



POLITECNICO
MILANO 1863

**SCUOLA DI INGEGNERIA INDUSTRIALE
E DELL'INFORMAZIONE**

EXECUTIVE SUMMARY OF THE THESIS

A platform for efficient and observable management of multi-tenant Neutral Host deployments

LAUREA MAGISTRALE IN INGEGNERIA INFORMATICA
COMPUTER SCIENCE AND ENGINEERING

Author: ANDREA CARAVANO

Advisor: PROF. MARCO MEZZAVILLA

Co-advisors: PROF. ANTONIO CAPONE, PROF. EUGENIO MORO

Academic year: 2025-2026

1. Introduction

Mobile Network Operators (MNOs) play a fundamental role in the modern era: an increasingly-high amount of needs have emerged, spanning from low-latency communications to real-time industrial automation.

As the geopolitical and regulatory landscape evolves, an increasing variety of concerns become more and more relevant: **supply-chain resilience, vendor lock-in and the need for transparent and auditable systems**.

New players have therefore entered the market. Among those, the one that gained more traction is the **Neutral Host**, a specialized third party entity that owns, deploys and manages a shared physical computation and radio infrastructure that is then leased to multiple MNOs [1].

The aim of this work is providing an open-source, **observable and efficiency-aware solution towards Neutral Host deployments**.

2. A computationally efficient 5G deployment

RED HAT OPENSIFT and its community project OKD represent an increasingly popular integrated ecosystem designed to abstract and automate management of a multi-node KUBERNETES cluster. KUBERNETES is an open-source solution for **multi-node containerized environments**.

Unlike standard KUBERNETES distributions, where the underlying OS and container orchestrator are decoupled, OPENSIFT merges these two layers adopting the **Infrastructure as Code** (IaC) paradigm.

2.1. A custom solution

As of June 2026, replicating a stable OKD setup is a non-trivial exercise, because of underlying incompatibilities across its distributed components.

We would therefore like to keep its principles and ease of declarative configuration and replace OKD with a custom solution which allows for:

- Declarative configuration
- Custom utilities

- Efficient tuning and optimization
- Generality

In recent years, the industry answered these needs with **ANSIBLE**, a descriptive open-source **automation language for any-scale IT deployments**, allowing for determinism and declarativity through IaC.

An ANSIBLE-managed environment is a standard GNU/Linux distribution (e.g. Debian / Ubuntu), the underlying operational mechanisms are the traditional ones we are used to, while ANSIBLE is able to act as an external enforcement engine, defeating configuration drifts.

2.2. Recreating a middle ground: the Ansible playbook

In ANSIBLE, the PLAYBOOK is the set of practical actions that will be applied on the chosen set of machines to which our deployment will apply. In our case, that will be all the machines included in our cluster.

We will:

- Apply system and kernel tuning for RAN workloads
- Bootstrap cluster nodes
- Configure networking interfaces
- Deploy a load balancer
- Bootstrap server and agent nodes
- Join additional server nodes to the Control Plane
- Apply static manifests after cluster creation
- Finalize updates and dependencies

Linux achieves Real-time Operating System (RTOS) goals through the **PREEMPT_RT** patch, guaranteeing **fixed bounds for chosen workloads**.

To mitigate the architectural heaviness of a standard Vanilla KUBERNETES deployment, RANCHER LABS developed K3S, a certified and lightweight KUBERNETES distribution designed specifically for efficient cluster deployments. K3S checks all the boxes in our list and allows for **complex networking interface definitions** through custom CNI plugins.

2.2.1 An example snippet

We will shortly use a Software Defined Radio (SDR) which makes use of SFP+ networking interfaces: let's try automating their configuration using ANSIBLE and its **templating** engine.

```
1 - name: Generate configuration for
  ↳ high-speed USRP N3xx interfaces
```

```
2 ansible.builtin.template:
3   src: templates/usrp.yaml.j2
4   dest: /etc/netplan/70-usrp.yaml
5   mode: 0600
```

