telecom

MDPI

Article

Evaluating Binary Serialization Protocols for IoT/M2M Applications over Hybrid Terrestrial and Non-Terrestrial Networks

Natesh Kumar [1], Mariano Falcitelli [2,*], Francesco Kotopulos De Angelis [3], Paolo Pagano [2] and Sandro Noto [2]

[1] Independent Researcher, 56100 Pisa, Italy; nateshkumar2014@gmail.com
[2] CNIT-National Inter-University Consortium for Telecommunications, Photonic Networks & Technologies National Laboratory, 56124 Pisa, Italy; paolo.pagano@cnit.it (P.P.); sandro.noto@cnit.it (S.N.)
[3] Scuola Superiore Sant'Anna, Telecommunications, Computer Engineering, and Photonics Institute (TeCIP), 56127 Pisa, Italy; f.kotopulosdeangelis@santannapisa.it
* Correspondence: mariano.falcitelli@cnit.it; Tel.: +39-050882283

Abstract

The rapid growth of Internet of Things (IoT) deployments in hybrid terrestrial/non-terrestrial networks (TN/NTN) faces a major bottleneck: the verbosity of standard data formats like JSON. This is critical for large-scale M2M systems tracking and monitoring multimodal dry containers, where devices must comply with the strict message-size limits of commercial satellite IoT (around 160 bytes per message). We present a comparative evaluation of four device-friendly binary serialization protocols (CBOR, MessagePack, Protocol Buffers, and a custom Struct+Zlib hybrid) targeted at battery-powered microcontrollers. Using a horizontally scalable testbed with up to 2000 concurrent devices and the oneM2M standard framework, we assess payload efficiency, throughput, latency, and maintainability. Only Protocol Buffers and Struct+Zlib meet NTN message-size limits, with Protocol Buffers providing the best trade-off between performance and long-term maintainability. Real-world validation with the Astrocast LEO satellite platform and the oneM2M Mobius framework confirms these results. Cost analysis suggests potential savings exceeding €62,000 per month for a 10,000-device maritime fleet, demonstrating both technical feasibility and economic viability. This study provides a methodological framework for designing efficient, scalable IoT systems in hybrid TN/NTN networks, offering practical guidance for global container tracking and monitoring deployments.

Keywords: Internet of Things; machine-to-machine; binary serialization; non-terrestrial networks; satellite IoT; Protocol Buffers; oneM2M; payload optimization; 5G NTN

Academic Editor: Panagiotis Gkonis

Received: 24 February 2026
Revised: 24 March 2026
Accepted: 7 April 2026
Published: 10 April 2026

1. Introduction

The fourth industrial revolution is fundamentally predicated on the deep integration of digital intelligence into physical infrastructure through the Internet of Things (IoT) and machine-to-machine (M2M) communication systems [1]. With forecasts predicting tens of billions of connected devices and potential economic impacts exceeding $11 trillion annually [2], the efficiency of data transmission has evolved from a technical detail to a critical determinant of system scalability, operational cost, and deployment feasibility.

This challenge is particularly acute in sectors requiring global coverage, such as maritime logistics, where IoT devices must operate seamlessly across either high-bandwidth terrestrial networks (TNs) or severely constrained non-terrestrial networks (NTNs).

Consider the mission-critical use case of tracking and monitoring multimodal dry containers during ocean voyages: smart sensors equipped with monitoring capabilities leverage 5G or LTE connectivity while in port; once vessels are at open sea, they must rely exclusively on satellite links [3]. The dichotomy between these operational environments creates a fundamental architectural challenge that cannot be addressed through network-layer solutions alone, as continuous global asset tracking demands a seamless transition between radically different connectivity paradigms.

The oneM2M standard, a global framework for IoT interoperability developed by leading standards development organizations (ETSI, ATIS, ARIB), has adopted JavaScript Object Notation (JSON) as its primary data interchange format [4]. While JSON's human-readable structure and universal support simplify development, its inherent verbosity (characterized by repetitive text-based keys and UTF-8 encoding overhead) creates substantial performance penalties. For a representative maritime telemetry payload containing 20 fields (GNSS coordinates, environmental sensors, battery status, and container metadata), the uncompressed JSON representation exceeds 400 bytes [5–9].

This overhead becomes untenable in the satellite IoT domain, particularly for operational commercial services. Among LEO satellite IoT operators, Astrocast has emerged as a fully operational commercial platform with well-defined technical specifications and established service-level characteristics [10,11]. Unlike experimental constellations in deployment phases, Astrocast's operational infrastructure serves as a practical benchmark for evaluating real-world constraints. The platform imposes strict message-size limits below 160 bytes to prevent fragmentation and optimize spectral efficiency (constraints reflecting fundamental physical-layer limitations common to compact LEO IoT architectures [10]). This operational reality makes Astrocast an appropriate benchmark for evaluating payload optimization strategies in commercially deployed satellite networks.

Simultaneously, 3GPP Release 17 and later releases extend NB-IoT and LTE-M standards to NTNs, enabling standardized cellular IoT over both GEO and LEO satellites, though with inherent trade-offs in latency, coverage, and commercial maturity [12,13]. GEO-based systems provide stable wide-area coverage but suffer from 500 to 600 ms round-trip delays and cannot serve polar regions [14,15], while LEO constellations offer lower latency and global reach but require continuous satellite handovers [16,17]. The parallel development of standards-based (3GPP) and commercial LEO (Astrocast and others) approaches represents complementary paths toward global IoT connectivity, with commercial services currently providing the most mature deployment model for mission-critical applications such as maritime container tracking.

The research question is therefore clear: which payload optimization strategy can satisfy both the <160-byte constraint of operational commercial NTN links and the performance requirements of large-scale TN deployments, while remaining implementable on resource-constrained, battery-powered microcontrollers typical of multimodal dry container monitoring devices?

Contributions

This paper makes the following key contributions:

1. Generalizable Validation Methodology: A systematic, reusable framework for evaluating binary serialization protocols in hybrid terrestrial/non-terrestrial (TN/NTN) IoT/M2M deployments, applicable to any NTN message-size constraint. The methodology integrates device-side feasibility assessment, server-side scalability testing, and real-world operational validation, providing a comprehensive approach that can be adapted to different satellite IoT platforms and payload limits beyond the specific Astrocast constraints examined in this study.

2. Comprehensive Protocol Evaluation: A systematic comparison of four binary serialization approaches (schema-less formats (CBOR [18], MessagePack [19]), a custom Struct+Zlib hybrid [20], and the schema-driven Protocol Buffers [21]) evaluated against device-side feasibility and server-side scalability metrics.
3. Rigorous Performance Analysis: Quantitative assessment through a containerized, load-balanced testbed processing up to 2000 concurrent IoT devices transmitting simultaneously, measuring throughput (RPS), latency percentiles, failure rates, and resource utilization under realistic network conditions representative of massive-scale maritime deployments.
4. NTN Constraint Validation: Empirical demonstration where only Protocol Buffers (138 bytes) and Struct+Zlib (127 bytes) achieve the 67–69% message-size reduction necessary to meet commercial satellite IoT constraints, with experimental validation using the Astrocast operational platform (the most mature LEO IoT constellation with well-defined message-size accounting).
5. Architectural Analysis: A Figure of Merit framework (Section 4.5.3) integrating throughput, latency, and reliability, revealing Protocol Buffers' superior balance and highlighting the long-term maintainability advantages of schema-driven approaches.
6. Economic Impact Quantification: Fleet-scale cost analysis demonstrating potential monthly savings exceeding €62,000 for 10,000-device deployments, validating both technical feasibility and commercial viability.
7. Practical Implementation Guidance: Evidence-based recommendations for system architects deploying global container tracking and monitoring systems, distinguishing optimal strategies for terrestrial-only versus hybrid TN/NTN maritime logistics deployments.

The remainder of this paper is organized as follows: Section 2 provides the technical background on oneM2M, NTN architectures, and serialization protocols; Section 3 reviews related work; Section 4 details the experimental design; Section 5 presents quantitative findings; Section 6 analyzes implications and validates the approach; and Section 7 concludes with recommendations and future directions.

2. Background and State of the Art

2.1. The oneM2M Standard and Resource-Oriented Architecture

The oneM2M Partnership Project addresses IoT fragmentation through a horizontal, common service layer that abstracts underlying network technologies [4]. This architecture is critical for hybrid TN/NTN deployments where devices must seamlessly transition between heterogeneous connectivity modes (5G/LTE-M in port, LEO satellites at sea) without application-layer reconfiguration. The oneM2M standard's design philosophy explicitly addresses the challenge of collecting and distributing massive volumes of IoT data from any network infrastructure (terrestrial cellular, Wi-Fi, satellite/NTN) using any application protocol (HTTP, CoAP, MQTT, WebSocket), making it especially well suited for global maritime container-tracking and monitoring scenarios [22].

Its functional architecture defines three core entities: Application Entities (AEs), which encapsulate application logic, Common Services Entities (CSEs), which provide middleware functions (data management, discovery, security), and Network Services Entities (NSEs), which represent the communication infrastructure. This layered design enables seamless operation across Wi-Fi, LTE-M, 5G, and satellite NTNs without application-layer modifications (a fundamental requirement for devices that must maintain continuous connectivity as containers transition from terrestrial to non-terrestrial networks during multimodal transport).

oneM2M follows a Resource-Oriented Architecture (ROA) in which entities are modeled as uniquely addressable resources in a hierarchical structure rooted at `<CSEBase>`. Key resource types include `<AE>` (registered applications), `<container>` (logical data folders), `<contentInstance>` (individual payloads), and `<subscription>` (event-driven notifications). RESTful APIs using standard CRUD primitives (CREATE, RETRIEVE, UPDATE, DELETE, mapped to HTTP POST, GET, PUT, DELETE respectively) enable standardized interactions regardless of the underlying network (terrestrial, cellular, or satellite NTNs).

For this research, the open-source Mobius CSE platform [23] provides the implementation foundation, demonstrating practical applicability and optimal performance within the oneM2M ecosystem [24]. Mobius's proven ability to handle large numbers of concurrent data streams from heterogeneous sources (combined with oneM2M's network-agnostic design) makes this framework particularly appropriate for evaluating payload optimization strategies in hybrid TN/NTN maritime scenarios where thousands of container-tracking and monitoring devices must simultaneously transmit telemetry data through diverse connectivity channels. The standard's protocol flexibility further ensures compatibility with the range of device capabilities and network conditions encountered across global supply chains.

2.2. Non-Terrestrial Networks for IoT

The integration of satellite links into IoT systems addresses the fundamental limitation of terrestrial coverage, which leaves vast oceanic and remote regions unconnected. Two primary architectural models exist [3]: (1) direct access, where end-devices communicate directly with satellites, and (2) backhaul aggregation, where local gateways collect data via terrestrial protocols and use satellites for wide-area network (WAN) transmission (this research adopts the former model).

3GPP Release 17 extended NB-IoT and LTE-M to NTNs, enabling standardized cellular IoT over satellites with adaptations for long propagation delays, Doppler shifts, and intermittent coverage [25]. GEO-based implementations offer continuous regional coverage but suffer from high latency (∼500–600 ms) and lack polar coverage [14,15], while LEO constellations provide lower latency and global reach but require frequent handovers [16,17].

The commercial satellite IoT landscape has evolved rapidly, with several operators deploying operational constellations offering diverse payload constraints (Table 1). Starlink's direct-to-device (D2D) service enables cellular connectivity via satellite, supporting LTE Cat-1 and Cat-1 Bis capabilities, with a commercial messaging service launched in early 2025 for T-Mobile (USA) and One NZ (New Zealand) customers [26,27]. The service targets IoT device connectivity starting in 2025, though European coverage remains under development. Swarm Technologies (acquired by SpaceX in 2021) operated a LEO constellation of compact nanosatellites optimized for low-power IoT messaging with global coverage [28]; however, the service was discontinued in March 2025 as SpaceX transitions toward Direct-to-Cell capabilities. Iridium's Certus family transceivers provide bidirectional short-burst data services for asset tracking and remote monitoring [29,30]. ORBCOMM operates a dedicated M2M constellation (OG2 satellites) serving maritime, transportation, and industrial applications with over 2.2 million subscriber devices, featuring VHF-band communications optimized for low-data-rate telemetry with typical payloads ranging from 6 to 250 bytes depending on message type [31,32]. Kinéis, the successor to the Argos system, deployed a constellation of 25 nanosatellites throughout 2024–2025 specifically designed for small IoT devices, offering two-way communications with a 30-byte message-size limit, 5–15 min revisit times globally, and targeting environmental monitoring, agriculture, and logistics applications [33,34].

