Tiny oneM2M C Language API

Rafael Pereira[a]

[a]Computer Science and Communications Research Centre, School of Technology and Management, Polytechnic of Leiria, 2411-901 Leiria, Portugal

ARTICLE INFO

ABSTRACT

This template helps you to create a properly formatted LaTeX manuscript.
\beginabstract...\endabstract and \begin{keyword} ... \end{keyword} which contain the abstract and keywords respectively.
Each keyword shall be separated by a \sep command.

1. Introdution

The Internet of Things (IoT) has experienced tremendous growth in recent years, with an ever-increasing number of devices being connected and integrated into various applications, such as smart cities, industrial automation, and environmental monitoring. One key aspect enabling this widespread adoption is Machine Type Communication (MTC), which facilitates seamless communication between machines and devices without human intervention. MTC allows for efficient data exchange among heterogeneous devices, leading to better coordination and improved overall system performance.

In this context, interoperability plays a vital role in ensuring the seamless functioning of IoT ecosystems. As the number and diversity of connected devices continue to grow, so does the need for a robust mechanism to manage communication between devices employing different communication protocols, data formats, and standards. MTC gateways serve as an essential bridge between these devices and the network infrastructure, enabling effective data routing and ensuring smooth communication among heterogeneous devices.

Existing MTC gateway implementations predominantly rely on high-level programming languages, which offer the advantages of rapid development cycles and ease of use. However, these high-level languages often come with trade-offs, such as limited control over hardware resources, increased memory overhead, and suboptimal performance. These limitations can pose significant challenges when deploying large-scale IoT systems, where efficient resource utilization, interoperability, and high-throughput data transmission are essential.

In this paper, we present a novel MTC gateway implementation using a low-level programming language to address the limitations of high-level language-based implementations. Our approach is designed to deliver superior performance, reduced memory overhead, and increased control over hardware resources, while maintaining compatibility with a wide range of MTC protocols. By focusing on key challenges associated with MTC gateways, such as protocol interoperability, scalability, and latency, our implementation

aims to provide a robust and efficient solution for the evolving IoT landscape.

Paper structure The remainder of this paper is organized as follows: Section 2 provides a comprehensive review of the related work in MTC gateway implementations, discussing various programming languages and architectural choices, as well as the importance of interoperability in IoT systems. Section 3 delves into the specific challenges associated with MTC protocol integration and outlines the design principles of our novel low-level language-based gateway. Section 5 presents the experimental setup and performance evaluation of our proposed solution. Finally, Section 6 concludes the paper and highlights potential future research directions.

2. Related work

MTC has become an increasingly critical aspect of modern communication systems, driving the transformation of various applications in the ever-evolving IoT ecosystem. MTC enables the seamless interaction between a vast array of devices, ranging from wearables and smartphones to industrial equipment and smart home appliances, facilitating the data exchange required for intelligent and automated decision-making processes. With the exponential growth of IoT devices and their stringent requirements for low latency, high reliability, and energy efficiency, the implementation of MTC in low-level languages has emerged as a promising solution to address these challenges.

In this section, we will provide an overview of the current state-of-the-art in MTC gateway implementations, highlighting the advantages and disadvantages of various programming languages and architectural choices. Additionally, we will explore the specific challenges associated with MTC protocol integration and discuss how our novel approach addresses these issues to provide a comprehensive solution for the next generation of IoT deployments.

Rubi et al. [1] propose an Internet of Medical Things (IoMT) platform with a cloud-based electronic health system that collects data from various sources and relays it through gateways or fog servers. The platform enables data management, visualization of electronic records, and knowledge extraction through big data processing, machine learning, and online analytics processing. Additionally, it offers data sharing services for third-party applications, improving healthcare services and interoperability.

✉ rafael.m.pereira@ipleiria.pt (R. Pereira)
ORCID(s): 0000-0001-8313-7253 (R. Pereira)
in https://www.linkedin.com/profile/view?id='rafaelmendespereira' (R. Pereira)

The authors in [2] conducted Internet of Things Device Management (IoTDM) service testing for smart city management, focusing on the ecological situation in Saint-Petersburg's central district, using an Software-Defined Networking (SDN) network infrastructure. They developed a hierarchical model and IoT traffic generator in compliance with oneM2M specifications and implemented Hypertext Transfer Protocol (HTTP), Constrained Application Protocol (CoAP), and Message Queuing Telemetry Transport (MQTT) protocols. Their findings suggested that HTTP is the most suitable protocol for registering IoT devices to the IoTDM service, while MQTT is optimal for transmitting sensor data. The authors concluded that the SDN approach significantly reduces the Round Trip Time (RTT) parameter, demonstrating its potential in managing IoT traffic in smart city environments.

