A model of an intelligent clustering system with an external module for the architecture of RTOS with intensive changes of states regarding their flexibility and balancing*

Oleksandr Kozelskiy[1,*,†], Antonina Kashtalian[1,†], Vasyl Stetsyuk[1,†], Dmytro Martiniuk[1,†], Anatoliy Sachenko[2,3,†]

[1] *Khmelnitsky National University, Khmelnitsky, Instytutska street 11, 29016, Ukraine*
[2] *Kazimierz Pulaski University of Technology and Humanities, Department of Informatics, Radom, Poland*
[3] *Research Institute for Intelligent Computer Systems, West Ukrainian National University, Ternopil, Ukraine*

Abstract

The article considers the creation of a model of a clustering system with an external module for the architecture of OSRCH with intensive changes of states, which concerns their flexibility and load. An overview of modern approaches to clustering and load balancing in real-time operating systems was performed, and their advantages and disadvantages were identified. The study is aimed at identifying the main problems that need to be solved in order to increase the efficiency of clustering and improve load balancing in the real conditions of the operation of real-time operating systems. The main challenges arising in the context of ensuring high performance of such systems are considered. The obtained results can be used to develop new methods that increase the efficiency of resource management and ensure the stability of real-time operating systems.

Keywords
operating system, RTOS, flexibility enhancement, task scheduler, resource management

1. Introduction

Real-time operating systems (RTOS) are essential in fields where precision and timely response are critical, such as avionics, automotive, healthcare, and telecommunications. They ensure system determinism and predictability, guaranteeing the completion of tasks within strictly defined time frames. However, modern RTOS face several challenges. Achieving full determinism is complicated by potential hardware and software failures, as well as the complexity of synchronizing parallel processes. Efficient resource management is not always accomplished, which can lead to system overload and delays in task execution. This is particularly critical in high-load environments, where the volume of data and the number of tasks exceed the system's capabilities.

The flexibility and scalability of RTOS also remain challenges. Developing architectures that can easily adapt to new tasks and conditions without significant resource expenditure is necessary to maintain the relevance and effectiveness of systems in a rapidly changing technological environment [28].

AdvAIT-2024: 1st International Workshop on Advanced Applied Information Technologies, December 5, 2024, Khmelnytskyi, Ukraine - Zilina, Slovakia
* Corresponding author.
† These authors contributed equally.
oleksandr.kozelskiy@khmnu.edu.ua (O. Kozelskiy); yantonina@ukr.net (A. Kashtalian); swmua@khmnu.edu.ua (V. Stetsyuk); martiniyuk.dim14@gmail.com (D. Martiniuk); as@wunu.edu.ua (A. Sachenko)
0009-0002-7157-6499 (O. Kozelskiy); 0000-0002-4925-9713 (A. Kashtalian); 0000-0001-9880-2666 (V. Stetsyuk); 0009-0002-3524-872 (D. Martiniuk); 0000-0002-0907-3682 (A. Sachenko)

The purpose of this study is to analyze the existing methods of clustering and load balancing in RTOS, to identify their advantages and disadvantages, as well as to pinpoint the main issues that need to be addressed. This will contribute to the development of new approaches and algorithms that will enhance the efficiency, reliability, and flexibility of real-time operating systems in mission-critical applications.

2. Analysis of existing clustering methods

Clustering in real-time operating systems (RTOS) is a method of grouping data or objects based on similar characteristics or properties. Various methods are used in this field, each with its own advantages and disadvantages. The main categories include time-based methods, machine learning methods, genetic algorithms, and graph-based approaches. Each of these approaches offers unique solutions for clustering tasks and has specific areas of application.

Time Division Multiplexing (TDM) is used for efficient resource management in RTOS by allocating specific time intervals for different tasks [2,3,4]. This ensures predictable and regular access to resources, which is critically important for maintaining the properties of real-time systems [1].

In TDM, each task or group of tasks receives exclusive access to processor resources during a pre-defined time slot in a cyclic order. This approach supports high system performance and reliability by guaranteeing the regular execution of tasks [4].