Among these commercial platforms, Astrocast has emerged as a mature, fully operational LEO IoT service with well-defined technical specifications and established connectivity characteristics [10,11]. The platform provides bidirectional messaging with strict payload limits (typically below 160 bytes per message) optimized for battery-powered devices, making it particularly suitable for maritime container tracking and remote asset monitoring. Critically, Astrocast's commercial availability and documented API enable practical validation testing of payload optimization strategies in operational satellite environments, bridging the gap between laboratory experimentation and real-world deployment feasibility (a capability leveraged in this research for empirical validation (Section 5.7)).

Table 1. Payload size constraints across commercial satellite IoT platforms.

Platform	Max Payload (bytes)	Reference
Astrocast	<160	[10]
Iridium SBD (9601/9602)	340 (MO)/270 (MT)	[35]
ORBCOMM OG1/OG2	6–250 typical	[31]
Kinéis	30 (max)/19 (standard)	[34,36]
3GPP NB-IoT/LTE-M NTN	Protocol-dependent	[25]

2.3. Binary Serialization Protocols

2.3.1. Schema-Less Formats: CBOR and MessagePack

CBOR (Concise Binary Object Representation), standardized in RFC 8949 [18], and MessagePack [19] represent "binary JSON" approaches that preserve JSON's flexible data model while using compact byte-oriented encoding. Both are self-describing, requiring no predefined schema, which simplifies development. However, this flexibility incurs overhead: field names (keys) must be transmitted as strings within the payload. For a field like `{''rssi'': 28}`, CBOR encodes as 8 bytes (`a1 64 72 73 73 69 18 1c`), where the key "rssi" consumes 5 bytes (1-byte header + 4-byte UTF-8 string) [18,37].

2.3.2. Hybrid Compression: Struct+Zlib

This custom approach eliminates keys entirely through a two-stage process: (1) data serialization into a fixed-layout binary format using C-struct packing, and (2) general-purpose compression via Zlib. For the `rssi` field defined as a 1-byte unsigned integer (`B` in Python 3.12 struct format), `struct.pack('>B', 28)` yields a single byte (`0x1c`). Subsequent Zlib compression exploits redundancy across the full payload. While achieving maximal efficiency, this approach suffers from brittleness: any data structure change requires coordinated firmware updates across all devices and server-side logic. Embedded libraries like Miniz [20] confirm ESP32-class feasibility with a <20 KB RAM footprint.

2.3.3. Schema-Driven Format: Protocol Buffers

Google's Protocol Buffers [21] combine binary efficiency with formal schema definition via language-agnostic `.proto` files. Field names are replaced by compact numeric tags with wire-type indicators. For `uint32 rssi = 6;`, the encoding uses 2 bytes: a key byte combining field number six and wire type zero (`0x30`), followed by the varint-encoded value 28 (`0x1c`). This schema-driven design provides type safety, forward/backward compatibility, and automatic code generation. The nanopb library [38,39] delivers Protocol Buffers functionality on constrained devices with a 30–40 KB flash footprint and 2–4 KB of runtime RAM.

2.4. Resource-Constrained Device Environment

Battery-powered microcontrollers used in maritime IoT applications represent a class of hardware characterized by severe resource limitations. Typical devices in this category feature 32-bit processors operating at 160–240 MHz, SRAM capacities of 512 KB to 1 MB, and flash memory of 4–16 MB [40,41]. These constraints impose strict requirements on serialization libraries: the flash footprint must not preclude other firmware components (wireless protocol stacks, sensor drivers, power management), runtime RAM consumption must avoid system instability, and encoding latency impacts energy consumption by extending CPU active time.

For battery-powered container monitoring devices, the primary optimization goal is to minimize radio transmission time (the most power-intensive operation) by drastically reducing the message size [42]. A typical scenario illustrates the trade-off: while encoding a payload with Protocol Buffers may consume 60–300 microseconds of CPU time (requiring approximately 16–20 microCoulombs of charge), transmitting a 300-byte uncompressed JSON payload over satellite can consume 30–50 millijoules (three orders of magnitude more energy [38]). This fundamental asymmetry validates the strategy of accepting modest CPU overhead for encoding in exchange for dramatic reductions in radio airtime, thereby extending battery life and reducing operational costs in maritime deployments where physical device access is impractical.

3. Related Work

3.1. Binary Serialization Performance Studies

Prior research has well established the superiority of binary serialization over text-based formats. Biswal and Almallah [38] demonstrated the advantages of Protocol Buffers in payload size, processing speed, and power consumption on microcontrollers. Jackson et al. [43] evaluated over 143 streaming/serialization technology combinations, finding that protocol-based binary formats consistently deliver the highest performance. Maltsev and Muliarevych [44] confirmed median size reductions exceeding 30% compared to JSON, while Huseynov et al. [45] established that payload compactness impacts end-to-end latency more significantly than raw serialization speed (validating this work's focus on aggressive size reduction).

A comprehensive analysis by Luis et al. [46] evaluated multiple serialization methods specifically for 5G IoT contexts, comparing JSON, XML, CBOR, MessagePack, and PSON across metrics including message size, encoding/decoding time, and energy consumption. Their work demonstrated that binary formats consistently outperform text-based alternatives in resource-constrained scenarios. Among schema-less encodings that can be converted to JSON, PSON stands out for its efficiency and yields encoding sizes similar to MessagePack, with both achieving similar compression ratios (approximately 15% reduction versus JSON). Notably, their findings showed that PSON's compression performance is comparable to MessagePack while requiring slightly more complex implementation. Based on this equivalence in compression efficiency and MessagePack's broader ecosystem support and simpler deployment characteristics, our study focused exclusively on MessagePack as the representative schema-less binary format, avoiding redundant evaluation of protocols with similar performance profiles.

Specialized techniques can push optimization even further. Molina Araque et al. [47] showed that delta encoding for time-series data can reduce size by up to 91% compared with standard CBOR. Zandberg et al. [48] reported that CBOR is highly effective for federated learning in constrained environments, achieving roughly a 75% reduction. Huseynov et al. [45] described zero-copy serializers (such as FlatBuffers) that, in specific edge-gateway scenarios, outperform Protocol Buffers by lowering processing latency.

These methods are highly tuned to particular use cases (time series, ML, edge computing); by contrast, our evaluation focuses on general-purpose protocols that serve the standard structured telemetry of maritime containers. Implementing the specialized approaches would require substantial architectural changes and address problems different from our principal constraint (the 160-byte message-size limit of the LEO satellite network).

3.2. Maritime IoT and Container Monitoring

Recent full-scale logistics deployments highlight JSON's inefficiency in maritime applications. Falcitelli et al. [3] demonstrated through the 5GT System project that container monitoring platforms generate verbose JSON sensor streams, imposing significant transmission overhead during multimodal transport, particularly during ocean transit, where only satellite connectivity is available. Their previous work [5] on multi-radio devices for dry container tracking and monitoring provides concrete evidence of this challenge. The Container Tracking Device (CTD) developed for the 5GT System transmits UDP packets with JSON payloads to a oneM2M-compliant cloud platform, with typical telemetry messages structured as follows:

```
{
  "containerID": "LMCU1231237",
  "timestamp": "2023-04-20T14:30:45Z",
  "location": {
    "latitude": 31.8682,
    "longitude": 28.7412,
    "altitude": 49.50
  },
  "sensors": {
    "temperature": 17.00,
    "humidity": 71.00,
    "pressure": 1012.40,
    "acceleration": {
      "x": -993.93,
      "y": -27.10,
      "z": -52.00
    }
  },
  "network": {
    "type": "NB-IoT",
    "rssi": 28,
    "cellID": "999-01-1-31D41"
  },
  "battery": {
    "stateOfCharge": 96,
    "voltage": 3.7
  },
  "doorStatus": "closed"
}
```

These findings confirm that JSON-based telemetry, while standardized within oneM2M frameworks, requires aggressive compression strategies to meet real-world bandwidth and power constraints in maritime scenarios (reinforcing the consensus that text-based formats are untenable in bandwidth-constrained or satellite-dependent logistics contexts [3,5]).

The challenge extends beyond academic implementations to industry-wide standardization efforts. The Digital Container Shipping Association (DCSA), a global standards body for the container shipping industry, has published comprehensive data models for Track and Trace and IoT Commercial Events [49,50]. The DCSA Information Model 2023.Q1 defines standardized data structures for container telemetry, equipment events, transport events, and shipment tracking across maritime supply chains. However, examination of the DCSA specifications reveals that all data models are defined using verbose, text-based JSON schemas with extensive key–value pairs. For instance, the IoT Event entity within the Information Model includes attributes such as `equipmentReference`, `eventDateTime`, `eventClassifierCode`, `facilityTypeCode`, and nested location objects (each requiring full string-based key transmission in standard JSON encoding [50]). The DCSA Track and Trace API documentation similarly specifies JSON as the primary data interchange format for all shipment, equipment, and transport events communicated between carriers, shippers, and supply chain participants [49].

This standardization on verbose JSON formats across the global container shipping industry (encompassing both oneM2M-based IoT platforms and DCSA-compliant commercial systems) creates a fundamental tension between interoperability requirements and the bandwidth constraints of hybrid TN/NTN deployments. The proliferation of text-based telemetry in maritime logistics validates the choice of container tracking and monitoring as the primary motivating use case for this research, as it represents the most demanding scenario combining global coverage requirements, extended battery life needs, strict NTN message-size constraints, and industry-wide pressure to maintain standards compliance while achieving operational efficiency.

3.3. Architectural Optimization for Hybrid IoT Networks

Complementary to data-level optimization, architectural approaches address platform scalability and network heterogeneity in IoT systems. Recent research emphasizes Software-Defined Networking (SDN) and edge computing as key enabling technologies for hybrid terrestrial-satellite architectures. Esmat et al. [51] presented resilient network slicing strategies for satellite-terrestrial edge computing IoT networks, demonstrating that SDN-based orchestration can provide service continuity across domain boundaries (a critical requirement for maritime container tracking and monitoring, where devices transition between cellular and satellite connectivity). Their multi-domain approach to failure detection and mitigation directly addresses the reliability concerns inherent in hybrid TN/NTN deployments.

The integration of edge computing with satellite networks further enhances system performance for distributed IoT applications. Studies on satellite edge computing architectures [52,53] show that SDN/NFV frameworks enable intelligent task offloading and resource allocation across heterogeneous network segments, reducing inter-transmission delays while optimizing computational resource utilization. These architectural innovations are particularly relevant for massive-scale maritime deployments where thousands of containers must simultaneously upload telemetry during brief satellite visibility windows. A comprehensive systematic review [54] demonstrates that integrating SDN, IoT, and edge computing into unified platforms provides centralized management of heterogeneous devices while supporting real-time data analysis requirements (capabilities that complement the payload optimization strategies investigated in this work).

Within this ecosystem of architectural optimization approaches, the oneM2M standard emerges as a particularly strategic choice for hybrid TN/NTN maritime IoT systems [4,22]. The oneM2M partnership's continuous evolution demonstrates its commitment to addressing emerging requirements. With Release 4 recently published and Release 5 currently in

active development, the standard is incorporating advanced capabilities, including AI enablement for IoT, enhanced semantic discovery and interoperability, improved data privacy mechanisms, and sustainability-focused features [55,56]. These ongoing enhancements position oneM2M as a future-proof framework capable of supporting next-generation maritime logistics applications that will increasingly rely on AI-driven predictive analytics, cross-domain data sharing, and energy-efficient operations. The standard's proven scalability (demonstrated through successful deployments in smart cities, intelligent transportation systems, healthcare, and industrial IoT [57,58]) validates its suitability for the massive concurrent device transmissions characteristic of global container fleet management.