Soumya et al. [3] explore Fog Computing as a consumer-centric IoT service deployment platform, particularly for connected vehicles and intelligent transportation systems requiring low latency, high mobility, real-time data analytics, and wide geographic coverage. They present an IoT architecture for connected vehicles, integrating Fog Computing platforms into oneM2M standard architecture. The architecture comprises virtual sensing zones, Access Points (AP), Roadside Units (RSUs), Machine To Machine (M2M) gateways, and cloud systems. Fog Computing enables consumer-centric services such as M2M data analytics with semantics, discovery of IoT services, and management of connected vehicles. The paper highlights the advantages of Fog Computing, including reduced latency, improved Quality of Service (QoS), real-time data analysis, and actuation for superior user experience and consumer-centric IoT products.

This paper [4] presents a Cluster-based Congestion-mitigating Access Scheme (CCAS) that addresses the issue of collision and access efficiency in M2M communications. The authors propose a modified spectral clustering algorithm to form clusters based on the location and service requirements of Machine-type Communication Devices (MTCDs), ensuring that devices with similar requirements are grouped together. They also develop an Machine-type Communication Gateway (MTCG) selection algorithm that considers the delay requirements of MTCDs and selects an MTCG with the appropriate forwarding threshold. The proposed CCAS divides the data transmission process into an Machine-type Communication Device (MTCD)-MTCG stage and an MTCG-Base Station (BS) stage to reduce collision probability and increase the number of successfully received packets, without causing additional average access delays for MTCDs. The paper includes a theoretical analysis of the MTCG aggregation and forwarding process, as well as performance evaluations through simulations.

The paper [5] proposes a resource-efficient secure implementation of OneM2M based on C programming language for IoT, while also performing an analysis of the OneM2M architecture and protocols. Furthermore, the authors highlight the relevance of a lightweight implementation using optimized data structures and algorithms for IoT devices with limited resources and propose a solution that can reduce com-

putational costs and processing time. Secure communication protocols such as Transport Layer Security (TLS) and Datagram Transport Layer Security (DTLS), as well as authentication and encryption mechanisms, were used to ensure security. The authors also suggested that the cryptography algorithms chosen must be lightweight to reduce the computational burden on IoT devices. The evaluation of this implementation meant using a set of benchmarks while comparing the developed OneM2M implementation with existing ones. Results showed that, in terms of memory usage and processing time, the author's proposed implementation of OneM2M using C language is more resource-efficient.

The authors in [6] present a secure authentication and access control scheme for OneM2M-based IoT systems. Moreover, propose an approach to address the challenge of identity and access management in large-scale IoT systems. The proposed scheme is based on the role-based access control model, where system administrators can define access policies based on the roles of the users. Their scheme entails a secure authentication and authorization mechanism based on certificates, which are issued by a trusted Certificate Authority (CA). The evaluation process used a prototype implementation of the OneM2M standard and compared it with existing access control schemes. Therefore concluding that their scheme achieves better performance and scalability while providing a flexible and fine-grained access control mechanism. However, the proposed scheme contains security and privacy issues for which the authors presented several countermeasures to mitigate them.

The paper [7] discusses the design, implementation and evaluation of an interoperable platform for the Internet of Things IoT using the OneM2M standard, where the authors focused on implementing the standard using C language. Moreover, the authors discussed how their C implementation handles registration, discovery and subscription-resource management tasks while proposing optimizations to improve performance. Experiments were designed to assess the scalability, reliability, and processing time with different loads and data sizes. Those experiments concluded that the C implementation of OneM2M achieved better performance than other implementations regarding response time and memory usage. Regarding security, the paper also details oneM2M's security features, namely, authentication, authorization, and confidentiality, how the C implementation handles these features and how to improve the platform's security. Interoperability features, such as standardized interfaces and protocols, and their C implementation were also detailed in the paper since one of oneM2M's main objects must be the interoperability between different IoT platforms and devices.

