

Benchmarking Methodology Working Group
Internet-Draft
Intended status: Informational
Expires: 23 April 2026

T. Samizadeh
fortiss GmbH
G. Koukis
ATHENA RC
R. C. Sofia
fortiss GmbH
L. Mamatas
University of Macedonia
V. Tsaoussidis
ATHENA RC
20 October 2025

CNI Telco-Cloud Benchmarking Considerations
draft-samizadeh-bmwg-cni-benchmarking-01

Abstract

This document investigates benchmarking methodologies for Kubernetes Container Network Interfaces (CNIs) in Edge-to-Cloud environments. It defines performance, scalability, and observability metrics relevant to CNIs, and aligns with the goals of the IETF Benchmarking Methodology Working Group (BMWG). The document surveys current practices, introduces a repeatable benchmarking frameworks (e.g., CODEF), and proposes a path toward standardized, vendor-neutral benchmarking procedures for evaluating CNIs in microservice-oriented, distributed infrastructures.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <https://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on 23 April 2026.

Copyright Notice

Copyright (c) 2025 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<https://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Revised BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Revised BSD License.

Table of Contents

1. Introduction	3
2. Requirements Language	3
3. Problem Statement and Alignment with BMWG Goals	3
3.1. Abbreviations	5
3.2. Scope of Metrics	5
4. CNI Benchmarking Key Aspects	6
4.1. Core Performance Metrics for CNI Benchmarking	6
4.1.1. Data Plane Performance Metrics	6
4.1.2. Control Plane Performance Metrics	8
4.1.3. System Resource Performance Metrics	8
4.2. Extended Performance Metrics (Optional)	9
4.3. Extended Quality of Experience for DevOps and Developers (Optional)	10
4.4. Interoperability and Scalability	11
4.5. Observability and Bottleneck Detection	11
4.6. Kubernetes CNI topologies	12
5. CNI Behavior in Federated and Multi-Cluster Environments	12
5.1. Overview of Federated Networking	12
5.2. Benchmarking Considerations for CNIs in Federated Environments	13
6. Best Practice Operational Example: CODEF	14
6.1. CODEF Benchmarking and CNI Support	16
6.2. Environment Configuration Aspects	17
6.3. Measurement Tools	18
7. Kubernetes CNI Benchmarking Telco-Cloud Methodology	18
7.1. Controlled Test Environments	19
7.2. Standardized Test Configurations	19
7.3. Test Repeatability and Statistical Significance	19
7.4. Traffic Generators, Traffic Models and Load Profiles	20
7.5. Workload Simulation, Emulation, and Stress Testing	20
7.6. Observability and Resource Instrumentation	21
7.7. Result Reporting and Output Format	21

8. IANA Considerations	21
9. Security Considerations	21
10. References	21
10.1. Normative References	22
10.2. Informative References	23
Acknowledgements	27
Appendix A. Change Log	27
Authors' Addresses	27

1. Introduction

This document presents an initial exploration of benchmarking methodologies for Kubernetes Container Network Interfaces (CNIs) in Edge-to-Cloud environments. It evaluates the performance characteristics of common Kubernetes networking plugins such as Multus, Calico, Cilium, and Flannel within the scope of container orchestration platforms. The draft aims to align with the principles of the IETF Benchmarking Methodology Working Group (BMWG) by proposing a framework for repeatable, comparable, and vendor-neutral benchmarking of CNIs. Emphasis is placed on performance aspects relevant to Software Defined Networking (SDN) architectures and distributed deployments. The goal is to inform the development of formal benchmarking procedures tailored to CNIs in heterogeneous infrastructure scenarios.

2. Requirements Language

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

3. Problem Statement and Alignment with BMWG Goals

BMWG proposes and debates methodologies and metrics to evaluate performance characteristics of networking devices and systems in a repeatable, vendor-neutral, and interoperable manner. While multiple Kubernetes CNI solutions exist and are critical to Kubernetes networking-and by extension, to telco-cloud networking-there is currently no standardized methodology for benchmarking their performance, resource utilization, or behavior under varying operational conditions. The absence of such standards leads to non-reproducible, vendor-specific results that are difficult to compare or rely on for deployment decisions in edge-cloud contexts.