The synergy between architectural optimization (SDN-based orchestration, edge computing, oneM2M standardization) and data-level optimization (efficient binary serialization) provides a comprehensive pathway toward sustainable, scalable hybrid TN/NTN IoT deployments. While architectural approaches address platform bottlenecks and network heterogeneity, efficient serialization provides universal benefits for every transaction, regardless of access pattern (together forming the foundation for economically viable global maritime logistics systems).

3.4. Research Gap

The existing literature lacks comprehensive evaluations of device-friendly serialization techniques under hybrid TN/NTN constraints for maritime container tracking. This work addresses this gap by: (1) systematically comparing four protocols against dual feasibility criteria (implementation on battery-powered microcontrollers and compliance with the <160-byte commercial NTN message-size limit, as exemplified by Astrocast's operational platform); (2) validating the findings through large-scale load testing and integration with a real-world satellite network; and (3) providing a holistic cost–benefit analysis for fleet-scale maritime deployments. The literature-based device-side feasibility assessment, considering flash footprint, RAM usage, and CPU cycles, follows standard academic practice [38,44,45], since direct hardware-in-the-loop validation falls outside the project scope. This approach ensures alignment with typical battery-powered IoT microcontroller constraints while maintaining focus on server-side scalability contributions.

4. Methodology

The evaluation framework presented in this section establishes a generalizable methodology for assessing binary serialization protocols in hybrid terrestrial/non-terrestrial IoT deployments, applicable to any satellite platform with message-size constraints. While the fundamental approach—integrating device-side feasibility analysis, server-side scalability testing, and real-world operational validation—is platform-agnostic, this research applies the methodology using Astrocast's 160-byte payload limit as the reference constraint.

This choice serves three strategic purposes. First, Astrocast's operational commercial platform provides well-documented, publicly accessible specifications that enable reproducible research and facilitate practical validation—unlike experimental or developmental constellations, where constraints may evolve during deployment. Second, the 160-byte limit represents a realistic and demanding threshold characteristic of compact LEO IoT architectures optimized for spectral efficiency and power consumption, making it a representative benchmark for evaluating aggressive payload optimization strategies. Third, leveraging an operational service with established API access allows end-to-end validation in genuine satellite infrastructure (Section 5.7), demonstrating not only theoretical compatibility but also practical deployment feasibility.

The methodology's design explicitly supports adaptation to different NTN platforms: system architects can substitute alternative message-size thresholds (see Table 1) with-

out modifying the core evaluation framework. The multi-dimensional assessment approach—payload efficiency analysis, horizontal scalability testing, Figure of Merit calculation, and cost–benefit modeling—remains valid regardless of the specific satellite platform or constraint value, providing a reusable template for protocol selection in diverse hybrid TN/NTN scenarios.

By grounding the general methodology in Astrocast's concrete operational constraints, this research bridges the gap between abstract protocol comparisons and real-world deployment requirements, offering both theoretical insights and immediately actionable guidance for maritime container tracking and similar global IoT applications.

4.1. Experimental Design Overview

The evaluation framework implements a client–server architecture simulating the end-to-end data flow from remote IoT devices to a centralized processing platform. The design prioritizes realism, scalability, and reproducibility through containerization and actual network transmission between distinct physical machines. Critically, this architecture is built upon the oneM2M global standard framework (selected specifically for its proven capability to collect and distribute massive volumes of IoT data from any network infrastructure (terrestrial cellular, Wi-Fi, satellite NTNs) using any application protocol (HTTP, CoAP, MQTT, WebSocket)). This network-agnostic and protocol-flexible design makes oneM2M uniquely suited for hybrid TN/NTN maritime scenarios where thousands of container tracking and monitoring devices must simultaneously transmit telemetry through heterogeneous connectivity channels [4].

4.2. System Architecture

4.2.1. Conceptual Four-Layer Model

The system is modeled as a four-layer architecture (see Figure 1):

1. Device Layer: Thousands of battery-powered IoT devices (MCU-class with constrained resources as detailed in Section 2.4) that generate sensor data and apply payload optimization using candidate protocols.
2. Network Layer: Dual-path transmission capability selecting either NTNs (commercial satellite service) or TNs (cellular) based on availability during container transit.
3. Server Layer: A cluster of virtual machines hosting a custom decoding and decompression service alongside a oneM2M-compliant Common Services Entity (CSE), namely Mobius, for standard data management.
4. Application Layer: Downstream applications that consume telemetry data via the Mobius API.

Mobius serves as the oneM2M service-layer backend, where decoded telemetry is stored using standard oneM2M resources, such as contentInstance objects.

4.2.2. Experimental Testbed

To empirically test the performance of the candidate protocols within the conceptual architecture, a four-layer practical testbed was implemented (Figure 2). This setup simulates the end-to-end data flow from massive IoT device fleets through hybrid TN/NTN connectivity to a standards-compliant oneM2M backend, while maintaining realistic network separation between client and server domains.

Layer 1: Client Domain-Massive Device Fleet Simulation. The client-side implementation utilizes Locust [59], an open-source Python-based distributed load testing framework designed for evaluating system performance under massive concurrent device loads. Originally developed for web application stress testing, Locust's event-driven architecture and lightweight worker processes make it particularly well suited for simulating thousands of

independent IoT devices transmitting simultaneously. In this testbed, Locust executes three sequential operations per simulated device: (1) generation of baseline JSON telemetry payloads conforming to the maritime container data model (Section 3.2), (2) protocol-specific encoding using the appropriate serialization library (CBOR via `cbor2`, MessagePack via `msgpack`, Struct+Zlib via `struct/zlib`, Protocol Buffers via generated ProtoBuf classes), and (3) HTTP POST transmission of the resulting binary payload to the server endpoint. A critical methodological control implemented in the Locust client is data pre-generation: thousands of encoded payloads for each protocol are computed and cached in memory before load testing begins, ensuring that measured server-side performance reflects genuine decoding capacity rather than client-side encoding overhead. This approach enables scalable simulation of 100–2000 concurrent devices on standard development hardware while maintaining measurement validity.

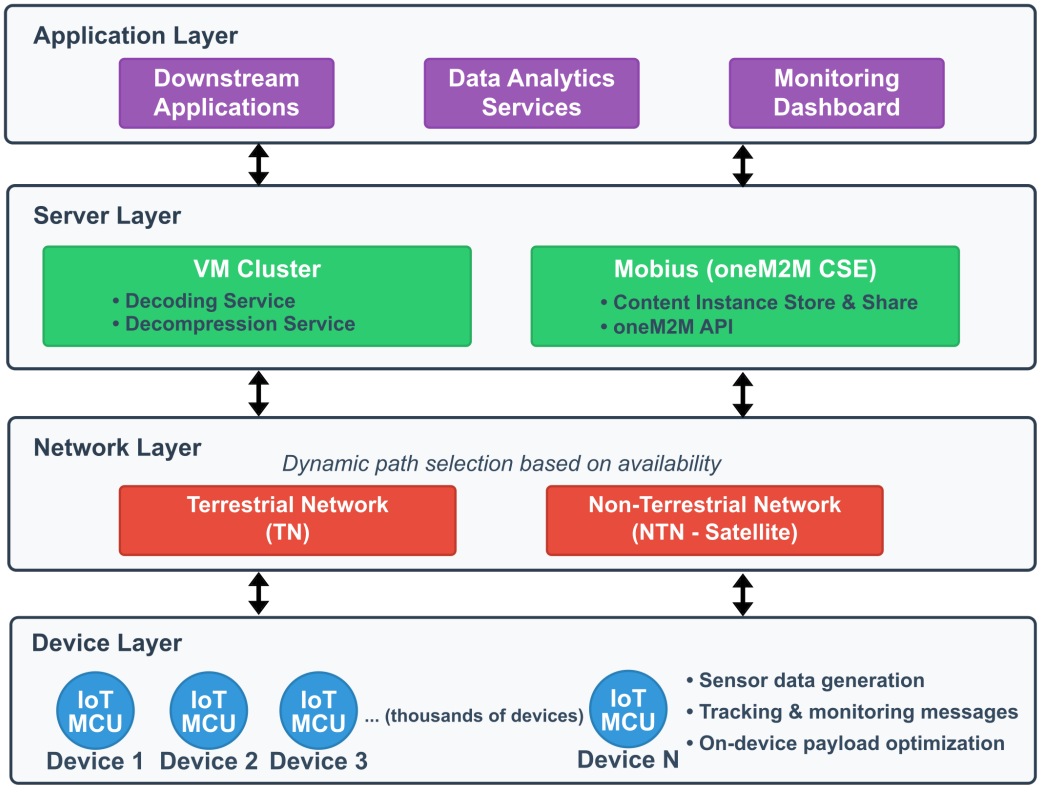

Figure 1. The 4-layer conceptual system architecture.

Layer 2: Network Link-Satellite Connectivity Emulation. The network layer deliberately employs terrestrial IP connectivity rather than actual satellite links, leveraging the Astrocast Astronode S DevKit's Wi-Fi board for protocol development and validation. As documented in the orange rationale box (Figure 2), this emulation strategy serves three purposes: (1) isolation of payload optimization performance from satellite-specific physical layer effects (propagation delay, Doppler shifts, link intermittency), enabling controlled, repeatable measurements focused on serialization efficiency (the core research objective); (2) enablement of high-frequency, iterative testing cycles impossible with actual satellite pass windows (typically 2–8 min per LEO satellite visibility period); and (3) alignment with Astrocast's standard development workflow, where device firmware undergoes Wi-Fi-based validation before operational satellite deployment. The Wi-Fi board emulates the complete Astrocast backend API interaction, allowing the system to process payloads "as if they had been sent via satellites" [10]. This methodology is validated through subsequent real satellite transmission experiments (Section 5.7), confirming that serialization perfor-

mance characteristics observed over terrestrial links translate directly to operational LEO satellite scenarios.