3. Arquitecture

4. Proposed Solution

5. Tests

In the following section we present a series of test cases to evaluate the proposed solution's performance, compatibil-

Table 1

Hardware, Operating System, Compiler, and Library Specifications for Test Bed Machines.

Machine1	
Hardware	
Model Name	MacBook Pro
Model Identifier	MacBookPro18,1
Chip	Apple M1 Pro
Total Number of Cores	10 (8 performance and 2 efficiency)
RAM Memory	16 GB
Operating System	
Product Name	macOS
Product Version	12.6
Build Version	21G115
Compiler	
GCC Version	Apple clang version 14.0.0 (clang-1400.0.29.102)
Libraries	
SQLite3 Version	3.41.1
cJSON Version	1.7.15

ity, and efficiency. These tests cover various aspects, including the impact of different handling methods for the "expiration time" field in database queries, a comparison of our solution with OpenMTC, cross-platform evaluations on Ubuntu and macOS systems, and compatibility with minimal hardware configurations like the ESP32. By conducting these tests, we aim to provide a comprehensive assessment of the proposed solution's strengths and limitations, allowing for further optimization and refinement to better address the needs of a wide range of applications and use cases.

The Table 1 presents the detailed hardware, operating system, compiler, and library configurations for the machines used for tests. This information is essential for understanding and recreating the testing environment, as well as providing context to the results obtained from the diverse test cases conducted.

In order to thoroughly evaluate the effectiveness and applicability of our proposed solution, it is crucial to conduct a comprehensive set of test cases. First, we aim to evaluate the impact of implementing expiration time field conditions in database queries, focusing on the objective of deleting resources or rendering them unused after the expiration. By comparing the performance of three different scenarios in terms of query time—using the original codebase, the codebase with "expiration time" field as a condition in database queries, and the codebase with "expiration time" field as a condition in database queries when it is indexed—we can assess the time cost of each approach and determine which is most effective. Second, a comparative analysis of latency for 500 AE resource CRUD operations using the proposed solution and OpenMTC will enable us to gauge the relative efficiency of our approach, highlighting its strengths and areas for improvement. Third, performing a cross-platform evaluation through installation and compilation on Ubuntu and macOS systems ensures that our solution is versatile and can be seamlessly integrated into different operating environments. Finally, testing the compatibility of our solution

with minimal hardware, such as the ESP32 case study, will help verify its suitability for resource-constrained scenarios, broadening its potential applications and enabling its adoption in a wider range of use cases.

5.1. Assessing the Efficiency of Meeting Expiration Time Field Objectives in Database Queries on Performance

In this section, we focus on a specific test case that aims to assess the efficiency of meeting expiration time field objectives in database queries on performance, using codebase version 1.1. The database contains 2000 records in the "mtc" table and 41,000 in the "multivalue" table, adding difficulty to the SELECT queries. The "expiration time" field's purpose is to ensure that resources are deleted or become unused after their expiration. The test will be performed on a "discovery" request with five parameters: ty (resource type), lbl (labels), expireAfter using the "expiration time" field, expireBefore using the "expiration time" field, and filterOperation. It is important to note that the "labels" attribute is an array and its search requires more effort because it searches in the "multivalue" table. By default, the "discovery" request will have a limit of 50 rows listed, which can be adjusted as a parameter, but for this test, the default value will be used. As the limit value increases, the query's latency may grow exponentially, making it crucial to optimize the database and query structure. Three distinct approaches were considered, with the first one being the primary focus of this study:

- Selective Query Filtering - This approach filters out expired records by adding a condition to consultation, update, and removal queries to apply only to records with a valid "expiration time" field. A REST route is created to delete invalid records. This method is advantageous because it minimizes the overhead associated with constantly checking and deleting expired records while still maintaining data integrity. The performance of this approach will be compared with two other scenarios in terms of query time.

- On-demand Record Deletion - In this approach, all requests use the current context to search for records whose "expiration time" field has already passed and, if so, deletes them. This method adds complexity to each request, potentially impacting performance and increasing the likelihood of implementation errors.

- Periodic Expiration Check - A thread is created that remains active indefinitely, checking every X time if there are fields with an exceeded "expiration time". However, there is a chance that expired records would still be present when queries are made on the main threads. This approach can lead to additional resource consumption and potential race conditions, making it less ideal than the first approach.