A template is an ANSIBLE object that patches a pre-defined artifact with default values and variables.

```
1 network:
2   version: 2
3   ethernet:
4     {{ usrp_interface_1 }}:
5       addresses:
6         - {{ usrp_interface_1_ip }}
7       mtu: 9000
8       dhcp4: false
9       dhcp6: false
10    {{ usrp_interface_2 }}:
11      addresses:
12        - {{ usrp_interface_2_ip }}
13      mtu: 9000
14      dhcp4: false
15      dhcp6: false
```

Variables are therefore customizable:

```
1 usrp_interface_1: enp65s0f0np0
2 usrp_interface_2: enp65s0f1np1
3 usrp_interface_1_ip: 192.168.50.1/24
4 usrp_interface_2_ip: 192.168.60.1/24
```

Having now completed both the management and deterministic data planes, the next natural requirement is High Availability. In a multi-node cluster, a Virtual IP is an IP address not permanently tied to a specific physical NIC or MAC address, but instead **floating across healthy nodes** in the cluster. We chose KUBE-VIP to achieve this and deployed its static manifest through KUBERNETES API and abstractions:

```
1 - name: Create directory for Kube VIP
  ↳ Manifest
2   ansible.builtin.file:
3     path: .../manifests
4     ...
5   when: inventory_hostname in
  ↳ groups['servers']
6 - name: Deploy Kube VIP Configuration
7   ansible.builtin.template:
8     src: templates/kube-vip-config.j2
9     dest: .../manifests/kube-vip.yaml
10    ...
11   when: inventory_hostname ==
  ↳ groups['servers'][0]
```

We are now able to identify one of the most relevant details to our solution: **generality**.

Since any set of actions running within ANSIBLE can be generalized to be extended to potentially any group of machines, our solution is **already valid for scalable deployments**.

2.3. System tuning and optimization

The foundation for determinism begins at the bootloader: passing specific parameters to the Linux Kernel via the bootloader allows for deep customization of the underlying hardware [2].

2.3.1 A notable example: CPU isolation

The following snippet instructs the Kernel scheduler to avoid placing generic user-space processes on specified cores.

```
isolcpus=2-11,14-23 nohz_full=2-11,14-23
→ rcu_nocbs=2-11,14-23
→ kthread_cpus=0-1,12-13
→ irqaffinity=0-1,12-13
```

We would like, instead, to **reserve them exclusively for 5G network functions and base station** (GNB) to allow for precise tracking and deterministic performance.

We finally apply some memory management and buffers optimization, hardware passthrough and thread management.

2.3.2 Run-time tuning

The **tuned** daemon is a commonly-used daemon for dynamic run-time configurations.

We applied adjustments to **CPU governors, virtual memory and network stack** (e.g. ring buffers) and finally enforced them on the KUBELET, the agent running on each cluster node.

3. A Neutral Host management platform

3.1. The Open-Source 5G Ecosystem

OPEN5GS is an open-source C implementation of the 5G Core. Control Plane functions are **decoupled microservices**: they communicate across an internal message bus via HTTP.

We will mostly interact with the following components:

- Access and Management Function (AMF)
- Session Management Function (SMF)
- User Plane Function (UPF)

The OPENAIRINTERFACE Software Alliance provides a complete software stack, including

the **OAI GNB and FLEXRIC**, which we will extensively make use of in the following.

3.2. Deploying the core network

OPEN5GS consists of ~ 10 stateless microservices, described by its **HELM chart**, a structured directory containing text templates for variable injection and metadata.

We adapted this to properly configure **authentication keys, management interfaces and subnets** throughout our deployment.

3.2.1 Programmable SIM cards and commercial UEs

Throughout our work, we used programmable SIM cards, designed with open and flexible encryption layers for research and private mobile network deployments. We also made use of commercial USB-based UEs and smartphones for practical testing on real commercial hardware.

3.3. Traffic policy enforcement

The current implementation of pure radio slicing (PRB blocklisting and Service Differentiation) [3] does not allow us to apply it to USRPs, which restricts our abilities on traffic control and usage limitation [4].

We built a **custom container image for the UPF** which embeds two pivotal components: **iperf** for network validation and a custom **TRAFFIC CONTROLLER** to enforce networking policies on specific UEs and sessions.

Linux offers a TRAFFIC CONTROL subsystem: it governs how packets are queued, scheduled and dropped by network interfaces. Each networking interface has a set of associated Queueing Disciplines (qdisc): we will use the HIERARCHICAL TOKEN BUCKET (HTB), for traffic shaping.