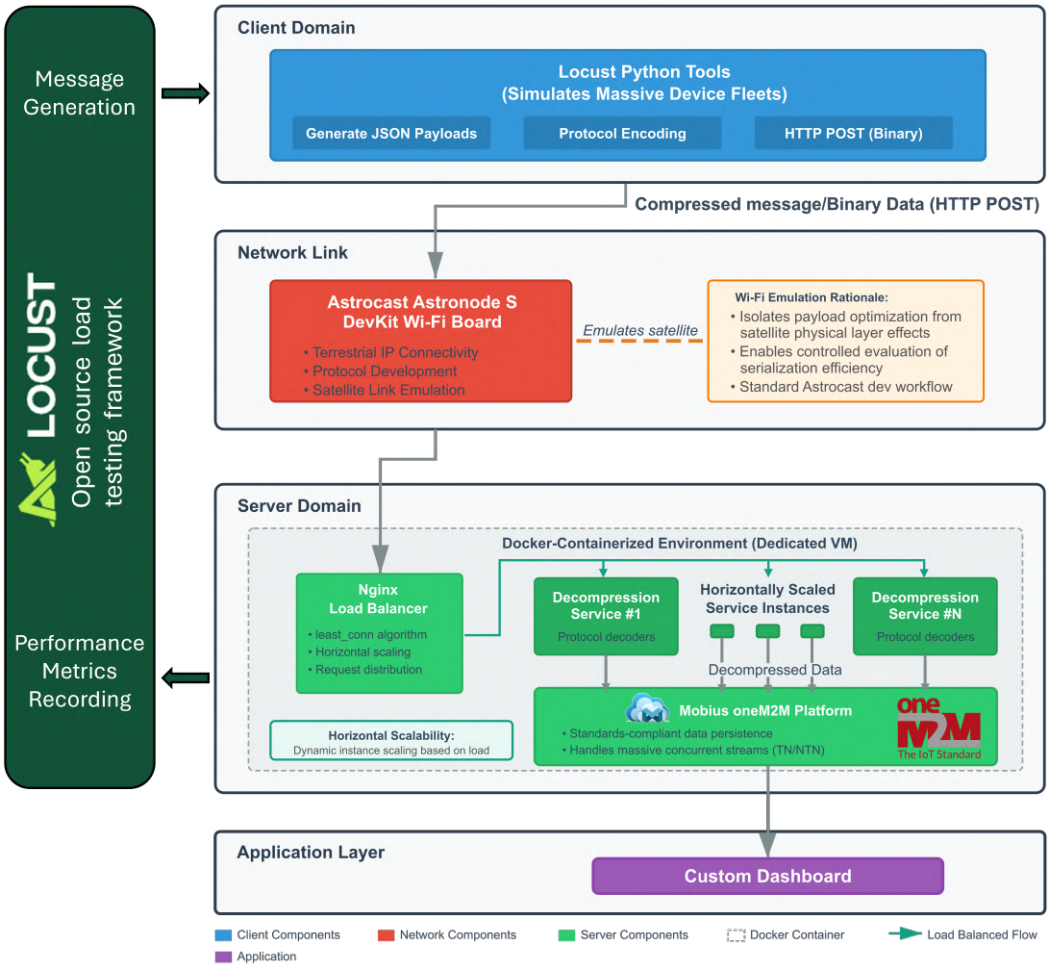

Figure 2. Experimental testbed architecture implementing four-layer model: (1) client domain with Locust load generation framework simulating massive device fleets, (2) network link leveraging Astrocast DevKit Wi-Fi emulation for controlled satellite link evaluation, (3) server domain with Docker-containerized Nginx load balancer, horizontally scaled decompression services, and Mobius oneM2M platform, and (4) application layer with custom visualization dashboard for end-to-end validation.

Layer 3: Server Domain-Containerized Horizontally Scalable Architecture. The server-side implementation is fully containerized using Docker Compose, enabling reproducible deployment, and dynamic horizontal scaling. The architecture comprises four components: (1) Nginx load balancer: Serves as the ingress point, distributing incoming HTTP POST requests across decompression service instances using the `least_conn` algorithm to minimize per instance queue depth. The load balancer configuration includes connection pooling, request buffering, and health check endpoints to ensure traffic is routed only to responsive instances. (2) Decompression service instances (1–4 replicas):Node.js applications implementing protocol-specific decoders (Section 4.4.2). Each instance operates independently with no shared state, enabling true horizontal scalability. Upon receiving a binary payload, the service identifies the protocol via HTTP headers, invokes the appropriate decoder, validates the reconstructed JSON against the expected schema, and forwards the payload to the Mobius platform. An asynchronous queue architecture decouples request reception (immediate HTTP 200 acknowledgment) from processing (decompression, forwarding), maintaining low response times under load. (3) Mobius oneM2M platform: An open-source

Common Services Entity (CSE) implementation providing standards-compliant resource management, subscriptions, and data persistence. Mobius receives decompressed JSON payloads wrapped in oneM2M `<contentInstance>` structures and stores them in hierarchical `<container>` resources, demonstrating practical integration of optimized serialization within standardized IoT middleware. (4) SQLite buffer:A lightweight file-based database colocated with the decompression service, providing resilient local persistence before Mobius forwarding. This architectural pattern ensures zero data loss even during Mobius unavailability, acting as a safeguard for production deployments.

The containerized architecture's horizontal scalability is a key experimental variable: tests execute with 1, 2, and 4 decompression service replicas to empirically validate linear scaling characteristics and identify performance ceilings. Resource utilization metrics (CPU, memory) are monitored via standard Docker tools to confirm efficient multi-instance operation.

Layer 4: Application Layer-End-to-End Validation Dashboard. A custom web-based visualization dashboard (HTML/CSS/JavaScript) served by the Node.js application provides real-time visibility into the complete data pipeline. The dashboard polls a `/get-latest-data` REST endpoint that queries the SQLite buffer, displaying: decoded telemetry fields (container ID, location, sensor readings, network status), aggregate statistics (total records processed, unique containers tracked, success rate, average compression ratio), and system health indicators (queue backpressure, forwarding failures). This application layer serves two purposes: (1) qualitative validation that optimized serialization preserves data correctness and usability throughout the end-to-end system, and (2) demonstration of the practical utility of the decompressed data for maritime logistics operators monitoring container fleet status.

Measurement Isolation and Validity. The two-machine deployment (client on local workstation, server on dedicated VM) ensures network transmission occurs over genuine IP infrastructure, capturing realistic latency and bandwidth characteristics. The physical separation eliminates localhost optimizations that would invalidate performance measurements. All reported metrics (throughput, latency, failure rate) are captured exclusively from the server side using Locust's distributed measurement framework, which aggregates statistics from client workers and provides percentile-accurate latency distributions. Each experimental configuration (protocol type, device count, server replica count) is executed for a sufficient duration to reach steady-state performance, with results averaged across multiple trials to ensure statistical validity and reproducibility.

4.3. Baseline Data Payload

The standardized test payload represents realistic maritime container telemetry with 20 fields spanning multiple data types essential for multimodal logistics tracking: SIM identifier (MSISDN), international container ID (ISO6346 [60]), UTC timestamp, cellular metrics (RSSI, CGI), battery state-of-charge, 3-axis accelerometer readings for shock detection, environmental sensors (temperature, humidity, pressure) for cargo condition monitoring, door status for security, and comprehensive GNSS data (latitude, longitude, altitude, speed, heading, satellite count, HDOP) for position tracking during ocean transit. The uncompressed JSON representation measures 418 bytes, serving as the definitive baseline for compression ratio calculations. This structure directly reflects industrial applications in container tracking and monitoring deployed in the 5GT System project and subsequent maritime IoT implementations [3,5], ensuring relevance to the primary target domain of global supply chain visibility.

4.4. Protocol Implementations

4.4.1. Client-Side Encoding

- CBOR: Python `cbor2` library [18] with `cbor2.dumps()` for dictionary-to-binary conversion. Embedded equivalent: `tinycbor` C library for resource-constrained microcontrollers;
- MessagePack: Python `msgpack` library [19] via `msgpack.packb()`. Embedded equivalent: `mpack` library optimized for battery-powered devices;
- Struct+Zlib: Two-stage process using Python `struct.pack()` for big-endian binary serialization followed by `zlib.compress()` at maximum level (9). Embedded equivalent: Miniz library [20] (<20 KB RAM footprint suitable for microcontroller deployment);
- Protocol Buffers: Schema-driven approach with `.proto` file compiled via `protoc` to generate Python classes. Serialization via `.SerializeToString()`. Embedded equivalent: nanopb library [38] (30–40 KB flash, 2–4 KB RAM, optimized for constrained IoT devices).

4.4.2. Server-Side Decoding

The Node.js decompression service implements protocol-specific decoders:

- CBOR: `cbor` package with `cbor.decode()`;
- MessagePack: `@msgpack/msgpack` package with `decode()`;
- Struct+Zlib: Native `zlib.inflateSync()` followed by custom `Buffer.read*()` parsing with offset management;
- Protocol Buffers: `protobufjs` library loading `.proto` schema at startup, decoding via `Message.decode()`.

Key architectural features include: (1) asynchronous queue processing decoupling reception from decoding to maintain low response times, (2) SQLite buffer for durable persistence before Mobius forwarding, and (3) oneM2M-compliant payload wrapping in `{“m2m:cin”: {“con”: {...}}}` structure.

4.5. Evaluation Metrics

4.5.1. Device-Side Feasibility

Direct on-device validation across a variety of battery-powered IoT microcontrollers is outside the scope of the present study. Instead, device-side applicability was assessed by (1) surveying authoritative measurements and implementation reports in the literature [38,44,45] and (2) verifying the availability of algorithm implementations in the C language for MCU classes typical of container tracking and monitoring devices (for example, embedded libraries such as nanopb, tinycbor/mpack and Miniz/minizip have known ports and usage notes for constrained toolchains). This approach follows standard academic practice when hardware-in-the-loop testing is not feasible within project constraints.

From a theoretical device-side perspective, and beyond the primary message-side compression ratio, a complete evaluation of on-device performance should consider the following metrics:

- Serialization speed: How quickly the payload can be encoded/decoded on the target MCU (affecting end-to-end latency);
- CPU usage: Processor load required for serialization (directly related to energy spent in active CPU cycles);
- Memory footprint: Flash and runtime RAM consumed by the serialization library and generated code (critical to avoid overload on severely constrained systems);
- Overall energy consumption: The net energy cost of encoding plus radio transmission (the dominant factor for battery life).

In the absence of direct measurements on IoT MCUs, device performance characteristics (serialization speed, CPU cycles, RAM usage, and estimated energy cost) can be extrapolated based on the cited literature in order to align conclusions with the realistic constraints of embedded systems, keeping this study focused on end-to-end payload feasibility and server-side scalability.

4.5.2. Server-Side Performance

- Throughput (RPS): Total successful requests per second (s^{-1}), primary scalability indicator;
- Latency (ms): Response time distribution (minimum, median, average, 95th percentile, maximum);
- Stability (Failure Rate %): Percentage of failed requests under load, identifying performance ceilings.

4.5.3. Figure of Merit

We define a dimensionless holistic Figure of Merit that rewards high throughput and reliability while penalizing latency:

$$\text{FoM} = \frac{\text{RPS } (s^{-1}) \times \text{Success Rate} \times [\text{Reference Latency (s)}]^2}{\text{95th Percentile Latency (s)}}, \quad (1)$$

where Reference Latency is set to 1 s as the normalization factor. This formulation yields a dimensionless index. Higher values indicate superior balance of speed, reliability, and stability. The metric can be interpreted as the ratio of actual system capacity to latency-normalized capacity, with the success rate weighting ensuring that unreliable systems are appropriately penalized.

4.6. Experimental Procedure

Load tests are executed progressively, increasing the concurrent IoT device counts (100, 300, 500, 1000, 2000 devices transmitting simultaneously) against server instances with varying replica counts (1, 2, 4 instances). This scaling pattern simulates realistic maritime deployment scenarios where large container fleets transmit periodic telemetry updates in coordinated batches (for example, when 2000 containers simultaneously report their status as vessels enter or exit satellite coverage zones during ocean transit). Pre-generation of thousands of encoded payloads before testing ensures measurements reflect server capacity rather than client encoding overhead. Each test run lasted a sufficient duration to reach steady-state performance, with results averaged across multiple trials to ensure statistical validity.

The experimental testbed employs terrestrial IP connectivity as described in Section 4.2.2 (Layer 2), isolating serialization performance from satellite-specific physical layer variability. The critical NTN constraint (the 160-byte message-size limit) is applied as an analytical filter in results evaluation. Subsequent validation experiments using actual Astrocast satellite transmission (Section 5.7) confirm operational compatibility.

5. Results

5.1. Message-Size Analysis

Table 2 presents the fundamental efficiency metric for each protocol. The results provide a useful filter for NTN viability.

Schema-less binary formats (MessagePack, CBOR) achieve valuable 28.0% reductions suitable for terrestrial networks, but their ∼300-byte payloads are 88% larger than the NTN constraint—conclusively disqualifying them for hybrid deployments requiring

satellite connectivity during ocean transit. Only Struct+Zlib (127 bytes, 69.2% reduction) and Protocol Buffers (138 bytes, 66.7% reduction) meet the <160-byte requirement characteristic of operational commercial LEO IoT platforms such as Astrocast. This empirical result validates the theoretical principle that eliminating key-based overhead is essential for achieving NTN-compatible compression ratios in maritime container tracking and monitoring applications.

Table 2. Comparative analysis of encoded message sizes.

