CEA Comité Español de Automática
XLVII JORNADAS DE AUTOMÁTICA CÓRDOBA 2026
UNIVERSIDADE DA CORUÑA

Jornadas de Automática

Virtual networking and routing in partitioned embedded systems

Ferran Vicent[a,*], Miguel Masmano[1]

[a]*Fent Innovative Software Solutions, S.L. (FENTISS), Ciudad Politécnica de la Innovación, Edificio 9B Despacho 1.67 Camino de Vera s/n, 46022, Valencia, España*

To cite this article: Vicent, F., Masmano, M. 2026. Virtual Networking and Routing in Partitioned Embedded Systems. Jornadas de Automática, 47. https://doi.org/10.17979/ja-cea.2026.47.13756

Resumen

En el ámbito de los sistemas embebidos críticos, el uso de hipervisores como XtratuM Next Generation (XNG) permite ejecutar múltiples entornos concurrentemente, llamados particiones, garantizando una estricta segregación espacial y temporal. Estas particiones requieren canales de comunicación estándar para interactuar entre sí y con redes externas de forma segura y sin comprometer la integridad del sistema. Este artículo presenta una arquitectura de red robusta y ligera para sistemas particionados. Al utilizar Interfaces de Red Virtuales (vNICs) mapeadas sobre memoria compartida integradas con una pila TCP/IP *lwIP* (Lightweight IP), se establece una arquitectura capaz de generar una red virtual entre particiones. Adicionalmente, la adopción de protocolos estándar como TCP/IP acerca el sistema a los estándares de la industria, evitando la necesidad de reescribir los mecanismos de comunicación cuando se interactúa con sistemas operativos complejos como Linux. Finalmente, la arquitectura demuestra que es capaz de establecer comunicación entre aplicaciones bare-metal y entornos de sistemas operativos invitados en un mismo sistema embebido.

Palabras clave: Arquitectura de software de control, Arquitecturas de control virtualizadas, Seguridad IT/OT en sistemas de automatización, Sistemas ciberfísicos, Seguridad y protección en control en red

Virtual networking and routing in partitioned embedded systems

Abstract

In the field of critical embedded systems, the use of hypervisors such as XtratuM Next Generation (XNG) allows for the concurrent execution of multiple environments, known as partitions, guaranteeing strict spatial and temporal segregation. These partitions require standard communication channels to interact with each other and with external networks securely, without compromising system safety. This paper presents a robust and lightweight networking architecture for partitioned systems. By utilizing Virtual Network Interfaces (vNICs) mapped over shared memory in conjunction with the *lwIP* (Lightweight IP) TCP/IP stack, an architecture capable of generating a virtual network between partitions is established. Additionally, the adoption of standard protocols like TCP/IP aligns the system with industry standards, eliminating the need to rewrite communication mechanisms when interfacing with complex operating systems such as Linux. Finally, the architecture demonstrates its capability to establish seamless communication between bare-metal applications and guest operating system environments within the same embedded system.

Keywords: Control software architecture, Virtualized control architectures, Safety and security in networked control, Virtualized control architectures, Cyber physical systems.

*Autor para correspondencia: fvicent@fentiss.com

1. Introduction

Critical embedded systems utilize partitioned architectures managed by hypervisors, such as XtratuM Next Generation (XNG), to guarantee strict spatial and temporal isolation (Kopetz, 2011), allowing multiple execution environments to run concurrently on the same hardware platform. Nevertheless, perfectly isolated partitions often require reliable and efficient standard communication channels to interact with each other and with external networks without compromising system safety.

Adopting standard protocols like TCP/IP aligns the internal networking with industry standards. This approach eliminates the necessity of building custom adaptations when communicating with complex operating systems such as Linux, increasing the connectivity of the bare-metal partitions.

