, no. 3, p. 2334–2360, 2019.

[27] I. Jawhar, N. Mohamed, J. Al-Jaroodi, D. P. Agrawal and S. Zhang, "Communication and Networking of UAV-based Systems: Classification and Associated Architectures," *Journal of Network and Computer Applications,* vol. 84, p. 93–108, 2017.

[28] G. Halkes and J. Pouwelse, "UDP NAT and Firewall Puncturing in the Wild," in *NETWORKING 2011*, Berlin, Heidelberg, Springer Berlin Heidelberg, 2011, p. 1–12.

[29] T. Shima and S. Rasmussen, UAV cooperative decision and control: challenges and practical approaches, SIAM, 2009.

[30] N. Hao, H. Yi, C. Tian, H. Yao and F. He, "A Distributed-Centralized Dynamic Task Allocation Algorithm for UAVs Tracking Moving Targets," in *2021 40th Chinese Control Conference (CCC)*, 2021.

[31] F. Cagatay Akyon, E. Akagunduz, S. Onur Altinuc and A. Temizel, "Sequence Models for Drone vs Bird Classification," *arXiv e-prints,* pp. arXiv-2207, 2022.

[32] B. R. Haverkort, Performance of Computer Communication Systems: A Model-Based Approach, Chichester: John Wiley and Sons, 1998.

[33] K. Driscoll, B. Hall, M. Paulitsch, P. Zumsteg and H. Sivencrona, "The Real Byzantine Generals," in *23rd Digital Avionics Systems Conference (DASC)*, 2004.

[34] R. G. Engoulou, M. Bellaïche, S. Pierre and A. Quintero, "VANET Security Surveys," *Computer Communications,* vol. 44, pp. 1-13, 2014.

[35] V. Lozupone, "Analyze Encryption and Public Key Infrastructure (PKI)," *International Journal of Information Management,* vol. 38, no. 1, pp. 42-44, 2018.

[36] Y. Yeh, "Safety Critical Avionics for the 777 Primary Flight Controls System," in *20th Digital Avionics Systems Conference (DASC)*, 2001.

[37] S.-C. Wang, Y.-J. Lin and K.-Q. Yan, "Reaching Byzantine Agreement Underlying VANET," *KSII Transactions on Internet and Information Systems,* vol. 13, no. 7, p. 3351–3368, 2019.

[38] L. Li, J. Liu, L. Cheng, S. Qiu, W. Wang, X. Zhang and Z. Zhang, "CreditCoin: A Privacy-Preserving Blockchain-Based Incentive Announcement Network for Communications of Smart Vehicles," *IEEE Transactions on Intelligent Transportation Systems,* vol. 19, no. 7, p. 2204–2220, 2018.

[39] H. Liu, C.-W. Lin, E. Kang, S. Shiraishi and D. M. Blough, "A Byzantine-Tolerant Distributed Consensus Algorithm for Connected Vehicles Using Proof-of-Eligibility," in *22nd International ACM Conference on Modeling, Analysis and Simulation of Wireless and Mobile Systems (MSWiM)*, 2019.

[40] L. Lamport, M. Pease and R. Shostak, "The Byzantine generals problem".

[41] Z. Cao, J. Kong, U. Lee, M. Gerla and Z. Chen, "Proof-of-Relevance: Filtering False Data via Authentic Consensus in Vehicle Ad-hoc Networks," in *IEEE INFOCOM Workshops*, 2008.

[42] L. Gupta, R. Jain and G. Vaszkun, "Survey of important issues in uav communication," *IEEE Communications Surveys Tutorials,* 2016.

Dmitrijs Rjazanovs was born in 1994 in Riga. He obtained a Master's degree (2018) in the study program "Intelligent Robotic Systems" from Riga Technical University. He has been working professionally in programming since 2015. Currently, is a software architect at C.T.Co Ltd. and a lecturer at Riga Technical University, teaching the courses "Cryptography" and "Real-Time E-Commerce".
His scientific interests include distributed systems, consensus algorithms, and software engineering.