Protocol	Original (bytes)	Encoded (bytes)	Reduction (%)	NTN Viable (<160 B)
JSON (Baseline)	418	418	0.0	No
MessagePack	418	301	28.0	No
CBOR	418	298	28.4	No
Struct+Zlib	418	127	69.2	Yes
Protocol Buffers	418	138	66.7	Yes

5.2. Single Server Performance

Table 3 presents averaged performance metrics for all four protocols under progressively increasing loads with a single server instance, simulating scenarios where growing numbers of IoT devices transmit data simultaneously.

Table 3. System performance with single server instance under simultaneous IoT device transmission (averaged results).

Protocol	# of Devices	RPS (s^{-1})	Avg. Resp. (ms)	95th Pctl (ms)	Failures (%)
CBOR	100	48	65	130	0
	300	143	62	145	0
	500	239	65	180	0
	1000	407	418	2100	0
	2000	435	2352	11,667	4
MessagePack	100	48	66	145	0
	300	143	59	120	0
	500	239	57	125	0
	1000	418	353	2033	0
	2000	478	2043	10,000	3
Struct+Zlib	100	48	67	115	0
	300	143	62	120	0
	500	239	60	125	0
	1000	418	359	2100	0
	2000	496	1853	8900	2
Protocol Buffers	100	48	72	140	0
	300	143	63	130	0
	500	239	60	125	0
	1000	425	321	1967	0
	2000	492	1863	9867	3

Performance remains stable up to 1000 concurrent IoT devices transmitting simultaneously across all protocols, achieving 407–425 RPS with sub-2.2-second 95th percentile latencies. However, at 2000 devices (representing a realistic scenario where a large maritime fleet experiences synchronized satellite visibility windows during ocean transit), all implementations exhibit catastrophic degradation: failure rates rise to 2.5–4%, average response times exceed 1850 ms, and 95th percentile latencies balloon to 8.9–11.7 s. HTTP 504 Gateway Timeout errors indicate the single Node.js instance reaches its process-

ing ceiling at approximately 400–500 RPS, empirically defining the baseline architecture's scalability limit.

Figure 3 illustrates the system throughput evolution as concurrent device load increases from 100 to 2000 devices transmitting simultaneously with a single server instance. The performance characteristics reveal three distinct operational regimes.

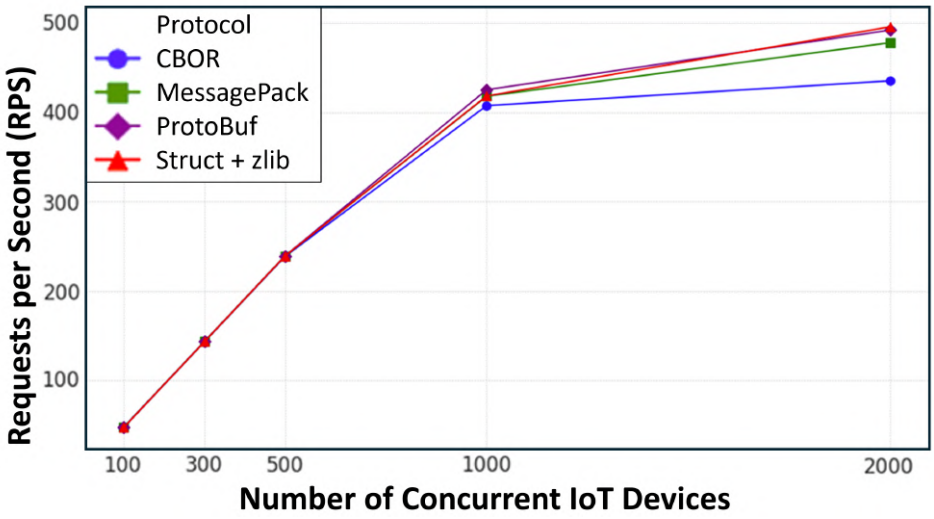

Figure 3. Throughput evolution as the number of transmitting devices increases with a single decoding server.

Linear Scaling Regime (100–500 devices): All four protocols exhibit nearly identical linear throughput growth, scaling from approximately 48 s^{-1} at 100 devices to 239 s^{-1} at 500 devices. The overlapping curves in this regime demonstrate that protocol choice has a negligible impact on system capacity when operating well below the server's processing ceiling. This behavior confirms that the primary bottleneck in low-to-moderate load scenarios is network I/O rather than CPU-bound deserialization overhead.

Saturation Transition (500–1000 devices): Beyond 500 devices, throughput growth becomes sublinear as the single Node.js instance approaches its event loop capacity limits. At 1000 concurrent devices, all protocols converge to 407–425 s^{-1}, representing approximately 85–90% of the architecture's sustainable maximum capacity. The maintained protocol parity in this regime indicates that even as the server enters a resource-constrained state, deserialization efficiency differences remain secondary to fundamental architectural limitations of single-instance deployment.

Performance Ceiling and Protocol Divergence (2000 devices): At 2000 devices, the system enters a stress regime where subtle protocol differences become observable. Protocol Buffers and Struct+Zlib achieve marginally higher throughput (492–496 s^{-1}) compared to CBOR and MessagePack (435–478 s^{-1}), representing a 5–14% performance advantage. However, this regime is characterized by system instability: all protocols experience failure rates of 2.5–4.0% (Table 3), with 95th percentile latencies exceeding 8.9 s due to HTTP 504 Gateway Timeout errors. The marginal throughput differences at this extreme load are less significant than the fundamental architectural conclusion: single-instance deployment cannot reliably sustain 2000-device concurrent transmission scenarios.

Architectural Implications: The near-perfect overlap of performance curves across vastly different serialization approaches (schema-less binary, compression-based, schema-driven) up to 1000 devices provides strong empirical evidence that network I/O, not CPU-bound deserialization, dominates system performance. This finding validates the message-size optimization strategy: reducing transmitted bytes directly improves throughput by minimizing I/O wait states, regardless of the modest CPU overhead differences

between protocols. The consistent performance ceiling at approximately 400–500 s^{-1} for single-instance deployment establishes the clear need for horizontal scaling architectures (Section 5.3) to achieve production-grade capacity for massive maritime IoT deployments. This finding underscores the importance of the oneM2M platform's design for horizontal scalability, as massive-scale maritime deployments require architectures capable of absorbing sudden traffic bursts when thousands of containers simultaneously transmit telemetry.

5.3. Horizontal Scalability Analysis

Based on message-size findings (Section 4.1), scalability testing focused exclusively on the two NTN-viable protocols: Struct+Zlib and Protocol Buffers.

5.3.1. Two Server Instances

Table 4 demonstrates the immediate stability gains from introducing a second server instance.

Table 4. Performance with two server instances under simultaneous IoT device load (averaged results).

Protocol	# of Devices	Instances	RPS (s^{-1})	Avg. (ms)	95th (ms)	Fail (%)
Struct+Zlib	500	2	239	60	130	0
	1000	2	478	59	150	0
Protocol Buffers	500	2	240	58	115	0
	1000	2	478	58	127	0

At 1000 devices, scaling to two instances doubles throughput to ∼478 RPS while dramatically stabilizing the system: 95th percentile latencies drop from >2000 ms (single instance) to 127–150 ms, demonstrating effective load distribution. This validates the oneM2M architecture's horizontal scalability design principle, which is essential for accommodating the variable traffic patterns characteristic of hybrid TN/NTN maritime deployments where device transmission synchronization depends on satellite coverage dynamics.

5.3.2. Four Server Instances

Table 5 presents the critical validation that horizontal scaling successfully eliminates failures under the 2000-devices stress test.

Table 5. Performance at 2000 concurrent IoT devices: single vs. four server instances.

Protocol	Instances	RPS (s^{-1})	Avg. (ms)	95th (ms)	Fail (%)
Struct+Zlib	1	496	1853	8900	2
	4	711	701	1124	0
Protocol Buffers	1	492	1863	9867	3
	4	712	706	1096	0

Four-instance deployment achieves: (1) 44–45% throughput increase to >711 RPS, (2) complete elimination of failures (0%), and (3) dramatic latency stabilization with 95th percentiles dropping 88–89% to ∼1.1 s. Figure 4 specifically shows the throughput comparison between single-instance and four-instance deployment at 2000 concurrent devices for NTN-viable protocols. These results conclusively demonstrate the containerized, load-balanced architecture's effectiveness for achieving production-scale capacity capable of handling massive simultaneous IoT device transmissions. The performance demonstrates

the architecture's ability to handle realistic maritime deployment scenarios where large container fleets transmit synchronized telemetry during satellite visibility windows.

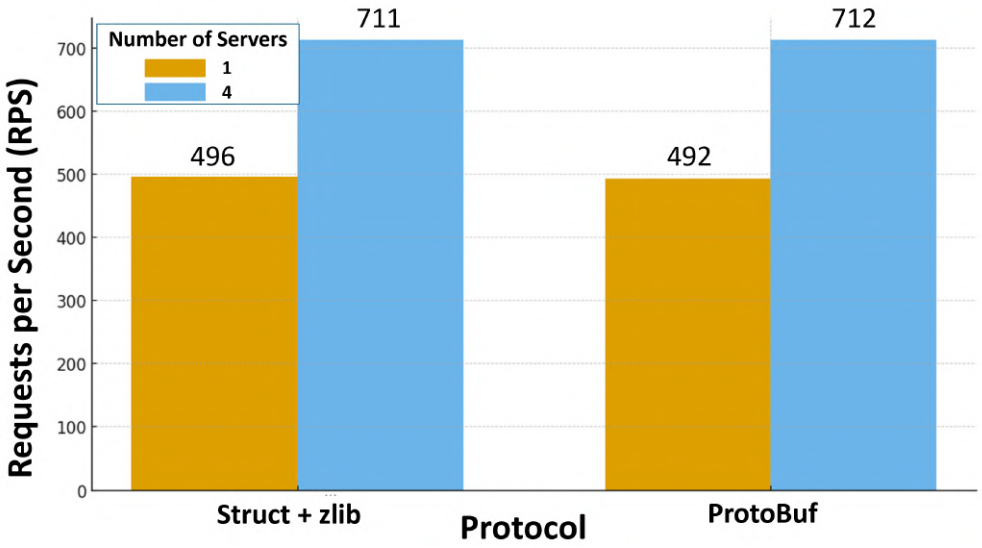

Figure 4. Horizontal scalability validation.

5.4. Resource Utilization

VM monitoring revealed stable resource consumption patterns across scaling instances. Under the 2000-devices, four-instance load representing maximum tested capacity, CPU utilization remained within acceptable bounds, and memory consumption showed no signs of exhaustion or leakage. This confirms the architecture's efficiency and indicates that further vertical scaling headroom exists before hardware constraints become limiting factors.

5.5. Figure of Merit Analysis

Table 6 synthesizes performance through the integrated FoM metric, revealing Protocol Buffers' consistent advantage across scaling server instances.

Table 6. Figure of Merit comparison across server instances.

Protocol	# of Devices	Inst.	RPS (s^{-1})	95th (s) (s)	Success	FoM
Single Instance Performance						
Struct+Zlib	500	1	239	0.125	100%	1914
	1000	1	418	2.100	100%	199
	2000	1	496	8.900	98%	54
Protocol Buffers	500	1	239	0.125	100%	1914
	1000	1	425	1.967	100%	216
	2000	1	492	9.867	97%	49
Scaled Instances						
Struct+Zlib	1000	2	478	0.150	100%	3188
	2000	4	712	1.124	100%	633
Protocol Buffers	1000	2	478	0.127	100%	3762
	2000	4	712	1.096	100%	650

At maximum tested capacity (2000 IoT devices transmitting simultaneously, four instances), Protocol Buffers achieves a dimensionless FoM of 650 versus Struct+Zlib's 633 (a 2.7% advantage reflecting slightly lower latency despite nearly identical throughput).

This quantitative result, combined with Protocol Buffers' superior maintainability through schema-driven design, solidifies its recommendation as the optimal solution for massive-scale maritime IoT deployments within the oneM2M framework.