This paper presents a lightweight networking architecture for partitioned systems. An internal routing engine is implemented by utilizing Virtual Network Interfaces (vNICs) mapped over shared memory and integrating the *lwIP* (Lightweight IP) stack. Furthermore, the architecture demonstrates seamless interoperability between bare-metal applications and Linux guest OS environments through customized network layers. Part of this development has been carried out within the context of the EdgeAI-Trust project (European Commission, 2024) to develop a secure system software with networking functionalities.

2. Core Technologies and Architecture

The foundation of the proposed communication architecture relies on bypassing traditional hardware network controllers for inter-partition traffic, utilizing instead inter-partition shared memory structures.

2.1. XtratuM Next Generation (XNG) Hypervisor

At the base of this architecture operates XNG, a bare-metal (Type-1) hypervisor specifically designed for safety-critical embedded systems (Masmano et al., 2009). XNG guarantees strict spatial isolation by utilizing the processor's Memory Management Unit (MMU) to confine each partition to its own private memory space. Furthermore, it enforces temporal isolation through a deterministic cyclic scheduler, granting each partition a specific execution time slot (Kopetz, 2011). This ensures that a fault in one partition cannot exhaust the CPU resources required by others.

2.2. Virtual Network Interfaces (vNICs) over Shared Memory

A Virtual Network Interface (vNIC) is a software abstraction that mimics the behavior of a physical network card, but operates entirely within the system's memory. Instead of transmitting information over a physical cable, partitions exchange data by writing to and reading from predefined shared memory regions managed by the hypervisor.

To achieve high-throughput communication, this implementation of the vNIC uses shared regions that are structured using double-linked circular lists and ring buffers. To ensure memory safety across different virtual address spaces mapped by partitions using complex OSes such as Linux, the vNIC implementation cleverly uses relative memory offsets rather than absolute pointers.

2.3. Lightweight IP (lwIP) Integration

To process standard network protocols like TCP or UDP over the vNICs, the architecture integrates *lwIP*, a highly optimized TCP/IP stack designed for resource-constrained embedded systems (Dunkels, 2001).

In this architecture, *lwIP* is integrated into the bare-metal partitions in a no-sys configuration. This means it operates without an underlying operating system, utilizing a single-threaded, deterministic polling mechanism to check for incoming packets. Crucially, this polling is strictly confined within the time slots allocated by the XNG scheduler, ensuring that network processing respects the temporal isolation of the system.

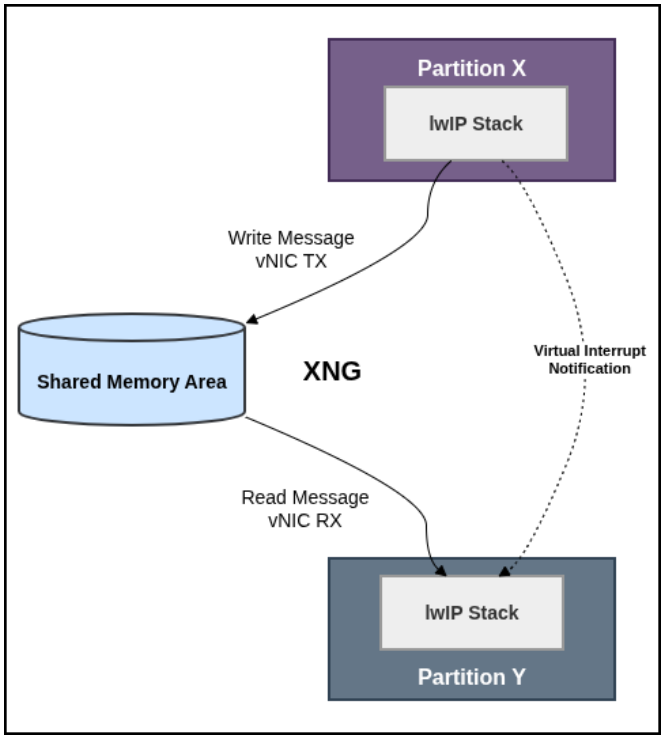

Figure 1: Conceptual data flow between isolated partitions utilizing vNICs and shared memory.