It is a positive feature that each task is executed at a defined time, ensuring regular system operation. For example, in manufacturing processes, TDM can allocate fixed time slots to different sensors and mechanisms, preventing conflicts and ensuring seamless operation. The simplicity of implementing this method makes it easy to deploy and configure, which is crucial in embedded systems. Moreover, TDM promotes efficient resource usage through controlled time-slot allocation, optimizing their use. However, if a task does not use the entire allocated time slot, the unused resource is wasted, leading to inefficiency.

In systems with variable loads, fixed time intervals may be inefficient because they do not adapt to current needs [5,6]. Scalability also becomes an issue: in large systems with many tasks, it is difficult to provide sufficient time slots for each task. Optimizing resource usage is critical, and there arises a need for methods to dynamically redistribute unused resources among tasks. Considering task priorities is also important to ensure timely execution of critical tasks, especially in systems where this can affect safety or efficiency [7].

Machine Learning (ML) methods are increasingly applied for clustering and load balancing in RTOS due to their ability to automatically find optimal solutions and adapt to changing conditions [9,10]. They include clustering algorithms, reinforcement learning, and neural networks, which provide efficient resource allocation and task management in real-time. ML algorithms can automatically detect optimal patterns and solutions based on large datasets, aiding effective load distribution. They can dynamically change resource allocation strategies in response to system demands, ensuring optimal performance even under varying conditions. ML methods also help optimize resource use by predicting system load and dynamically adjusting resource distribution [11].

However, ML methods require large volumes of high-quality data for effective training, which can be challenging in many RTOS. The computational complexity of ML algorithms, especially neural networks, demands significant resources, which can be an issue for systems with limited capabilities. Implementing and tuning ML methods can be complex and require a high level of expertise [13, 14]. The complexity of algorithms may hinder understanding and trust in their results, affecting their integration and monitoring in real systems.

Genetic Algorithms (GA), based on the principles of natural selection, are widely used to solve optimization problems, such as clustering and load balancing in RTOS [15]. They employ processes of selection, crossover, and mutation to find optimal or near-optimal solutions to complex problems. GAs can efficiently handle various types of tasks and resource constraints, finding global optima and

adapting to different problems without significant changes to the algorithm [17]. They are also resilient to changes and uncertainties in the system, allowing stability and performance to be maintained under dynamic conditions. However, genetic algorithms have high computational complexity and require careful tuning of parameters, such as population size and crossover and mutation rates. Slow convergence can be an issue in real-time systems where timely decisions are critical. Scalability is also a concern because, as system size increases, computational requirements grow. Defining effective termination criteria and handling dynamic changes in the system remain challenges.

Graph-based methods use graph theory to model interactions between tasks and resources in RTOS. Tasks and resources are represented as graph nodes, and the connections between them as edges. This allows for effective task distribution and management using spectral clustering algorithms, minimum cut/maximum flow, and graph partitioning. These methods can model complex interactions and dependencies between tasks and resources, which contributes to more precise clustering and load balancing. They scale well and can work with large systems [19], which is essential for RTOS with numerous tasks. Flexibility in optimization allows these methods to be adapted to various goals, such as minimizing communication costs or evenly distributing the load. However, it is important to consider that these methods have high computational complexity, as building and processing large graphs require significant resources. This can be a problem for RTOS with limited capabilities and strict timing constraints.

The complexity of handling dynamic changes is also a challenge because these methods often assume a static graph structure. The need for accurate information on dependencies between tasks and resources can complicate their application in complex systems where such dependencies are not always clearly defined. Scalability remains an issue, as computational demands increase with system size and complexity. Developing more efficient algorithms and using parallel processing methods may help address these issues.

Thus, the analysis of existing clustering methods in RTOS shows that each has its advantages and disadvantages. The choice of the optimal method depends on the specific requirements of the system, its size, complexity, and resource constraints.

Further research is aimed at overcoming existing drawbacks and developing more efficient, adaptive, and scalable clustering methods for real-time operating systems [16,18]. This may include implementing hybrid approaches [20,21] that combine the advantages of different methods [22], and developing new algorithms that can work effectively in dynamic and complex RTOS environments, as well as monitoring security against viruses, such as with BOTNET [25, 26, 27].

3. Formulation of the problem

To create a new architecture for real-time operating systems (RTOS), it is essential to conduct a thorough analysis of the limitations and shortcomings of existing clustering methods. Understanding why current approaches do not provide the desired efficiency, flexibility, and scalability is crucial. Based on this analysis, innovative solutions should be developed that combine the strengths of various methods and include mechanisms to counter malware [24].