5.6. End-to-End System Validation

A real-time data visualization dashboard consuming decoded telemetry from the SQLite buffer confirmed full pipeline integrity. Key metrics included: (1) 3943 total records processed, (2) 402 unique containers tracked simultaneously, (3) a 100% success rate for Mobius forwarding, (4) a 2.86× average compression ratio, and (5) zero queue backpressure under load. This qualitative validation demonstrates that optimized serialization preserves data correctness and usability throughout the end-to-end system (see Figure 5).

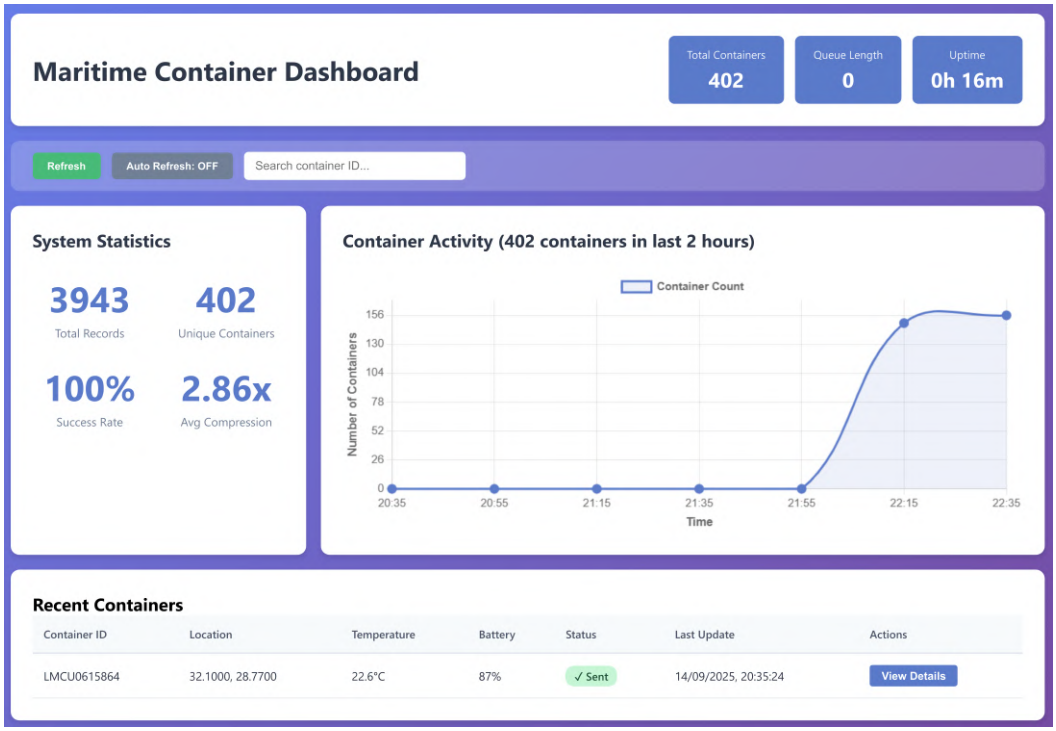

Figure 5. Screenshot of the Maritime Container Dashboard.

5.7. Astrocast Platform Validation

Live experiments using the Astrocast Astronode S DevKit (the development interface for the operational commercial LEO IoT constellation) confirmed end-to-end operational viability (see Figure 6). Binary payloads generated through the developed pipeline (137–139 bytes) were successfully: (1) transmitted via the DevKit firmware module, (2) delivered to the Astrocast portal with Text/HEX/Base64 representation, (3) forwarded to the VM decoder service, (4) deserialized to the original JSON format, and (5) ingested into the Mobius platform. This end-to-end validation across genuine commercial satellite IoT infrastructure demonstrates that the proposed optimization strategy is not merely theoretical but operationally compatible with deployed LEO constellations, providing global coverage for maritime container tracking and monitoring during ocean voyages. The consistent message sizes (137–139 bytes) across multiple test runs closely match the emulated experiment results (∼138 bytes for Protocol Buffers), validating the methodological approach and confirming that the optimization techniques meet real-world operational requirements of commercial satellite IoT services.

Figure 6. Setup of the live experiment (bottom left photo) using the Astronode S DevKit (upper right photo).

6. Discussion

6.1. Interpretation of Key Findings

6.1.1. Payload Efficiency and NTN Constraint Filtering

The message-size results (Table 2) provide definitive answers to the research question's first component. Schema-less formats (CBOR, MessagePack) deliver meaningful improvements for terrestrial deployments but fundamentally cannot satisfy NTN requirements for maritime container tracking and monitoring during ocean transit. The 28% reduction, while valuable, leaves payloads nearly double the limit imposed by operational commercial LEO IoT platforms such as Astrocast. This outcome directly validates the theoretical principle articulated in Section 2.3: for structured telemetry data typical of container monitoring applications, the primary verbosity source is the keys themselves, and viable NTN solutions must eliminate this overhead.

The profound 67–69% reductions achieved by Struct+Zlib (127 bytes) and Protocol Buffers (138 bytes) demonstrate that key elimination is not merely beneficial but essential for enabling continuous global asset tracking. This finding aligns with prior work showing protocol-based binary formats consistently outperform schema-less alternatives in space-constrained scenarios [37,61], while extending these conclusions to the specific context of commercial satellite IoT services with well-defined operational constraints.

6.1.2. Network I/O as Primary Bottleneck

The single server results (Table 3) reveal a critical architectural insight: for this data ingestion workload representing massive simultaneous IoT device transmissions, the primary performance bottleneck is network I/O, not CPU capacity. All protocols achieved nearly identical throughput (∼240 RPS at 500 devices, ∼420 RPS at 1000 devices) despite varying decoding complexity, demonstrating that the time savings from reduced network transmission significantly outweigh minor CPU overhead differences.

This represents bottleneck shifting: by reducing message size, servers spend less time in I/O-wait states per request, freeing the event loop to process subsequent concurrent requests more rapidly. The net effect is increased sustainable throughput (a data-level optimization that directly enhances server scalability). This finding is particularly significant for oneM2M-based platforms designed to aggregate data from heterogeneous network sources (TN/NTN), as it validates that payload optimization provides universal performance

benefits regardless of the underlying connectivity type. The observation extends prior research showing payload compactness impacts end-to-end latency more than serialization speed [44,45], while contextualizing it specifically for hybrid network architectures where the oneM2M abstraction layer must efficiently handle diverse data streams.

6.1.3. Horizontal Scalability Validation

The four-instance configuration results (Table 5) empirically validate containerized horizontal scaling as an effective strategy for eliminating single-point bottlenecks in massive IoT deployments. The 44.7% throughput increase and complete failure elimination at a 2000-device simultaneous transmission load demonstrate that the load-balanced architecture successfully distributes workload across instances. Resource utilization monitoring confirmed stable CPU/memory consumption, indicating that further vertical scaling potential exists before hardware limits are reached.

This validation is particularly significant for oneM2M-based platforms serving hybrid TN/NTN maritime scenarios, where traffic patterns exhibit high variability due to satellite coverage dynamics. The ability to horizontally scale the CSE layer while maintaining the oneM2M standard's protocol flexibility (HTTP, CoAP, MQTT) and network abstraction ensures that platforms can accommodate sudden surges in concurrent device transmissions (such as when entire fleets of containers simultaneously acquire satellite connectivity after extended periods of ocean transit). The results confirm that oneM2M's architectural design, combined with appropriate payload optimization, provides a robust foundation for global-scale IoT data collection and distribution.

6.2. Protocol Selection: Architectural Considerations

While both NTN-viable protocols demonstrated excellent performance, fundamental architectural differences inform the final recommendation. Struct+Zlib represents peak manual optimization, achieving 8.7% smaller payloads (127 vs. 138 bytes) through bespoke binary formatting. However, this comes at a substantial cost: the rigid, code-defined format requires manual, error-prone updates to packing/unpacking logic on both the server and every device firmware whenever data structures evolve. This brittleness presents significant long-term maintenance challenges and operational risk.

Protocol Buffers, conversely, employs a schema-driven approach where data structures are defined in centralized `.proto` files serving as formal, unambiguous contracts. Changes require only schema updates and automatic code regeneration via the ProtoBuf compiler (dramatically reducing human error potential and accelerating evolution cycles). The FoM analysis (Table 6) quantitatively confirms Protocol Buffers' superior balance (649.5 vs. 633.3), primarily through slightly lower latency (1.096 vs. 1.124 s 95th percentile) despite nearly identical throughput.

This combination of marginally better performance and substantially superior maintainability makes Protocol Buffers the definitive choice for production-grade, large-scale IoT deployments requiring long-term reliability and extensibility.

6.3. Economic Impact Analysis

To quantify economic viability for maritime logistics applications, we model a 10,000-device fleet tracking and monitoring dry containers across global supply chains with staggered reporting every 6 h (four messages/day, 120 messages/month per device). Table 7 presents fleet-scale data volumes and illustrative costs based on operational commercial satellite IoT pricing and reasonable, though not rock-bottom, pricing for TN NB-IoT data.

Table 7. Monthly fleet transmission costs (10,000 devices, 4 messages/day).

Protocol	Fleet/Month (MB)	TN Cost @ €0.02/MB	Fleet/Month (KB)	NTN Cost @ €0.183/KB
JSON	478.4	€9.57	489,882	€89,648
Protocol Buffers	157.9	€3.16	161,690	€29,589
Struct+Zlib	145.3	€2.91	148,787	€27,228

Assumptions: NTN rate €0.183/KB (commercial LEO IoT platform class [62]), likely TN rate.

The results demonstrate: (1) JSON is infeasible for NTNs (exceeds 160-byte limit and incurs ∼k€89/month), (2) optimized protocols achieve ∼k€62 monthly savings versus JSON in the NTN context for global container tracking, and (3) even in lower-cost TN environments, optimization reduces bandwidth consumption by ∼320 MB/month while extending battery life through reduced airtime (a critical factor for devices deployed in sealed containers with multi-year battery design life). These figures validate that payload optimization delivers both a technical necessity for NTN feasibility during ocean transit and substantial economic benefits at fleet scale for maritime logistics operators.

6.4. Practical Deployment Recommendations

Based on empirical findings, we provide evidence-based guidance for system architects:

For Terrestrial-Only Deployments: Schema-less binary formats (CBOR, MessagePack) are recommended. Their 28% message-size reduction provides meaningful cost savings and performance improvements in high-bandwidth environments while minimizing implementation complexity through schema-free operation. CBOR's IETF standardization [18] and MessagePack's widespread adoption [19] ensure robust ecosystem support.

For Hybrid TN/NTN Deployments: Protocol Buffers is strongly recommended as the superior choice for maritime container tracking and monitoring and similar applications requiring global coverage. While Struct+Zlib achieves marginally smaller payloads (127 vs. 138 bytes), Protocol Buffers' schema-driven design provides: (1) type safety and data integrity through formal contracts, (2) graceful evolution via forward/backward compatibility as telemetry requirements change, (3) reduced integration error risk through automatic code generation for diverse device firmware and server platforms, and (4) slightly better FoM performance (649.5 vs. 633.3). The economic analysis further validates this recommendation: 67% message-size reduction translates to ∼k€62 monthly savings for 10,000-device fleets, making Protocol Buffers both technically optimal and economically sustainable for commercial maritime logistics deployments.

6.5. Limitations and Future Work

Three primary limitations warrant acknowledgment. First, we infer device-side performance characteristics (serialization speed, RAM usage, CPU cycles, energy consumption) from the authoritative literature [38,44,45], rather than evaluating them through direct hardware-in-the-loop (HIL) measurements on battery-powered microcontrollers. This approach follows standard academic practice when HIL validation is outside the project scope, ensuring alignment with embedded constraints while maintaining focus on server-side scalability contributions. However, future work should conduct rigorous on-device profiling using cycle-accurate tools and power analyzers to definitively validate the encoding latency, memory footprint, and energy consumption characteristics discussed in Section 2.4.