As illustrated in Figure 1, the inter-partition communication relies on a secure and efficient sequence of events without bypassing the hypervisor's isolation. When an application in a partition needs to transmit data, vNIC processes the packet and writes the message directly into the predefined Shared Memory Area. Immediately after, a virtual interrupt is triggered to notify the destination. Upon receiving this notification, the *lwIP* stack in the receiving partition retrieves the message from the shared memory. This mechanism ensures a highly efficient, low-latency data exchange while fully preserving the spatial boundaries enforced by XNG.

3. Additional Features of the Architecture

While XNG, vNICs, and *lwIP* provide the basic requirements for isolated inter-partition communication, the system is able to configure more complex network topologies that extend the functionality from a basic point-to-point communication between bare-metal partitions to multiple virtual networks.

Particulary, this section details how the architecture establishes a centralized network router capable of acting as an internal switch, while securely interfacing with the physical world and general-purpose operating systems like Linux.

3.1. Network Router

To interconnect the local network created by this architecture with other external networks, a dedicated bare-metal partition can be assigned to act as the central router and gateway among every partition on the system. This routing engine adds an extra layer of complexity to the system providing multiple internal virtual networks (e.g., vNET1 and vNET2) alongside with support to other advanced features.

3.2. Linux Integration via meta-network-xng

While bare-metal partitions handle highly critical tasks, systems often require a general purpose OS like Linux for complex data processing or for application layers with multiple dependencies.

For that specific application the XNG hypervisor enables the application layer developers to integrate any software that requires dependencies only provided by complex OSes like Linux. This is achieved without compromising the temporal and spatial isolation of any of the other partitions on the system.

Linux support on XNG is achieved through a Yocto-based Linux distribution with modifications to the Linux kernel that abstract hardware-specific dependencies. This approach enables Linux to operate as a secure and isolated guest OS within XNG. Therefore, to integrate Linux into the partitioned network, a custom Yocto/OpenEmbedded layer named *meta-network-xng* was developed. This layer introduces a specialized kernel module that exposes the XNG shared memory vNICs as standard Linux network interfaces (e.g., *eth0*). Consequently, the Linux network stack is able to communicate within its assigned virtual network and reach other virtual networks or external devices through the centralized router, without breaking the boundaries enforced by XNG.

3.3. External Hardware Routing

The partitions inside XNG have a specific IP address assigned to them. Therefore, when a partition sends a packet to another IP, the *lwIP* stack first checks if that specific IP belongs to a partition inside the same virtual network. If it does not, the packet is forwarded to the router partition that will check every other virtual network for a match. If the router is not able to find that IP, it will assume that this message belongs to an external IP address and it will forward the traffic to the specific Ethernet driver of that system.

Additionally, because the internal virtual traffic skips standard hardware-level checks for efficiency, the router must manually calculate and attach the required network integrity signatures (IP, UDP, and TCP checksums) in software before dispatching any data to the outside world. This ensures that the packets follow the standard format required for messaging with general purpose OS.

4. System Deployment Example

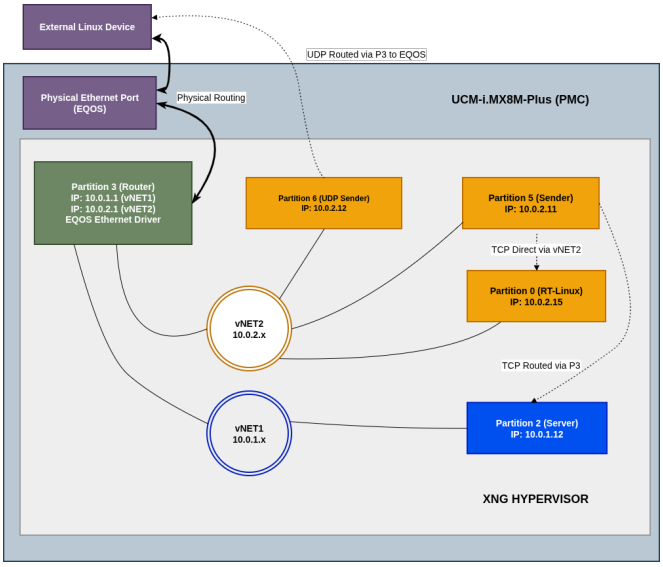

Figure 2: Example of a partitioned system with the core and the additional features.