The focus should be on developing new algorithms capable of effectively operating under dynamic real-time conditions. This may involve integrating the adaptability of machine learning with the efficiency of genetic algorithms and the ability of graph-based methods to model complex interactions. Such an approach would help overcome the limitations of each individual method, resulting in a more universal and effective solution.

It is important to optimize the computational complexity of new algorithms to ensure their operation under resource-constrained environments with strict timing requirements [23]. This can be achieved by utilizing parallel processing methods or developing lighter algorithms that do not require extensive computations.

Adaptation to dynamic changes within the system is critical. New approaches should include mechanisms that allow for rapid response to changes in load and resource availability, maintaining

stable performance and reliability of the system in real-time. Additionally, addressing the issue of data availability and quality for training algorithms is essential.

In conclusion, creating a new architecture will require a comprehensive approach that combines innovative technologies, resource optimization, and a deep understanding of the specifics of real-time operating systems. This will enable the development of solutions that enhance the efficiency, flexibility, and reliability of such systems, meeting modern demands and challenges.

4. Main part

4.1. Architecture model of a task scheduler with intelligent clustering on an external module

The architecture model of a task scheduler with intelligent clustering is based on the principle of collecting datasets that are sent to a separate system. This system analyzes the data and uses it to improve resource management efficiency in the future.

The first step in developing a new approach is to collect data that includes information about tasks, resources, time parameters, and other important factors. This may involve metrics like task execution time, resource consumption, priorities, interdependencies between tasks, and other critical characteristics. Each task is linked to the corresponding parameters that reflect its requirements and behavior in the system.

Let $X = \{x_1, x_2, \dots, x_n\}$ — be a set of tasks, and $R = \{r_1, r_2, \dots, r_m\}$ — be a set of resources. Each task $x_i \in X$ is described by a set of characteristics $F_i = \{f_1, f_2, \dots, f_k\}$, where f_j represents one of the parameters, such as execution time, resource consumption, priorities, and interdependencies.

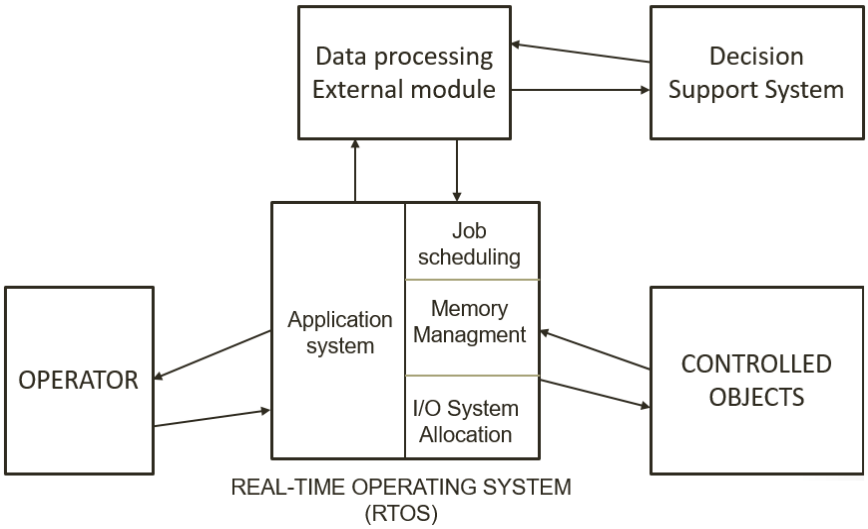

Figure 1: The architecture model of an intelligent clustering system with an external module.

Thus, data is collected to determine the set of features F_i for each task x_i, meaning each task is linked to a set of characteristics.:

$$F_i = \{f_1(x_i), f_2(x_i), \dots, f_k(x_i)\}, \forall x_i \in X, \quad (1)$$

where $f_j(x_i)$, — is the value of the *j*-th characteristic for problem x_i.

For the correct functioning of machine learning algorithms, feature vectors F_i are normalized. Let $\tilde{F}_i$ — be the normalized feature vector:

$$\tilde{F}_i = \left(\frac{f_1(x_1) - minf_1}{max\ f_1 - minf_1}, \dots, \frac{f_k(x_i) - minf_k}{max\ f_k - minf_k}\right) \quad (2)$$

This brings all features to a common scale for further analysis.