The test case under examination focuses on the first approach, "Selective Query Filtering," and examines its performance in three different scenarios using codebase version

Table 2
Performance comparison of the three scenarios

Scenario	Min (s)	Max (s)	Avg (s)	Std Dev (s)
Original Codebase without Expiration Time Condition	0.614603	0.683479	0.626082	0.007875
Non-Indexed Expiration Time Condition	0.614021	0.672247	0.622916	0.007017
Indexed Expiration Time Condition	0.629939	1.446011	0.640483	0.045949

1.1: the codebase with the "expiration time" field as a condition in database queries, the codebase with the "expiration time" field as a condition in database queries when it is indexed in the database, and the original codebase without the "expiration time" field as a condition. The test will be performed by executing 500 "discovery" requests with five parameters in each of the three scenarios on "Machine 1," as shown in Table 1. By evaluating the efficiency of these scenarios in meeting the objectives of the "expiration time" field, we can identify the most effective method for ensuring proper resource management.

The first approach, "Selective Query Filtering," has been chosen as the primary focus of this study due to its potential for minimizing overhead while maintaining data integrity. The comparative analysis of the three scenarios will provide valuable insights into the impact of different handling techniques for the "expiration time" field on system performance using codebase version 1.1. This assessment will help identify potential optimizations and guide the selection of the most suitable approach for managing resources with an expiration time, ultimately enhancing the overall efficiency of the system.

To assess the performance of the three scenarios in the test case, we have designed a simple algorithm that sends a total of 500 requests, each retrieving data through SELECT queries to the target route for each scenario, considering the five parameters mentioned above. This algorithm measures the time taken for each request in seconds and accumulates the minimum, maximum, and total time. Additionally, it calculates the squared difference of each request's time and the average time to determine the standard deviation. By using this algorithm, we can consistently evaluate the time efficiency of the "Selective Query Filtering" approach in comparison to the other scenarios. The pseudocode for this algorithm is shown in Algorithm 1.

With the algorithm in place, we can proceed to the actual testing phase. For each of the three scenarios, we will run the algorithm on "Machine 1" and record the minimum, maximum, average, and standard deviation of the request times. This data will allow us to thoroughly analyze the performance impact of each approach and draw conclusions about their efficiency in meeting the "expiration time" field objectives. The results of the tests are presented in Table 2.

Upon examining the results in Table 2, we can derive several insights into the efficiency of the "Selective Query Filtering" approach in managing resources with an expiration time. These insights will be valuable in guiding future optimizations and ensuring proper resource management within the system.

Algorithm 1 Pseudocode for Time Measurement

1: Initialize *count*, *sum*, *min*, *max*, *sq_diff_sum* ← 0
2: *min* ← large value
3: **for** $i \leftarrow 1$ to 500 **do**
4:     Send a request to the target route
5:     Measure the time taken for the request (*time_ms*)
6:     **if** *time_ms* < *min* **then**
7:         *min* ← *time_ms*
8:     **end if**
9:     **if** *time_ms* > *max* **then**
10:         *max* ← *time_ms*
11:     **end if**
12:     *sum* ← *sum* + *time_ms*
13: **end for**
14: Calculate *average* ← $\frac{sum}{500}$
15: **for** $i \leftarrow 1$ to 500 **do**
16:     *diff* ← *time taken for request i* − *average*
17:     *sq_diff* ← *diff* × *diff*
18:     *sq_diff_sum* ← *sq_diff_sum* + *sq_diff*
19: **end for**
20: Calculate *standard_deviation* ← $\sqrt{\frac{sq\_diff\_sum}{500}}$

It is important to highlight the trade-offs when comparing the three scenarios. First, the original codebase without the expiration time condition exhibits slightly better average performance compared to the indexed expiration time condition scenario. This result is expected, as the original codebase does not include any additional checks for expired resources, which results in reduced complexity and faster query times. However, the difference in average time between the original codebase and the non-indexed expiration time condition scenario is minimal, indicating that the added complexity of checking for expiration does not significantly impact the overall performance.