In HTB, tokens are generated at a constant rate (the defined limit). To transmit a byte of data, a token must be consumed from the bucket. If the bucket is empty, the **packet is queued until enough tokens are generated**: the result is a **throughput enforcement solution**.

Each UE is associated with a class derived from its IPv4 address:

```
def get_class_and_prio(ip: str):
    minor_hex = f"{octet3:02x}-{octet4:02x}"
    ...
```

And its current **baseline throughput measured** in a 15-seconds estimation cycle:

```

1 def measure_current_bandwidth(ip: str) ->
  ↳ float:
2     cmd =
  ↳ f"timeout {MEASUREMENT_TIME} tcpdump
3     -i {INTERFACE} -n 'dst host {ip}'" ...
4     try:
5         result = subprocess.run(cmd, ...)
6         return mbps
7     ...

```

Embedding such a specialized component inside the UPF container allows us to exploit the **tunnelling interface** (ogstun) **connecting each UE to external networks** and therefore enforcing custom rate limiting policies.

3.4. Deploying OpenAirInterface

OAI also provides a HELM CHART for its deployment. We **adapted it to include support for Universal Software Radio Peripherals** (USRPs), in particular:

- USRP B210: compact, USB-powered SDR. We enabled hardware passthrough and device file mapping.
- USRP N300: high-performance SDR using dual 10 GbE SFP+ interfaces. We used MULTUS, a meta-CNI plugin for KUBERNETES allowing for declarative interface bridging.

3.4.1 Observability through Vector

VECTOR is a high-performance observability data pipeline for containerized environments. Thanks to its **extensive API integration with KUBERNETES** and its custom expression-oriented language, it enables non-trivial **data transformation** tasks.

We used VECTOR to:

- Match identity mappings (UE-ID to RNTI)
- Track the state of sessions
- Extract RF signal quality metrics
- Capture channel state, modulation and throughput

Results are encoded in **JSON payloads and forwarded to an HTTP Webhook**, written in PHP and hosted directly on the cluster through APACHE WEB SERVER.

The webhook uses REDIS for short-term storage and POSTGRES SQL for long-term storage.

3.5. Deploying FlexRIC

The Near-RT RIC is a software-defined platform allowing for third party applications (XAPPS) to

perform radio resource analysis, traffic inspection and metrics extraction with small latencies, thanks to its standardized service models (e.g. KPM).

We built a **custom containerized environment for FLEXRIC**, embedding an observability XAPP written in C, responsible for:

- Subscriptions management
- Data extraction and aggregation
- Data exfiltration

An exemplified code snippet, extracting the current percentage of used Physical Resource Blocks (PRBs) is the following:

```

1 char *name_str =
  ↳ cp_ba_to_str(info_item.meas_type.name);
2 char *name_unit = get_meas_unit(name_str);
3 ...
4 else if (strcmp(name_str, "RRU.PrbTotDl")
  ↳ == 0)
5     ue_stats[idx].PrbTotDl += val;
6 else if (strcmp(name_str, "RRU.PrbTotUl")
  ↳ == 0)
7     ue_stats[idx].PrbTotUl += val;

```

As before, aggregated **state and metrics are serialized in a JSON payload and forwarded** to a PHP processing solution for both short and long-term storage.