From the set of all features F_i , the most significant features $S \subset F$, which have the greatest impact on system performance, are selected. Let $S = \{s_1, s_2, \dots , s_p\}$, where s_j — is a selected feature from the set F_i, with $p < k$.

Tasks are clustered based on their features. Algorithms such as *k*-means or hierarchical clustering are used for this [8]. Clustering defines the function of task x_i belonging to cluster C_j:

$$\forall x_i \in X, x_i \to C_j, \text{where } j = arg\ \min_j d\ (\tilde{F}_i, \mu_j), \quad (3)$$

where $d(\cdot,\cdot)$ — is the distance between feature vectors, and μ_j — is the center of cluster C_j.

After clustering, a machine learning model M is built and trained on clusters $C_1, C_2, \dots , C_t$. The model uses the feature values S to classify new tasks x_{new}:

$$M(x_{new}) = C_j, \quad (4)$$

where x_{new} — is a new task, and C_j — is the cluster it belongs to.

The machine learning model predicts future resource needs based on the current system state. The prediction function $P(t)$determines resource requirements at time t:

$$P(t) = \{r_1(t), r_2(t) , \dots , r_m\ (t)\}, \quad (5)$$

where $r_j(t)$ — is the predicted load on resource r_j at time t.

For optimal task distribution, a minimization function is used to reduce deviations between actual load $r_j^{fact}(t)$ and predicted $r_j^{opt}(t)$:

$$min \sum_{j=1}^{m} \left(r_j^{fact}(t) - r_j^{opt}(t)\right)^2, \quad (6)$$

This enables balanced load distribution and resource management, prioritizing the execution of the most critical tasks and improving the system's overall performance.

The model predicts future load and dynamically allocates tasks between processors or cores to achieve optimal balance. This helps prevent bottlenecks and ensures smooth system operation without failures.

In summary:

- The system collects data on tasks and resources in parallel with their execution in real time, without affecting its performance.
- Data processing is offloaded to a separate module or service.
- Information analysis and machine learning model updates are performed periodically to keep the data current.

The results are used for decision-making regarding resource allocation and task scheduling in real-time mode.

5. Experimental research

5.1. Prototype preparation

To implement the prototype, it is proposed to use the FreeRTOS operating system with modified tasks.c and main.c files. In tasks.c, which is the main file for the FreeRTOS scheduler and contains task management functions, the vTaskSwitchContext() function is modified to call a custom method

for task distribution. In main.c, code is added to create data collection tasks and call a custom scheduler, which will be implemented in a new file custom_scheduler.c. This file will define functions for data collection, task classification, and optimization of their distribution.

The algorithm consists of the following steps:

1. Data Collection in FreeRTOS: Data is collected by recording parameters during system operation or through simulation. Key parameters to be used in the model are defined, such as task type, resource usage (CPU), task execution time, and task priority. Based on this, a software model is created. Task execution in the system is simulated, with parameters being recorded during execution. This can be implemented using loops and timers to simulate real-time operation. The collected data is analyzed and transformed into the required format for further use in training machine learning models. Data is sent from the microcontroller via a configured HTTP client to an online server built on C# .NET 8, which receives this data.
2. Data Preparation for Modeling: Based on the collected data, a dataset is formed containing information about tasks, their characteristics, and time parameters. An external mathematical library, Math.NET Numerics, is used to process and prepare data for sending to the TinyML service.
3. Sending Data for Training: Using the .NET service, the data is converted into a format compatible with TinyML, sent for training, and a trained model is obtained.
4. Model Analysis and Prediction: After successful model training on the server, the next step is to use it for predictions. In this case, the model will be applied for prediction to optimize task distribution.
5. Sending Results to the Microcontroller: After performing predictions, the results are sent back to the microcontroller to update decision-making processes in the custom_scheduler.c file.

The data collection process will occur at intervals defined in constants and will automatically repeat.

5.2. Conducting experiments and comparing efficiency

FreeRTOS uses a time management principle through its task scheduler, which is very similar to Time Division Multiplexing (TDM). We can compare the results of default FreeRTOS and the results of FreeRTOS with the modified task scheduler.