To validate the technology, a complex network topology was deployed on a UCM-i.MX8M-Plus platform running the XNG Hypervisor.

As illustrated in Figure 2, the architecture consists of multiple internal subnetworks. The central router (Partition 3) holds the physical routing capabilities and manages traffic across vNET1 (10.0.1.x) and vNET2 (10.0.2.x). This layout enables to test every feature developed for the architecture:

- **vNET Traffic:** Bare-metal instances (Partition 5) can establish direct TCP connections with the Linux environment (Partition 0) over the same virtual subnetwork without routing overhead.
- **Inter-vNET Routing:** Applications in different subnets (Partition 5 and Partition 2) can communicate via TCP, with traffic routed through Partition 3.
- **External Routing:** Internal nodes (Partition 6) can send UDP and TCP streams to external devices. The routing engine intercepts the virtual traffic, applies the necessary software checksums and Ethernet padding, and forwards the frames through the physical EQOS port.

Table 1: System Architecture Throughput and Scheduling Configuration.

Scenario	Scheduling Profile	Window Allocation	Protocol	TX	RX
1. Slow Major Frame	Single-Core, 400ms cycle	**P5:** 150ms @ 0ms **P3:** 100ms @ 150ms **P2:** 50ms @ 250ms	TCP	~40 kbps	~40 kbps
2. Intermediate Major Frame	Single-Core, 40ms cycle	**P5:** 15ms @ 0ms **P3:** 10ms @ 15ms **P2:** 5ms @ 25ms	TCP	~4.2 Mbps	~4.2 Mbps
3. Fastest Major Frame	Single-Core, 10ms cycle	**P5:** 3ms @ 0ms **P3:** 4ms @ 3ms **P2:** 3ms @ 7ms	TCP	~35.5 Mbps	~35.4 Mbps
4. Raw Hardware Limit	Single-Core, 10ms cycle	**P5:** 3ms @ 0ms **P3:** 4ms @ 3ms **P2:** 3ms @ 7ms	UDP	~73 Mbps	~63 Mbps
5. True SMP (Failed)	Multi-Core, 100% Overlap	**P5 (CPU2):** 10ms @ 0ms **P3 (CPU0):** 10ms @ 0ms **P2 (CPU1):** 10ms @ 0ms	Both	N/A	Crash
6. Staggered SMP	Multi-Core, 10ms Staggered	**P3 (CPU0):** 4ms @ 0ms **P5 (CPU2):** 3ms @ 4ms **P2 (CPU1):** 4ms @ 4ms	UDP	~78 Mbps	~65 Mbps

Table 1 presents the throughput of the system architecture between Partition 5 (P5) and Partition 2 (P2). These partitions were selected to establish a controlled testing environment for evaluating the throughput of the vNIC and router, without the computational overhead of a complex operating system such as Linux.

The *Scenario* column identifies the evaluated execution configuration. The *Scheduling Profile* describes the Major Frame duration, which defines the periodic execution cycle of the system, and the processor allocation strategy, distinguishing between single-core and multi-core execution. The *Window Allocation* column specifies the temporal partitioning schedule, including the execution window assigned to each partition and its start offset within the Major Frame. The execution windows are fixed time slots within the Major Frame assigned to each partition that ensure deterministic execution time in each partition. The *Protocol* column indicates whether the communication is performed using TCP or UDP, while *TX* and *RX* represent the measured transmission and reception throughput, respectively.