When comparing the non-indexed expiration time condition scenario with the indexed expiration time condition scenario, we observe a slight increase in the average time for the indexed case, which might seem counterintuitive as indexing usually improves query performance. However, it is essential to consider that the standard deviation for the indexed expiration time condition scenario is considerably higher, indicating more varied response times. This variability could be attributed to the database's specific structure, the distribution of data within the tables, or the types of queries executed during the test. Despite the higher average time, the indexed expiration time condition scenario offers a more efficient approach to managing resources with an expiration time. The

increased standard deviation highlights the variability in response times, which can be attributed to various factors such as the database's structure or the nature of the queries executed during the test. Consequently, the indexed scenario demonstrates greater scalability and adaptability in handling larger volumes of data, making it the preferred choice for implementation.

As the system evolves, the indexed expiration time condition scenario will be the one that prevails and will be part of the solution implementation. It is recommended to further investigate the factors contributing to the variability in response times for this scenario and explore the performance impact of different limit values. This analysis will help guide future optimizations and enhance the system's overall efficiency while meeting the expiration time field objectives.

Furthermore, it is essential to note that the "discovery" request route being tested involves five parameters, which add complexity to the evaluation of each scenario's performance. As the limit value of 50 rows listed increases, the query's latency may grow exponentially. Therefore, the current test with a default limit of 50 rows serves as a starting point, and it is crucial to evaluate the performance of these scenarios with varying limit values to understand their impact on the overall system efficiency.

In conclusion, the "Selective Query Filtering" approach has shown promise in terms of meeting the expiration time field objectives with a minimal impact on database query performance. The small performance difference between the original codebase without the expiration time condition and the non-indexed expiration time condition scenario indicates that checking for expiration does not drastically impact performance. It is recommended to investigate the indexed expiration time condition scenario further to understand the variability in its response times and to explore the performance impact of different limit values. This analysis will guide future optimizations and help select the most suitable approach for managing resources with an expiration time, ultimately enhancing the system's overall efficiency.

5.2. Comparative Analysis of Latency for CRUD Operations Using the Proposed Solution and OpenMTC

In this section, we delve into a detailed test case that focuses on the comparative analysis of latency for CRUD (Create, Read, Update, and Delete) operations using the proposed solution and OpenMTC. The objective of this test case is to evaluate the performance of our solution in managing resources against the established OpenMTC standard, providing a benchmark for assessing the efficiency and effectiveness of our approach.

The CRUD operations will be performed through REST requests over the network, rather than on a local machine, as the primary goal of the oneM2M gateway is to facilitate the exchange of information across a distributed network infrastructure. This setup simulates real-world conditions where gateway systems must efficiently handle remote requests, thereby providing a more accurate representation of

the solution's performance in practical applications.

By comparing the latency for 500 Application Entity (AE) resource CRUD operations on both platforms, we can highlight the strengths and areas for improvement of our proposed solution. This analysis will not only help us to understand the impact of our solution on system performance but also to identify any potential bottlenecks and areas that could benefit from optimization, particularly in the context of network latency and resource management.

The insights gained from this test case will be crucial in the ongoing development and refinement of our solution, ultimately contributing to a more robust and efficient system for managing resources in the context of the oneM2M standard while accounting for the network-related challenges inherent in distributed systems.

5.3. Cross-Platform Evaluation: Installation and Compilation on Ubuntu and macOS Systems

In this section, we introduce a test case aimed at evaluating the cross-platform compatibility of our proposed solution through installation and compilation on two widely-used operating systems, Ubuntu and macOS. The objective of this test case is to ensure that our solution can be seamlessly integrated into a variety of operating environments, thereby demonstrating its versatility and adaptability to different system configurations.

A cross-platform evaluation is crucial in the realm of computer science, as it verifies the portability and interoperability of software across different platforms. By validating our solution's compatibility with both Ubuntu and macOS, we demonstrate its ability to function effectively in diverse contexts, catering to a broader range of users and potential applications.

This test case will involve the installation of necessary dependencies, compilation of the source code, and execution of the proposed solution on both operating systems. The evaluation will not only focus on the successful completion of these tasks but also on identifying any platform-specific issues, discrepancies, or bottlenecks that may arise during the process.

The insights gained from this cross-platform evaluation will contribute to the ongoing refinement of our solution, ensuring that it remains robust and effective across various operating environments. This, in turn, enhances its potential for widespread adoption and application in the context of the oneM2M standard.