To compare the efficiency of standard FreeRTOS and FreeRTOS with the modified architecture, several key metrics will be used:

- Task: This could be a data processing task, input/output, computational task, or simply a background task.
- Resource Usage (CPU): Measurement of CPU usage during task execution.
- Task Execution Time: Measurement of the time required to complete a task.

First, we will run the program using the standard task scheduler and collect data on task execution time, resource usage, throughput, and latency through logging. After this, we will analyze the performance of the task scheduler in FreeRTOS with the modified architecture. For statistical purposes, we will set the number of tasks to 4. We will connect the DSView program, which typically works with the DS Logic logic analyzer.

Based on the results of the standard FreeRTOS task scheduler and the modified one (Figure 2, figure 3), several conclusions can be drawn.

Standard Scheduler:

- Tasks 1, 2, and 4 have equal priority, as seen from their even distribution on the graph. They run in parallel or take turns, but without clear prioritization.
- Task 3 has high priority and is executed first after every interrupt generated by the system timer "tick".
- The idle process does not run often because the processor is continuously occupied with task execution.

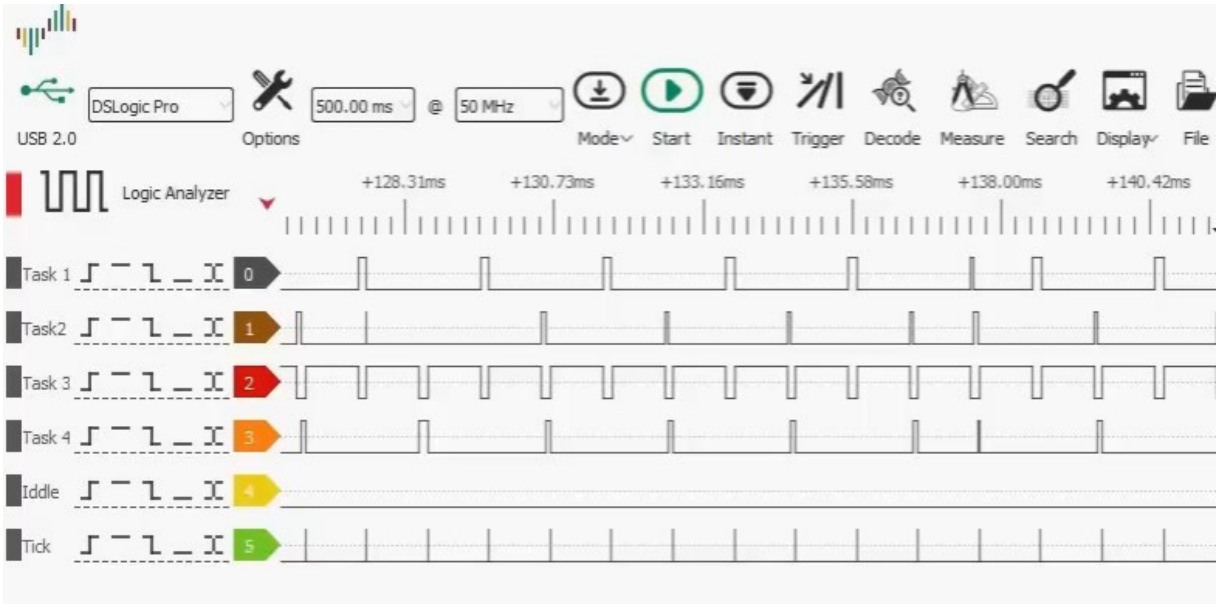

Figure 2: Graph of CPU time distribution between tasks in the standard FreeRTOS scheduler and the modified scheduler**.**