This document aligns with BMWG goals by proposing benchmarking considerations for Kubernetes Container Network Interface (CNI) plugins that adhere to the following principles:

- * **Repeatability and Reproducibility:** The draft emphasizes deterministic test environments by leveraging clean-slate container orchestration through automation frameworks such as the experimental open-source Cognitive Decentralised Edge Cloud (CODECO) [codeco_d10] and the Experimentation Framework (CODEF) [codef]. Test cases are repeatable across deployments, and variability in underlying infrastructure (e.g., bare metal vs. virtualized environments) is explicitly documented to preserve reproducibility, following BMWG best practices [RFC2544] and [RFC7312].
- * **Vendor-Neutral Evaluation:** The proposed approach includes a diverse set of CNIs from multiple vendors and open-source communities, avoiding platform-specific optimizations. CNIs are evaluated under the same environmental and workload conditions to provide fair comparisons, consistent with BMWG's commitment to vendor-agnostic test procedures.
- * **Metrics-Based Assessment:** The document adopts classical benchmarking metrics including latency, throughput, jitter, and resource consumption (CPU, memory), extending them with CNI-relevant attributes such as pod network initialization time and observability overhead. These metrics are aligned with performance evaluation goals outlined in [RFC1242], [RFC2285], and more recent benchmarking efforts for virtualized environments [RFC8172].
- * **Applicability to Emerging Architectures:** The targeted environment includes Edge-to-Cloud deployments, which represent modern distributed system architectures. While BMWG has historically focused on network appliances, this work extends those principles to the networking aspects of containerized and software-defined infrastructures, continuing the evolution of benchmarking methods to address dynamic, microservice-based platforms.
- * **Traffic and Control Plane Separation:** Following BMWG precedent (e.g., [RFC6808]), the methodology distinguishes between control-plane operations (e.g., pod deployment and CNI setup latency) and data-plane behavior (e.g., packet forwarding performance), allowing comprehensive benchmarking of CNIs across operational dimensions.

- * Scalability and Stress Testing: The methodology incorporates stress and scalability scenarios, consistent with goals in [ietf-bmwg-07], to uncover performance degradation points and assess operational resilience of CNIs under heavy load and fault conditions.
- * Model Reference: CNIs in Kubernetes follow the models described in [RFC6808].

This alignment ensures that future extensions of this document toward a formal benchmarking specification can be scoped within the BMWG charter and contribute to standardized practices for container network evaluation.

3.1. Abbreviations

- * CNI: Container Network Interface
- * SUT: System Under Test
- * DUT: Device Under Test
- * SDN: Software Defined Networking
- * OVS: Open vSwitch
- * OVN: Open Virtual Network
- * RTT: Round-Trip Time
- * eBPF: Extended Berkeley Packet Filter
- * ENI: Elastic Network Interface
- * QoE: Quality of Experience

3.2. Scope of Metrics

The core benchmarking metrics in this document, such as latency, throughput, jitter, packet loss, and pod lifecycle time, are aligned with BMWG practices. Additional metrics such as resource usage, energy efficiency, and operational ease are included to reflect real-world operator concerns but are considered informational and outside the core BMWG scope.

4. CNI Benchmarking Key Aspects

While several performance-benchmarking suites are already available from CNI providers [cilium-bench], the open-source community [TNSM21-cni], and also in the IETF BMWG [ietf-bmwg-07], a comprehensive CNI evaluation SHOULD incorporate relevant performance metrics, scalability aspects and identify bottlenecks. This section provides a view on relevant aspects to ensure reliable and replicable performance evaluation, considering aspects that are relevant from a telco-cloud perspective.

4.1. Core Performance Metrics for CNI Benchmarking

Considering the architecture of microservice-based applications, microservices may interact with each other and external services. Having containerized applications and orchestration platforms like Kubernetes, there is a continuous need to address communication and networking as Kubernetes doesn't handle networking itself. Moreover, communication between containers is extremely important to meet QoS requirements of applications. To evaluate the performance of CNIs there are several metrics that should be taken into account including network throughput, end-to-end latency, pod setup and deletion times, CPU and Memory utilization, etc. This section defines the core benchmarking metrics used to assess the performance of Container Network Interface (CNI) plugins in Kubernetes environments. The metrics conform to the standard benchmarking framework set forth in [RFC2544], [RFC1242], [RFC8172], and are extended where necessary to include container-specific control-plane considerations. Measurements MUST be conducted under controlled conditions as described in Section 8, and SHOULD include both steady-state and dynamic workloads.