As observed in the table, the throughput scales inversely with the Major Frame duration. The primary bottleneck validating these results is the queue depth of the vNIC ring buffers. Specifically, P5 exhausts the entire queue capacity within the first few milliseconds of the execution. During the remainder of the execution time of that partition every new packet is discarded because it cannot be queued. Furthermore, when we measure the received data at P2 it is strictly bounded to the queue depth of P5. This limitation can be mitigated by migrating the architecture to a multi-core environment, where P2 and P3 can process the data concurrently while P5 produces it. However, uncoordinated simultaneous access to shared memory inevitably leads to race conditions and memory corruption. Therefore, this queue bottleneck is only safely overcome by deploying a Staggered SMP Schedule, which leverages multiple cores to process data in parallel, but uses deterministic time offsets to prevent both queue saturation and memory collisions.

Additionally, as observed in 1, UDP achieves higher throughput than TCP due to its lower protocol overhead, as it avoids connection establishment, acknowledgments, and congestion control mechanisms, which introduce additional latency.

5. Conclusions

The proposed architecture successfully demonstrates that partitioned critical systems can leverage robust, standard-compliant networking without compromising spatial or temporal isolation. By combining shared memory, vNICs, the *lwIP* stack, and the custom meta-network-xng integration for Linux, the system achieves a highly flexible internal topology that complies with the widely used TCP and UDP protocols. Centralizing the physical MAC driver within a dedicated router partition not only simplifies hardware multiplexing but also provides a secure, monitored gateway for all external communications.

Furthermore, the evaluation of this architecture revealed the current constraints and limitations are dependent of the scheduling policy and of the queue depth. Moreover, the analysis demonstrated that protocols such as TCP are heavily bottlenecked by the time imposed by the Major Frame duration.

To overcome the queue bottleneck without sacrificing memory integrity, the deployment of a Staggered Symmetric Multiprocessing schedule proved effective. That allowed concurrency between partitions achieving a maximum raw throughput of 78 Mbps. Ultimately, this will make more feasible the implementation of network-related application layers with dependencies like TCP or UDP, reducing the overall effort of developing software over the XNG Hypervisor.

5.1. Future Work

Future work in this network architecture will focus on extending its compatibility across different versions of the XNG hypervisor, ensuring broader applicability. Additionally, efforts will be directed towards reinforcing the overall stability of the architecture by fully completing and consolidating the support for Symmetric Multiprocessing (SMP), addressing current concurrent access limitations. Finally, future iterations will focus on deploying and validating real-world application software that leverages the established network topology, proving the system's capacity to route communication between internal partitions and external networks.

References

Dunkels, A. (2001). "Design and Implementation of the lwIP TCP/IP Stack." *Swedish Institute of Computer Science, Technical Report 51*. URL: https://www.researchgate.net/publication/316051065_Design_and_Implementation_of_the_lwIP_in_Embedded_System

Kopetz, H. (2011). *Real-Time Systems: Design Principles for Distributed Embedded Applications*. Springer Science & Business Media. DOI: https://doi.org/10.1007/978-1-4419-8237-7

Masmano, M., Ripoll, I., Crespo, A., and Metge, J. J. (2009). "XtratuM: a hypervisor for safety critical embedded systems." *Proceedings of the 11th Real-Time Linux Workshop (RTLWS'09)*, Dresden, Germany. URL: https://www.fentiss.com/wp-content/uploads/2022/11/XM_for_Safety_Critical_Embedded_Systems.pdf

European Commission (2024). "Decentralized Edge Intelligence: Advancing Trust, Safety, and Sustainability in Europe (EdgeAI-Trust)." *CORDIS, Horizon Europe Project Fact Sheet*. Grant Agreement ID: 101139892. DOI: https://doi.org/10.3030/101139892

Acknowledgements

This work has been partially supported by the EdgeAI-Trust project. EdgeAI-Trust "Decentralized Edge Intelligence: Advancing Trust, Safety, and Sustainability in Europe" project has received funding from Chips Joint Undertaking (Chips JU) under grant agreement No 101139892.