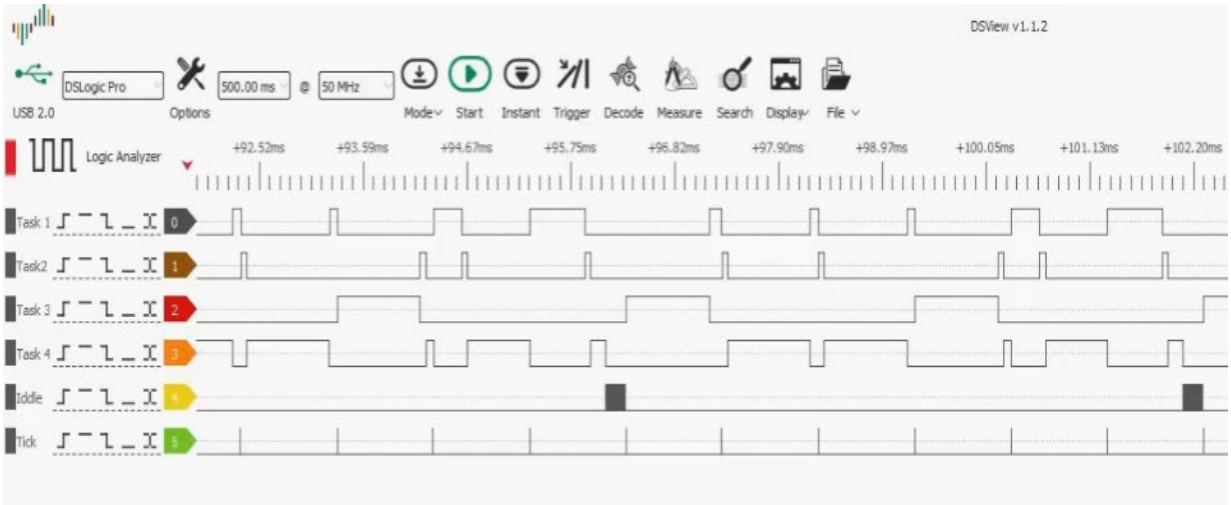

Figure 3: Graph of CPU time distribution between tasks in the standard FreeRTOS scheduler and the modified scheduler.

Modified Scheduler:

- Priority-based task allocation: This graph shows that the modified scheduler implements task distribution based on their priority and execution time. There is a clear division of time between tasks. More controlled and structured use of CPU time is observed.

- Preemptive scheduling: Tasks are interrupted by other higher-priority tasks, demonstrating preemptive scheduler behavior where lower-priority tasks are paused to allow more important tasks to execute.
- Idle process: The idle process now runs more actively when other tasks finish their execution time or are in a waiting state. This indicates that the system operates more efficiently as CPU power is used more evenly.

In the standard scheduler, resources are not fully optimized because tasks with equal priorities can interrupt each other, and the high-priority task completely occupies the CPU time.

In the modified scheduler, priority-based preemptive scheduling is clearly implemented, allowing the system to dynamically allocate task execution time and use CPU power more efficiently, enabling the system to respond faster to changes.

6. Conclusions

This work conducted a detailed analysis of existing clustering methods in real-time operating systems (RTOS), including time-based methods, machine learning methods, genetic algorithms, and graph-based approaches. It was found that each of these methods has its advantages and disadvantages, which limit their effectiveness, flexibility, and scalability in modern dynamic systems.

To overcome the identified limitations, intelligent clustering based on the use of machine learning methods on external modules was considered. This approach involves collecting and analyzing data on system tasks and resources, classifying tasks based on their characteristics, and optimizing resource allocation based on machine learning model predictions.

Experimental results show improved performance, including reduced CPU usage and decreased task execution time compared to the classic FreeRTOS scheduler.

Thus, intelligent clustering demonstrates the potential of using machine learning methods, and shifting the analysis logic to external modules for resource management and load balancing in real-time operating systems does not burden already limited resources.

Further research may focus on improving machine learning models, optimizing the computational complexity of algorithms, and adapting the system to dynamic changes and scaling for operation in more complex and demanding environments. Efforts to increase the efficiency of CPU time distribution are ongoing.

Declaration on Generative AI

During the preparation of this work, the authors used Grammarly in order to: grammar and spelling check; DeepL Translate in order to: some phrases translation into English. After using these tools/services, the authors reviewed and edited the content as needed and take full responsibility for the publication's content.

References

[1] L. Zhang et al., "Layered-Division-Multiplexing: Theory and Practice," in IEEE Transactions on Broadcasting, vol. 62, no. 1, pp. 216-232, March 2016.

[2] F. Zhang, G. Sun, Y. Zhou, B. Gao and S. Pan, "Towards High-Resolution Imaging With Photonics-Based Time Division Multiplexing MIMO Radar," in IEEE Journal of Selected Topics in Quantum Electronics, vol. 28, no. 5: Lidars and Photonic Radars, pp. 1-10, Sept.-Oct. 2022, Art no. 6000310, doi: 10.1109/JSTQE.2022.3146862.