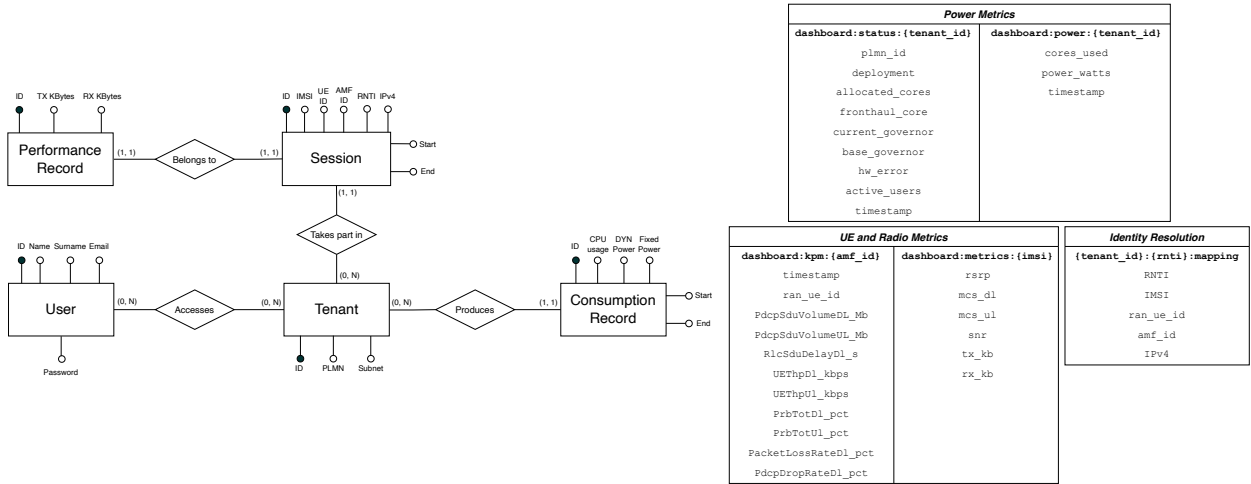
3.6. The identity resolution problem

Cross-layer UE identifiers correlation is a fundamental challenge in 5G networks: disaggregation of Network Functions have produced an innovative and competitive environment, while creating some architectural disadvantage:

Identifier	Visibility/context
RNTI	Only known to GNB and UE, no conception in 5G Core.
IMSI	Known to AMF and SMF, no conception in GNB.
NGAP IDs	Known to both GNB and AMF.
IPv4	Known to SMF and UPF, linked to NGAP IDs.

Identifiers in a 5G deployment

We resolved this challenge by exploiting the privileged access we have to the infrastructure as a whole, analyzing and dissecting the standard output and **log entries provided by the AMF, SMF and GNB** containers.



E-R model diagram and key-value pairs stored for the Neutral Host management platform

We built a GNB-Core mapper, which uses NGAP IDs as a crucial bridge to link the RNTI and IMSI identifiers.

The result is a custom container image that embeds a PYTHON mapper using the KUBERNETES API to **match relevant log lines in both GNB and Core Network** deployments.

It is also capable of managing re-establishments (brief radio signal losses), removed or updated PODS (cleaning up relevant sessions).

4. Experimental evaluation and use-cases

4.1. System profiling and energy tuning on AMD platforms

Since we are using an AMD workstation, we need to adapt our workload to use custom solutions coming from AMD: most general approaches are only compatible with Intel platforms and unofficial ones are not yet updated to support the CPU architecture in our workstation. Luckily, AMD provides the Host System Management Port and μ Prof, allowing for the request of a **target power limit** and **telemetry profiling**, respectively.

μ Prof produces a **CSV timechart**, outlining **core power draw**, thread effective **frequency** and CPU **heat dissipation**. We built a PYTHON analytical and parsing framework for μ Prof charts using JUPYTER NOTEBOOKS.

4.2. A power governance framework

Since we are not able to act on specific per-core/thread consumption metrics, we will use Linux **CPU governors as our control knob** and study their impact on our implementation.

CPU governors in Linux ultimately act on clock speeds to scale power consumption and heat generation. Among others, they also include:

- **performance:** bypasses dynamic scaling and fixes frequency to the highest value.
- **powersave:** the other extreme, statically locks the CPU to the lowest available.
- **ondemand:** traditional dynamic scaling, overlooks the CPU load at defined intervals and requests frequency jumps correspondingly.