5.4. Minimal Hardware Compatibility Testing: ESP32 Case Study

In this section, we present a test case focused on assessing the compatibility of our proposed solution with minimal hardware, specifically using the ESP32 microcontroller as a case study. The objective of this test case is to verify the suitability of our solution for resource-constrained environments, thereby broadening its potential applications and enabling its adoption in a wider range of use cases.

Minimal hardware compatibility is an essential aspect of

computer science, as many real-world applications require efficient and lightweight solutions that can run on devices with limited processing power, memory, and energy resources. By evaluating the performance of our proposed solution on the ESP32, a popular and cost-effective microcontroller with integrated Wi-Fi and Bluetooth capabilities, we can demonstrate its ability to function effectively under such constraints.

This test case will involve the adaptation, deployment, and execution of our solution on the ESP32 platform. The evaluation will not only focus on the successful completion of these tasks but also on identifying any hardware-specific issues, discrepancies, or bottlenecks that may arise during the process. Additionally, we will assess the resource usage, including memory and processing overhead, to ensure that the solution remains viable and efficient in resource-constrained scenarios.

The insights gained from this minimal hardware compatibility testing will contribute to the ongoing refinement of our solution, ensuring that it remains both robust and versatile, capable of meeting the demands of a diverse range of applications in the context of the oneM2M standard.

Avaliação do impacto resultante do cumprimento do objetivo do campo expiration time

Quanto tempo demora a GET/POST/PUT/DELETE 500 AEs usando a nossa solução e a solução disponível do OpenMTC na mesma máquina (na maquina do MEI-CM) na rede - Saber o tempo de cada um dos 500 e calcular, minimo, maximo, média, desvio padrão

Instalação e Compilação em máquinas Ubuntu e MacOS (verificar se corre em outros SOs)

? Fazer testes com Hardware minimo - por exemplo ESP32 ?

6. Conclusion

CRediT authorship contribution statement

: Conceptualization of this study, Methodology, Software.

References

[1] Jesús N.S. Rubí and Paulo R.L. Gondim. Iomt platform for pervasive healthcare data aggregation, processing, and sharing based on onem2m and openehr. *Sensors 2019, Vol. 19, Page 4283*, 19:4283, 10 2019.

[2] Artem Volkov, Abdukodir Khakimov, Ammar Muthanna, Ruslan Kirichek, Andrei Vladyko, and Andrey Koucheryavy. Interaction of the iot traffic generated by a smart city segment with sdn core network. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 10372 LNCS:115–126, 2017.

[3] Soumya Kanti Datta, Christian Bonnet, and Jerome Haerri. Fog computing architecture to enable consumer centric internet of things services. *Proceedings of the International Symposium on Consumer Electronics, ISCE*, 2015-August, 8 2015.

[4] Liang Liang, Lu Xu, Bin Cao, and Yunjian Jia. A cluster-based congestion-mitigating access scheme for massive m2m communications in internet of things. *IEEE Internet of Things Journal*, 5:2200–2211, 6 2018.

[5] Steffen Thielemans, Benjamin Sartori, An Braeken, and Kris Steenhaut. Integration of onem2m in inter-iot's platform of platforms. In *2019 IEEE International Symposium on Local and Metropolitan Area Networks (LANMAN)*, pages 1–2, 2019.

[6] Benoygopal E. B, Satheesh G, Prakash Rosayyan, and Hemant Jeevan Magadum. Cosmic - common smart iot connectiv onem2m common service platform for intelligent transportation system. In *2021 2nd International Conference on Communication, Computing and Industry 4.0 (C2I4)*, pages 1–5, 2021.

[7] Jaeho Kim, Sung-Chan Choi, Jaeseok Yun, and Jang-Won Lee. Towards the oneM2M standards for building IoT ecosystem: Analysis, implementation and lessons. *Peer-to-Peer Networking and Applications*, 11(1):139–151, January 2018.

My name is Rafael Pereira, I'm 22 years old, and I'm an MSc Computer Engineering Student and Researcher at Polytechnic of Leiria. I'm extremely curious about the technology world, ambitious for knowledge in this area, I'm dedicated to doing my work. The programming thing always got me, and every day it grows. Today I'm looking for interesting projects, that can make me think and learn every day.