[3] A. Gelgor et al., "Flexible Satellite Direct-to-Home Services With Layered-Division Multiplexing," in IEEE Transactions on Broadcasting, vol. 67, no. 1, pp. 83-95, March 2021, doi: 10.1109/TBC.2020.3031733.

[4] Hung, T.-Y.; Chen, G.-H.; Lin, Y.-Z.; Chow, C.-W.; Jian, Y.-H.; Kuo, P.-C.; Peng, C.-W.; Tsai, J.-F.; Liu, Y.; Yeh, C.-H. Wideband and Channel Switchable Mode Division Multiplexing (MDM) Optical Power Divider Supporting 7.682 Tbit/s for On-Chip Optical Interconnects. Sensors 2023, 23, 711. https://doi.org/10.3390/s23020711.

[5] G. Schwäricke, T. Kloda, G. Gracioli, M. Bertogna, M. Caccamo. Fixed-Priority Memory-Centric Scheduler for COTS-Based Multiprocessors. In 32nd Euromicro Conference on Real-Time Systems (ECRTS 2020). Leibniz International Proceedings in Informatics (LIPIcs), Volume 165, pp. 1:1-1:24, Schloss Leibniz-Zentrum für Informatik (2020).

[6] U. V. Rane, C. Panem, G. Abhyankar and R. S. Gad, "Network on Chip(NoC) Mesh Topology FPGA Verification: Real Time Operating System Emulation Framework," 2024 Fourth International Conference on Advances in Electrical, Computing, Communication and Sustainable Technologies (ICAECT), Bhilai, India, 2024, pp. 1-5, doi: 10.1109/ICAECT60202.2024.10468944.

[7] Peiji Song, Yuan Liu, Zhouyi Hu, Chun-kit Chan, and Di Che, "Extending the coverage of user-diversified IM-DD PON by FDM: a mathematical model with experimental verification," Opt. Lett. 49, 2457-2460 (2024).

[8] Debi Pada Jana, Abhishek M. Shukla, Sumanta Gupta, "K-Means algorithm-based detection for wavelength division multiplexed OOK PD-NOMA system over turbulent optical channel," Opt. Eng. 61(3) 036111 (29 March 2022) https://doi.org/10.1117/1.OE.61.3.036111

[9] Sharma, S., Chmaj, G., Selvaraj, H. (2023). Applying Machine Learning to Minimize the Impact of Sensor Failures to RTOS Based Internet of Things Systems. In: Selvaraj, H., Chmaj, G., Zydek, D. (eds) Advances in Systems Engineering. ICSEng 2023. Lecture Notes in Networks and Systems, vol 761. Springer, Cham. https://doi.org/10.1007/978-3-031-40579-2_14 Cho, C.; Seong, Y.; Won, Y. Mandatory Access Control Method for Windows Embedded OS Security \ Electronics, 10, 2021; p12.

[10] A.Mohammad, R. Das, M. A. Islam, F. Mahjabeen, (2024). Real-time Operating Systems (RTOS) for Embedded Systems. Asian Journal of Mechatronics and Electrical Engineering, 2(2), 95–104. https://doi.org/10.55927/ajmee.v2i2.7761

[11] Sharma S. Check for Applying Machine Learning to Minimize the Impact of Sensor Failures to RTOS Based Internet of Things Systems Saugat Sharma, Grzegorz Chmaj (), and Henry Selvaraj //Advances in Systems Engineering: Proceedings of the 30th International Conference on Systems Engineering, ICSEng 2023, Las Vegas, Nevada, USA August 22-24, 2023. – Springer Nature, 2023. – T. 761. – C. 135.

[12] J. Pradhani et al., "Prototype Development for Water Leakage Monitoring System with RTOS Implementation," 2024 International Conference on Distributed Computing and Optimization Techniques (ICDCOT), Bengaluru, India, 2024, pp. 1-7, doi: 10.1109/ICDCOT61034.2024.10515556.

[13] B. Shivkumar, J. Murphy, L. Ziarek. Real-time MLton: A Standard ML runtime for real-time functional programs. Journal of Functional Programming. 2021;31:e19.