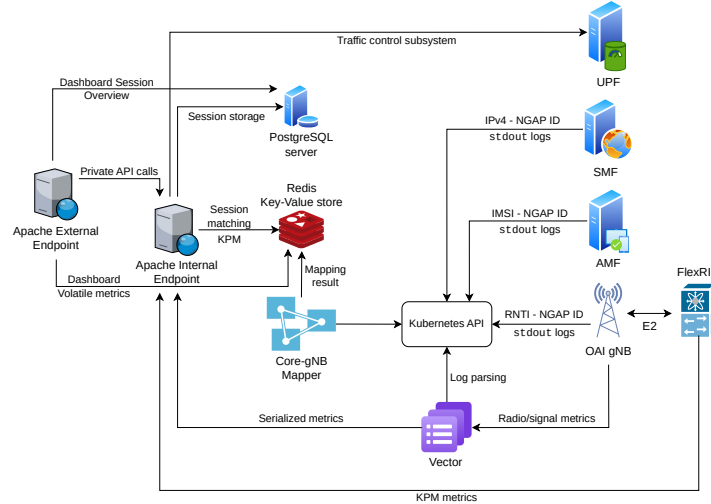
4.2.1 The staircase state machine

Our governance framework acts as a **vigilance solution energy-wise**, exploiting privileged access to the infrastructure and per-UE metrics we collected earlier. It acts, essentially as a Finite State Machine:

- Monte Carlo approach: randomly **sampling 30% of active UEs**. If any of them exceeds a **2% packet loss**, an alarm is triggered.
- Asymmetric scaling (staircase): if sampled **performance degrades**, the system adapts by taking the **CPU governor up** until the issue is resolved.
- Hardware interrupts: if **processing errors** are detected (e.g. overflows), the state is temporarily forced to the **performance governor**.

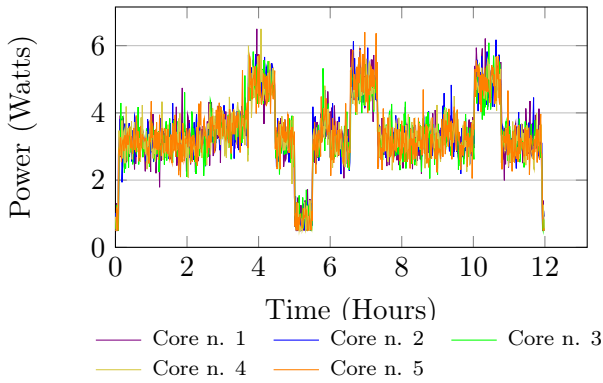
4.2.2 A complete experimental execution

We made use of our complete solution simulating **half of a normal day** operating this infrastructure as a Neutral Host.



A complete architectural overview including all the components we developed and integrated

Individual core power consumption



Power draw measurements outlines our goal: our energy controller **keeps the gNB in powersave as much as possible**, but brief external interference or packet loss are always possible.

4.3. Use-cases

- User and tenant billing
- Real-time performance and SLA monitoring
- Performance correlation
- Anomaly and abuse detection
- Spatial user mapping and positioning

5. Conclusions

In the increasingly complex landscape we live in, external pressures are driving the development of paradigmatic shifts towards transparent, efficient and more inspectable systems.

This work challenged these needs by proposing, implementing and validating an efficient and observable platform for multi-tenant Neutral Host deployments, composed of:

- **A Neutral Host deployment:** a multi-tenant environment on KUBERNETES

- **A custom and efficient Kubernetes distribution:** thanks to IaC tools, we built a transparent, auditable and efficient setup.
- **An open-source 5G stack**
- **An observability platform for Neutral Hosts:** administrators are now able to inspect and control the usage of hardware resources, analyze performance and radio metrics for their UEs and enforce traffic control limitations on them.
- **A power governance framework,** which validates the chosen implementation strategies.

6. Bibliography

- [1] Y.M. Allawi et al. A Sustainable Business Model for a Neutral Host Supporting 5G and beyond (5GB) Ultra-Dense Networks: Challenges, Directions, and Architecture. *Sensors*, 22(14), 2022.
- [2] M. Bevilacqua. EcoRAN: an energy-aware optimization framework for multi-tenant neutral host O-RAN deployments. Master's thesis, Politecnico di Milano, <https://hdl.handle.net/10589/240867>, 2025.
- [3] L. Bonati et al. NeutRAN: An Open RAN Neutral Host Architecture for Zero-Touch RAN and Spectrum Sharing. *IEEE Transactions on Mobile Computing*, 23(5):5786–5798, 2024.
- [4] H. Cheng et al. ORANSlice: An Open Source 5G Network Slicing Platform for O-RAN. In *Proceedings of the 30th Annual International Conference on Mobile Computing and Networking*, ACM MobiCom '24, New York, 2024.