4.1.1. Data Plane Performance Metrics

Benchmarking Quality of Service (QoS) for CNI plugins typically focuses on traditional performance metrics such as one-way latency, round-trip delay, packet loss, jitter, and achievable data rates under varied network conditions. These metrics are fundamental to assessing the efficiency and responsiveness of a CNI in both intra-cluster and inter-cluster communication scenarios. To ensure comprehensive evaluation, the benchmarking methodology SHOULD include tests using multiple transport protocols, primarily TCP and UDP. This is essential, as CNI plugins may exhibit significantly different performance profiles depending on the protocol type due to variations in connection setup, flow control, and packet processing overhead. For TCP, two key test modes are RECOMMENDED:

- * TCP_RR (Request/Response): Measures the rate at which application-layer request/response pairs can be exchanged over a persistent TCP connection. This reflects transaction latency under connection reuse scenarios.
- * TCP_CRR (Connect/Request/Response): Assesses the rate at which new TCP connections can be established, used for a request/response exchange, and torn down. This test exposes connection setup overhead and potential scalability bottlenecks.

For UDP, the benchmark SHOULD include UDP_RR testing, which captures round-trip time (RTT), latency variation (jitter), and packet loss characteristics under lightweight, connectionless exchanges. In all tests, the benchmarking suite MUST include a representative range of payload sizes, including at least 64 bytes, 512 bytes, and 1500 bytes. If supported by the underlying network and CNI plugin, jumbo frames (e.g., MTU > 1500 bytes) SHOULD also be tested to expose potential fragmentation penalties and their impact on latency, jitter, and throughput. These metrics evaluate the efficiency of packet forwarding and transport under varying traffic patterns, and are REQUIRED:

- * One-Way Latency (ms) SHOULD be measured using timestamped probes [RFC1242].
- * RTT (ms) SHOULD be measured via TCP_RR, TCP_CRR, and UDP_RR test modes. [RFC2544].
- * Throughput (Mbps or Gbps) SHOULD be assessed via the highest sustained rate of successful packet delivery for the CNI without packet loss [RFC2544].
- * Packet loss rate (%) SHOULD be considered for reliability and congestion tolerance of the CNI [RFC2544].
- * Jitter MAY be relevant to assess variability. High jitter may indicate queuing inefficiencies or variable path latency [RFC5481].
- * Packet size variability SHALL be evaluated using a representative set of frame sizes (64B, 512B, 1500B). If jumbo frames (>1500B) are supported, testing MUST include these cases to expose fragmentation overheads [RFC2544].
- * Concurrent flow handling SHOULD be measured using concurrent connections and sustained request/response patterns for both TCP and UDP [RFC2285].

4.1.2. Control Plane Performance Metrics

These metrics evaluate the responsiveness of the CNI plugin and Kubernetes components during pod and network lifecycle operations and are REQUIRED:

- * Pod initialization time (s) SHOULD be measured from kubelet interaction to completion of CNI ADD operation [RFC8172].
- * Pod deletion time (s) SHOULD be measured to understand issues with tear down [RFC8172].
- * CNI plugin deployment time (s) SHOULD be assessed, to understand the duration required for each CNI plugin to be fully deployed across the whole network (cluster nodes).

4.1.3. System Resource Performance Metrics

These metrics are essential in resource-constrained environments (e.g., edge deployments) where efficiency impacts scalability and are RECOMMENDED:

- * CPU/GPU utilization SHOULD be reported per node and per CNI process [RFC8172].
- * Memory utilization (MB/GB) measurements MUST consider average and peak memory used by the CNI [RFC8172].
- * CNIs SHOULD be evaluated under varying load conditions (idle, low-traffic, high traffic).

The CPU and memory footprint of a Container Network Interface (CNI) plugin has substantial implications for workload density and system scalability, especially in resource-constrained or heterogeneous environments. In modern Edge-to-Cloud deployments often comprising diverse processor architectures (e.g., ARM64, AMD64) and variable memory constraints resource efficiency is critical to maximizing node utilization and sustaining performance. The architectural design of a CNI directly affects its resource profile. CNIs with extensive feature sets and complex data-plane capabilities such as policy enforcement, encryption, overlay encapsulation (e.g., VXLAN, IP-in-IP), or eBPF/XDP acceleration tend to exhibit higher CPU and memory consumption. For example, CNIs that perform user-space packet processing typically incur higher overhead, as each packet traverses the kernel-user boundary multiple times, resulting in increased CPU cycles and memory copies [RFC8172]. In contrast, in-kernel eBPF-based processing can reduce such overhead by executing directly in the Linux kernel [RFC9315]. In cloud-native deployments, CNIs that

manage external interfaces (e.g., Elastic Network Interfaces (ENIs) in public cloud environments) may also introduce persistent memory usage due to API caching, state tracking, and metadata management [aws-vpc-cni-docs]. These variabilities are further amplified under dynamic workloads. It is frequently observed that a CNI optimized for high-throughput TCP bulk traffic may perform suboptimally under UDP-heavy traffic, high pod churn, or policy-intensive workloads. These behavioral differences necessitate a systematic and multi-dimensional benchmarking approach. Accordingly, a robust benchmarking methodology SHOULD assess each CNI under at least three operating states: idle, low-traffic (and low load), high traffic (and high load). Such profiling enables the identification of baseline resource usage, saturation thresholds, and degradation points ("performance peaks"). Measurements SHOULD be taken at both the node level (e.g., using Prometheus [prometheus-docs]) and at the container or pod level (e.g., using cAdvisor [cadvisor-docs]). These practices are consistent with recommendations for virtualized and cloud-native benchmarking environments as described in [RFC8172].