Second, while the testbed employed terrestrial IP networks validated by the Astrocast DevKit methodology [10], full operational constellation validation remains pending. Future experiments should deploy satellite modules on actual container ships under open-sky conditions to confirm robustness under authentic NTN channel characteristics (latency

variability, Doppler shifts, intermittent coverage during vessel movement) aligned with 3GPP Release 17 specifications [12,13] and operational commercial LEO IoT platform performance. The preliminary validation in Section 5.6 using actual Astrocast satellite transmission provides initial confidence in real-world compatibility, but comprehensive maritime deployment testing remains essential for production system validation.

Third, payload structure generalizability requires consideration. The flat, 20-field telemetry payload reflects maritime container monitoring [3,5] but may not represent all IoT scenarios. Deeply nested JSON hierarchies (e.g., healthcare records, automotive diagnostics) could amplify JSON's inefficiency and alter relative protocol performance. However, the fundamental conclusion (that key elimination is essential for NTN viability in commercial satellite IoT platforms) remains valid across domains that require global coverage for extended periods without terrestrial connectivity.

Future research directions include: (1) integration of payload optimization with architectural caching strategies for synergistic performance gains, (2) hybrid transport protocols combining MQTT for local sensor clusters with optimized backhaul serialization, (3) exploration of zero-copy serializers (FlatBuffers) for edge gateway scenarios [45], and (4) application-specific compression schemes (delta encoding for time series [47]) for specialized workloads.

7. Conclusions

This paper addressed the critical challenge of data transmission efficiency in hybrid terrestrial/non-terrestrial IoT networks through systematic evaluation of four binary serialization protocols, with specific focus on maritime container tracking and monitoring during ocean voyages. The research conclusively demonstrates that standard JSON encoding is fundamentally incompatible with operational commercial satellite IoT constraints, exceeding the 160-byte message-size limit characteristic of LEO platforms such as Astrocast by 161% while imposing unsustainable economic overhead.

The key findings include: (1) only Protocol Buffers (138 bytes, 66.7% reduction) and Struct+Zlib (127 bytes, 69.2% reduction) achieve NTN-viable compression for maritime applications, disqualifying schema-less formats (CBOR, MessagePack) despite their 28% improvements; (2) horizontal scaling successfully eliminates single-point bottlenecks, with four-instance deployments sustaining 2000 concurrent IoT devices transmitting simultaneously at 712 RPS with zero failures; (3) dimensionless Figure of Merit analysis reveals Protocol Buffers' superior balance (FoM: 650 vs. 633) combining performance and maintainability; (4) live validation using the Astrocast platform (the most mature operational commercial LEO IoT service) confirms end-to-end compatibility across genuine satellite infrastructure; and (5) fleet-scale cost modeling demonstrates potential monthly savings exceeding €62,000 for 10,000-device container tracking and monitoring deployments. The successful integration with the oneM2M Mobius platform validates the standard's capability to efficiently aggregate and distribute massive volumes of optimized IoT data from heterogeneous TN/NTN networks using flexible application protocols.

While this study employed Astrocast's 160-byte constraint as the reference threshold, the methodology and findings generalize across the diverse landscape of commercial satellite IoT platforms. For ultra-constrained systems like Kinéis (30 bytes) [34,36], the validated Protocol Buffers (138 bytes) and Struct+Zlib (127 bytes) solutions would require payload fragmentation or field reduction, but the systematic evaluation framework remains directly applicable for assessing optimization strategies within stricter limits. For moderate-capacity platforms like ORBCOMM (6–250 bytes) [31], Protocol Buffers and Struct+Zlib satisfy payload constraints while schema-less formats (CBOR: 298 bytes, MessagePack: 301 bytes) approach feasibility limits. For higher-capacity services like Iridium SBD (340 bytes) [35],

all evaluated protocols satisfy constraints, with Protocol Buffers recommended based on its superior Figure of Merit (649.5 vs. 633.3) and maintainability advantages regardless of technical feasibility margins. The methodology's multi-dimensional framework—payload efficiency analysis, horizontal scalability testing, operational validation, and economic modeling—provides a reusable template applicable to any NTN message-size constraint, enabling system architects to systematically select optimal serialization strategies tailored to their specific platform requirements.

The research establishes Protocol Buffers as the optimal solution for production-grade hybrid TN/NTN systems. Its schema-driven architecture provides not only the compression efficiency necessary for satellite links but also the type safety, forward/backward compatibility, and automatic code generation essential for long-term maintainability in evolving IoT ecosystems. The integration with oneM2M's global standard framework (specifically designed to collect and distribute massive IoT data volumes from any network using any protocol) demonstrates that optimized serialization strategies can be seamlessly deployed within standards-compliant platforms that serve heterogeneous TN/NTN maritime deployments.

These findings directly apply to maritime logistics, environmental monitoring, and asset tracking and monitoring scenarios requiring global coverage, including ocean transit. This work demonstrates that careful data-level optimization is not merely a performance enhancement but a fundamental enabler of hybrid network deployments. Without such optimization, commercial satellite IoT remains economically prohibitive for large-scale container fleets; with appropriate serialization strategies, it becomes both technically feasible and commercially sustainable.

For terrestrial-only deployments where the NTN constraint does not apply, schemaless formats (CBOR, MessagePack) remain excellent choices, offering 28% message-size reduction with minimal implementation complexity. However, as IoT deployments increasingly demand global coverage spanning heterogeneous network technologies (particularly in maritime supply chain visibility applications), schema-driven binary serialization emerges as the cornerstone of scalable, sustainable M2M architectures capable of supporting continuous tracking and monitoring during extended ocean voyages.

Future work should extend this foundation through hardware-in-the-loop validation on battery-powered microcontroller platforms, full LEO constellation testing under operational maritime conditions, integration with architectural optimization strategies (caching, database tuning), and exploration of hybrid transport protocols combining local MQTT clusters with optimized satellite backhaul. The methodology and findings presented here provide both a technical roadmap and economic justification for organizations deploying massive-scale IoT systems in the era of terrestrial/satellite network convergence, with direct applicability to the critical use case of multimodal dry container tracking and monitoring across global supply chains.

Author Contributions: Conceptualization, N.K., M.F., F.K.D.A. and P.P.; methodology, M.F., N.K. and P.P.; software, N.K. and F.K.D.A.; validation, N.K.; formal analysis, N.K., M.F., F.K.D.A. and P.P.; investigation, N.K.; resources, M.F., F.K.D.A. and P.P.; data curation, N.K. M.F., F.K.D.A. and P.P.; writing—original draft preparation, N.K., M.F. and F.K.D.A.; writing—review and editing, M.F., N.K., F.K.D.A. and S.N.; visualization, N.K. and M.F.; supervision, M.F. and P.P. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: The original contributions presented in this study are included in the article. Further inquiries can be directed to the corresponding author.

Acknowledgments: The authors gratefully acknowledge the support of the CNIT PNTLab in providing research infrastructure and the Astrocast platform for enabling real-world validation experiments. Special thanks to the University of Pisa Department of Computer Science and Networking for providing the academic environment supporting this research.

Conflicts of Interest: The authors declare no conflicts of interest.

Abbreviations

The following abbreviations are used in this manuscript:

5G	Fifth Generation (Mobile Communication System)
API	Application Programming Interface
CBOR	Concise Binary Object Representation
CGI	Cell Global Identity
CoAP	Constrained Application Protocol
CPU	Central Processing Unit
CRUD	Create, Read, Update, and Delete
FoM	Figure of Merit
GNSS	Global Navigation Satellite System
HIL	Hardware-In-the-Loop
HDOP	Horizontal Dilution of Precision
HTTP	HyperText Transfer Protocol
IETF	Internet Engineering Task Force
IoT	Internet of Things
JSON	JavaScript Object Notation
LEO	Low Earth Orbit
LTE-M	Long-Term Evolution for Machines
M2M	Machine-to-Machine
MCU	MicroController Unit
MO	Mobile-Originated
MQTT	Message Queuing Telemetry Transport
MSISDN	Mobile Subscriber ISDN Number
MT	Mobile-Terminated
NB-IoT	Narrowband Internet of Things
NTN	Non-Terrestrial Network
RPS	Requests Per Second
RSSI	Received Signal Strength Indicator
SIM	Subscriber Identity Module
TN	Terrestrial Network
VM	Virtual Machine

References

1. Qiu, F.; Kumar, A.; Hu, J.; Sharma, P.; Tang, Y.B.; Xu, X.Y.; Hong, J. A Review on Integrating IoT, IIoT, and Industry 4.0: A Pathway to Smart Manufacturing and Digital Transformation. *IET Inf. Secur.* **2025**, *2025*, 9275962. [CrossRef]
2. Singhvi, H. The internet of things in 2025: Trends, business models, and future directions for a connected world. *Int. J. Internet Things* **2025**, *3*, 17–24. [CrossRef]
3. Falcitelli, M.; Noto, S.; Pagano, P.; Gharbaoui, M.; Isca, A.; Fresi, F.; Mancina, A.; Toffetti, M.; Amatruda, A.; Bendoni, N.; et al. Full-Scale Assessment of the "5GT System" for Tracking and Monitoring of Multimodal Dry Containers. *IoT* **2024**, *5*, 922–950. [CrossRef]
4. oneM2M Partnership Project. An Introduction to oneM2M's Architecture. Available online: https://recipes.onem2m.org (accessed on 29 December 2025).
5. Falcitelli, M.; Misal; Noto, S.; Pagano, P. Development of a Multi-Radio Device for Dry Container Monitoring and Tracking. *IoT* **2024**, *5*, 187–211. [CrossRef]