[14] O. Delgadillo, B. Blieninger, J. Kuhn and U. Baumgarten, "An Architecture to Enable Machine-Learning-Based Task Migration for Multi-Core Real-Time Systems," 2021 IEEE 14th International Symposium on Embedded Multicore/Many-core Systems-on-Chip (MCSoC), Singapore, Singapore, 2021, pp. 405-412, doi: 10.1109/MCSoC51149.2021.00066.

[15] G. Premalatha and S. Abinaya, "An Effective Implementation of Vehicle Document Verification Using Genetic Algorithm," 2021 International Conference on System, Computation, Automation and Networking (ICSCAN), Puducherry, India, 2021, pp. 1-6.

[16] Park S, Kwon M-Y, Kim H-K, Kim H. Execution Model to Reduce the Interference of Shared Memory in ARINC 653 Compliant Multicore RTOS. Applied Sciences. 2020; 10(7):2464.

[17] Kabilesh, S. K., et al. "Resemblance of Real Time Scheduling Algorithms for Real Time Embedded Systems." Journal of Optoelectronics and Communication 2.3 (2020).

[18] B. Kim and H. Yang, "Reliability Optimization of Real-Time Satellite Embedded System Under Temperature Variations," in IEEE Access, vol. 8, pp. 224549-224564, 2020.

[19] Liew, JY., Ng, KH., Khor, KC., Tee, KY. (2024). Evaluating Path-Finding Algorithms for Real-Time Route Recommendation System Built using FreeRTOS. In: Ghazali, R., Nawi, N.M., Deris, M.M., Abawajy, J.H., Arbaiy, N. (eds) Recent Advances on Soft Computing and Data Mining. SCDM 2024. Lecture Notes in Networks and Systems, vol 1078. Springer, Cham. https://doi.org/10.1007/978-3-031-66965-1_17

[20] Vignesh Manjunath, Marcel Baunach, A framework for static analysis and verification of low-level RTOS code, Journal of Systems Architecture, Volume 154, 2024, 103220, ISSN 1383-7621, https://doi.org/10.1016/j.sysarc.2024.103220.

[21] Haur, I., Béchennec, JL., Roux, O.H. (2022). Formal Verification of the Inter-core Synchronization of a Multi-core RTOS Kernel. In: Riesco, A., Zhang, M. (eds) Formal Methods and Software Engineering. ICFEM 2022. Lecture Notes in Computer Science, vol 13478. Springer, Cham. https://doi.org/10.1007/978-3-031-17244-1_9

[22] Xiongli Gu, Peng Liu, Mei Yang, Jie Yang, Cheng Li, Qingdong Yao, An efficient scheduler of RTOS for multi/many-core system, Computers & Electrical Engineering, Volume 38, Issue 3, 2012, Pages 785-800, ISSN 0045-7906, https://doi.org/10.1016/j.compeleceng.2011.09.009.

[23] Merchant, S., & Dedhia, K. Performance comparison of rtos. Dept of Computer Science Columbia University.

[24] B. Savenko, A. Kashtalian, S. Lysenko, O. Savenko, "Malware Detection By Distributed Systems with Partial Centralization," 2023 IEEE 12th International Conference on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications (IDAACS), Dortmund, Germany, 2023, pp. 265-270, doi: https://doi.org/10.1109/IDAACS58523.2023.10348773

[25] Savenko, O., Sachenko, A., Lysenko, S., Markowsky, G., N., Vasylkiv, (2020). Botnet detection approach based on the distributed systems. International Journal of Computing, 19(2), 190-198. https://doi.org/10.47839/ijc.19.2.1761 43

[26] O. Savenko, S. Lysenko, A. Kryshchuk, Y. Klots / Proceedings of the 7-th IEEE International Conference on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications, Berlin, September 12–14, 2013. Berlin, 2013. Pp. 363–368.

[27] O. Pomorova, O. Savenko, S. Lysenko, A. Kryshchuk. Multi-Agent Based Approach for Botnet Detection in a Corporate Area Network Using Fuzzy Logic. Communications in Computer and Information Science. 2013. Vol. 370. PP.243-254, ISSN: 1865-0929.

[28] I. Zasornova, M. Fedula, A. Rudyi, Optimization of cyber-physical system parameters based on intelligent IoT sensors data, Computer systems and information technologies 2 (2024) 53–58. https://doi.org/10.31891/csit-2024-2-7.