6. Hagaseth, M.; Rødseth, Ø.J.; Krogstad, T.; Bakke, M. A new architectural framework for digitalization of maritime intelligent transport systems. *TransNav Int. J. Mar. Navig. Saf. Sea Transp.* **2023**, *17*, 769–780. [CrossRef]
7. Bouter, C.; Biagioni, G.; van Gessel, T.; Korteling, W.; de Graaf, E.; Hofman, W. Towards a Modular Ontology for Event-Based Data Sharing in the Logistics Domain. In Proceedings of the Poster and Demo Track and Workshop Track of the 18th International Conference on Semantic Systems (SEMANTiCS 2022), Vienna, Austria, 13–15 September 2022; CEUR Workshop Proceedings; Volume 3235.
8. Bol Raap, W.; Iacob, M.-E.; van Sinderen, M.; Piest, S. An Architecture and Common Data Model for Open Data-Based Cargo-Tracking in Synchromodal Logistics. In *On the Move to Meaningful Internet Systems: OTM 2016 Conferences*; Debruyne, C., Panetto, H., Meersman, R., Dillon, T., Kühn, E., O'Sullivan, D., Ardagna, C.A., Eds.; Springer International Publishing: Cham, Switzerland, 2016; pp. 327–343. [CrossRef]
9. Gnimpieba, Z.D.R.; Nait-Sidi-Moh, A.; Durand, D.; Fortin, J. Using Internet of Things Technologies for a Collaborative Supply Chain: Application to Tracking of Pallets and Containers. *Procedia Comput. Sci.* **2015**, *56*, 550–557. [CrossRef]
10. Astrocast S.A. Astronode S DevKit Product Brief. Available online: https://www.astrocast.com/products/astronode-devkit/ (accessed on 23 March 2026).
11. u-blox AG; Astrocast S.A. Journey to the Center of Satellite IoT Connectivity. Available online: https://www.u-blox.com/en/blogs/tech/satellite-iot-connectivity (accessed on 29 December 2025).
12. Lin, X.; Rommer, S.; Euler, S.; Yavuz, E.A.; Karlsson, R.S. 5G from Space: An Overview of 3GPP Non-Terrestrial Networks. *IEEE Commun. Stand. Mag.* **2021**, *5*, 147–153. [CrossRef]
13. Liu, W.; Hou, X.; Wang, J.; Chen, L.; Yoshioka, S. Uplink Time Synchronization Method and Procedure in Release-17 NR NTN. In Proceedings of the IEEE 95th Vehicular Technology Conference (VTC2022-Spring); Helsinki, Finland, 19–22 June 2022; pp. 1–5. [CrossRef]
14. Jung, D.-H.; Nam, H.-J.; Choi, J. Coverage Analysis of GEO Satellite Networks. In Proceedings of the 22nd International Symposium on Modeling and Optimization in Mobile, Ad Hoc, and Wireless Networks (WiOpt 2024); Seoul, Republic of Korea, 21–24 October 2024; pp. 70–75.
15. Cai, Y.; Zhang, S.; Hou, J.; Jiao, H. Internet of Things over GEO Satellite: A Novel Space Information Network Solution. In *Proceedings of the 2021 International Wireless Communications and Mobile Computing (IWCMC)*; IEEE: Piscataway, NJ, USA, 2021; pp. 177–181. [CrossRef]
16. Jin, L.; Wang, L.; Jin, X.; Zhu, J.; Duan, K.; Li, Z. Research on the Application of LEO Satellite in IoT. In *Proceedings of the 2022 IEEE 2nd International Conference on Electronic Technology, Communication and Information (ICETCI)*; IEEE: Piscataway, NJ, USA, 2022; pp. 739–741. [CrossRef]
17. Bhattacharya, A.; Petrova, M. Study on Handover Techniques for Satellite-to-Ground Links in High and Low Interference Regimes. In *Proceedings of the 2023 Joint European Conference on Networks and Communications & 6G Summit (EuCNC/6G Summit)*; Gothenburg, Sweden, 6–9 June 2023; pp. 359–364. [CrossRef]
18. Bormann, C.; Hoffman, P.E. *Concise Binary Object Representation (CBOR)*; RFC 8949; RFC Editor: Fremont, CA, USA, 2020. [CrossRef]
19. MessagePack. Available online: https://msgpack.org/ (accessed on 30 December 2025).
20. Miniz, Version 0.0.1; Bernstone, L., Maintainer. Small Zlib-Compatible Library. GitHub Repository. Available online: https://github.com/richgel999/miniz/ (accessed on 30 December 2025).
21. Protocol Buffers (Protobuf). Google. Available online: https://protobuf.dev/ (accessed on 30 December 2025).
22. oneM2M Partnership Project. Published Specifications. Available online: https://www.onem2m.org/technical/published-specifications (accessed on 30 December 2025).
23. IoTOcean Developers. Mobius—Module Archives. IoTOcean Developers Portal. Available online: http://developers.iotocean.org/archives/module/mobius (accessed on 29 December 2025).
24. Pranavasri, V.J.S.; Francis, L.; Pal, G.; Mogadali, U.; Vattem, A.; Vaidhyanathan, K.; Gangadharan, D. Exploratory study of oneM2M-based interoperability architectures for IoT: A smart city perspective. In *2024 IEEE 21st International Conference on Software Architecture Companion (ICSA-C)*; IEEE Computer Society: Los Alamitos, CA, USA, 2024; pp. 16–23. [CrossRef]
25. Medina-Acosta, G.A.; Zhang, L.; Chen, J.; Uesaka, K.; Wang, Y.; Lundqvist, O.; Bergman, J. 3GPP Release-17 Physical Layer Enhancements for LTE-M and NB-IoT. *IEEE Commun. Stand. Mag.* **2022**, *6*, 80–86. [CrossRef]
26. SpaceX. Starlink Direct to Cell Service Now Available. February 2025. Available online: https://starlink.com/public-files/DIRECT_TO_CELL_SERVICE_FEB_25.pdf (accessed on 8 January 2026).
27. SpaceX. Starlink Business: Direct To Cell. 2025. Available online: https://starlink.com/business/direct-to-cell (accessed 8 January 2026).
28. Ralph, E. SpaceX's First (Public) Acquisition Is Nanosat and Internet of Things Startup Swarm. *Teslarati*, 8 August 2021. Available online: https://www.teslarati.com/spacex-acquires-swarm-technologies-iot-constellation/ (accessed on 16 January 2026).

29. Iridium Communications Inc. Iridium Certus 9770 Module. 2024. Available online: https://www.iridium.com/products/iridium-certus-9770/ (accessed on 8 January 2026).
30. Apollo Satellite. Iridium Certus Global Land & Maritime Internet. September 2022. Available online: https://apollosat.com/iridium-certus/ (accessed on 8 January 2026).
31. Ilcev, D.S. Architecture of Orbcomm Space Segment for Global Mobile Satellite Communications (MSC) Networks. *J. Marit. Res.* **2024**, *21*, 398–403. Available online: https://www.jmr.unican.es/jmr/article/view/902/887 (accessed on 8 January 2026).
32. Ilcev, D.S. Architecture of Orbcomm Ground Segment for Global Mobile Satellite Communications (MSC) Networks. *J. Marit. Res.* **2024**, *21*, 60–64. Available online: https://www.jmr.unican.es/jmr/article/view/903/908 (accessed on 8 January 2026).
33. Kinéis. Space-Based IoT connectivity Everywhere, All the Time. June 2024. Available online: https://kineis.com/en/spatial-iot-connectivity/ (accessed on 8 January 2025).
34. Kinéis. Kinéis FAQs—Space-Based IoT Connectivity. 2025. Available online: https://kineis.com/en/faqs/ (accessed on 8 January 2026).
35. Iridium Communications Inc. *Iridium Short Burst Data Service Developers Guide*; Release 3.0; Iridium Communications Inc.: McLean, VA, USA, 2012. Available online: https://www.ydoc.biz/download/IRDM_IridiumSBDService.pdf (accessed on 16 January 2026).
36. Kinéis. Data Uplink Service. 2024. Available online: https://connect.kineis.com/en_US/services-overview/data-uplink (accessed on 16 January 2026).
37. Viotti, J.C.; Kinderkhedia, M. A Survey of JSON-Compatible Binary Serialization Specifications. *arXiv* **2022**, arXiv:2201.02089. [CrossRef]
38. Biswal, A.K.; Al Mallah, O. Analytical Assessment of Binary Data Serialization Techniques in IoT. Master's Thesis, Politecnico di Milano, Milan, Italy, December 2019. Available online: https://www.politesi.polimi.it/handle/10589/150617 (accessed on 29 December 2025).
39. Nanopb Contributors. Nanopb: Small-Footprint Protocol Buffers Implementation for Embedded Systems; GitHub Repository, Release v0.4.5 (Commit 1a2b3c4), 2025. Available online: https://github.com/nanopb/nanopb (accessed on 30 December 2025).
40. Espressif Systems. *ESP32-S3 Series—Datasheet*; Espressif Systems: Shanghai, China, 2022. Available online: https://www.espressif.com/sites/default/files/documentation/esp32-s3_datasheet_en.pdf (accessed on 29 December 2025).
41. Pereira, F.; Correia, R.; Pinho, P.; Lopes, S.I.; Carvalho, N.B. Challenges in resource-constrained IoT devices: Energy and communication as critical success factors for future IoT deployment. *Sensors* **2020**, *20*, 6420. [CrossRef]
42. Friesel, D.; Spinczyk, O. Data serialization formats for the Internet of Things. *Electron. Commun. EASST* **2021**, *80*. Available online: https://eceasst.org/index.php/eceasst/article/download/2256/2428/2439 (accessed on 29 December 2025).
43. Jackson, S.; Cummings, N.; Khan, S. Streaming Technologies and Serialization Protocols: Empirical Performance Analysis. *arXiv* **2024**, arXiv:2407.13494. [CrossRef]
44. Maltsev, E.; Muliarevych, O. Beyond JSON: Evaluating Serialization Formats for Space-Efficient Communication. *Adv. Cyber-Phys. Syst.* **2024**, *9*, 9–15. [CrossRef]
45. Huseynov, K.; Cakir, L.V.; Al-Shareeda, S.; Özdem, M.; Canberk, B. A data serialization-based framework for efficient IoT management. In *Proceedings of the 2024 IEEE 10th World Forum on Internet of Things (WF-IoT)*; IEEE: Piscataway, NJ, USA, 2024; pp. 707–712. [CrossRef]
46. Luis, Á.; Casares, P.; Cuadrado-Gallego, J.J.; Patricio, M.A. PSON: A serialization format for IoT sensor networks. *Sensors* **2021**, *21*, 4559. [CrossRef] [PubMed]
47. Molina Araque, S.; Martinez, I.; Papadopoulos, G.Z.; Montavont, N.; Toutain, L. Yet another compact time series data representation using CBOR templates (YACTS). *Sensors* **2023**, *23*, 5124. [CrossRef] [PubMed]
48. Zandberg, K.; Gulati, M.; Wunder, G.; Baccelli, E. Model CBOR Serialization for Federated Learning. *arXiv* **2024**, arXiv:2401.14056. [CrossRef]
49. Digital Container Shipping Association. IoT Commercial Events Standard Documentation. Available online: https://dcsa.org/standards/track-and-trace/documentation-iot-commercial-events (accessed on 31 December 2025).
50. Digital Container Shipping Association. DCSA Information Model 2023.Q1. March 2023. Available online: https://dcsa-website.cdn.prismic.io/dcsa-website/65ca12999be9a5b998b5ad40_20230301-DCSA-InformationModelQ1.pdf (accessed on 31 December 2025).
51. Esmat, H.H.; Lorenzo, B.; Shi, W. Toward Resilient Network Slicing for Satellite–Terrestrial Edge Computing IoT. *IEEE Internet Things J.* **2023**, *10*, 14621–14645. [CrossRef]
52. Mohammed, A.S.; Venkatachalam, K.; Hubálovský, S.; Trojovský, P.; Prabu, P. Smart Edge Computing for 5G/6G Satellite IoT for Reducing Inter Transmission Delay. *Mob. Netw. Appl.* **2022**, *27*, 1050–1059. [CrossRef]
53. Kim, T.; Kwak, J.; Choi, J.P. Satellite Edge Computing Architecture and Network Slice Scheduling for IoT Support. *IEEE Internet Things J.* **2022**, *9*, 14938–14951. [CrossRef]

54. Jazaeri, S.S.; Jabbehdari, S.; Asghari, P.; Haj, Seyyed, Javadi, H. Edge Computing in SDN-IoT Networks: A Systematic Review of Issues, Challenges and Solutions. *Cluster Comput.* **2021**, *24*, 3187–3228. [CrossRef]
55. oneM2M Partnership Project. oneM2M Completes 10 Years. Available online: https://www.onem2m.org/iot-news/802-onem2m-completes-10-years (accessed on 30 December 2025).
56. oneM2M Partnership Project. Release 5 Development Status. Available online: https://www.onem2m.org/technical/published-specifications/release-5 (accessed on 30 December 2024).
57. Troscia, M.; Sgambelluri, A.; Paolucci, F.; Castoldi, P.; Pagano, P.; Cugini, F. Scalable OneM2M IoT Open-Source Platform Evaluated in an SDN Optical Network Controller Scenario. *Sensors* **2022**, *22*, 431. [CrossRef] [PubMed]
58. Datta, S.K.; Gyrard, A.; Bonnet, C.; Boudaoud, K. oneM2M Architecture Based User-Centric IoT Application Development. In *Proceedings of the 3rd International Conference on Future Internet of Things and Cloud (FiCloud 2015)*; IEEE: Piscataway, NJ, USA, 2015; pp. 100–107. [CrossRef]
59. Locust-A Modern Load Testing Framework. Available online: https://locust.io (accessed on 29 December 2025).
60. *ISO 6346:2022*; Freight Containers—Coding, Identification and Marking; International Organization for Standardization: Geneva, Switzerland, 2022.
61. Kasper, B.; Pawlitzek, R.; Hellwig, M. Towards Sustainable IoT: Space Efficiency and Serialization Speed of Data Exchange Formats. In *Proceedings of the 2025 IEEE International Conference on Engineering, Technology, and Innovation (ICE/ITMC)*; IEEE: Piscataway, NJ, USA, 2025; pp. 1–10. [CrossRef]
62. Astrocast. Event Portfolio & Service—ArrowTrackSAT. 2025. Available online: https://www.astrocast.com/event-portfolio-service-arrowtracksat/ (accessed on 2 January 2026).