4.2. Extended Performance Metrics (Optional)

While outside the core BMWG scope, these metrics reflect real-world operator needs and may be included for extended analysis, in particular for edge-cloud heterogeneous and resource constrained scenarios. As such, the following metrics are RECOMMENDED:

- * Policy enforcement delay (ms)
- * Telemetry overhead.
- * Power and energy consumption (J per bit). Where applicable, node- or pod-level energy usage MAY be reported using tools such as Kepler . Results SHOULD include error margins due to estimation variance, or energy models.

While not core to BMWG benchmarking, and currently non-normative, energy metrics MAY be collected where relevant. Tools such as Kepler MAY be used, but results SHOULD be accompanied by a disclaimer about accuracy limitations in virtualized environments, and also on issues related with the applied energy models. A related discussion on energy metrics and energy-sensitivity can be found in IETF GREEN, [draft-ea-ds], and in the IRTF NMRG [I-D.irtf-nmrg-energy-aware], as well as in IRTF SUSTAIN.

4.3. Extended Quality of Experience for DevOps and Developers (Optional)

Quality of Experience (QoE) benchmarking for Container Network Interface (CNI) plugins extends beyond conventional network performance metrics such as latency and throughput. It focuses on assessing operational usability, deployment efficiency, and portability, i.e., factors that directly affect the user experience of platform administrators, DevOps engineers, and developers. For instance, time to deploy or configure the CNI, ease of troubleshooting, and impact of the CNI on application performance are examples of QoE parameters. Key QoE indicators OPTIONAL MAY include:

- * Deployment time, the time required to install or upgrade a CNI plugin using declarative tooling (e.g., Helm charts, YAML manifests).
- * Configuration simplicity, the extent to which configuration is automated, validated, and integrated with Kubernetes-native workflows.
- * Troubleshooting tooling, the presence of purpose-built CLI utilities that simplify diagnostics, expose internal CNI state, and reduce reliance on low-level log inspection or manual `kubectl` commands.

For example, CNI-specific command-line interfaces such as `cilium` and `calicoctl` provide capabilities such as one-command installation, real-time policy and connectivity status, and automated diagnostics. The `cilium status --verbose` command provides IPAM allocations, agent health, and datapath metrics, while the `calicoctl node diags` generates complete diagnostic bundles for analysis. CNI integration with Kubernetes distribution CLIs (e.g., `k3s`, `MicroK8s`) further improves QoE by streamlining lifecycle operations. For instance, `MicroK8s` leverages snap-based add-ons that can enable or disable CNIs via a single command, reducing complexity and configuration drift. Although these attributes are not part of the core benchmarking metrics defined by BMWG, their inclusion is RECOMMENDED to reflect practical DevOps concerns and enhance the applicability of CNI benchmarking results in production environments.

4.4. Interoperability and Scalability

To ensure comprehensive benchmarking coverage, scalability and stress-testing phases SHOULD be incorporated into the evaluation methodology. These phases are essential to identify the performance ceilings of a given CNI plugin and to assess its behavior under saturation conditions, including whether key observability features remain functional. Such assessments are consistent with guidance outlined in [RFC8239] and extend benchmarking scope beyond nominal operation to failure and recovery modes. Stress tests SHOULD simulate high-load scenarios by concurrently scaling multiple Kubernetes components. This includes initiating rapid pod-creation bursts, deploying multiple concurrent services and network policies, and triggering controlled resource exhaustion events (e.g., CPU throttling, memory pressure, disk I/O contention). Furthermore, network issues such as increased latency, jitter, or packet loss SHOULD be introduced using tools like [tc-netem] to assess the CNI's robustness under adverse network conditions. The use of orchestration tools such as Kube-Burner [kube-burner] and chaos engineering frameworks (e.g., Chaos Mesh or Litmus) is RECOMMENDED to coordinate scalable and repeatable test scenarios. Network performance metrics during stress tests MAY be collected with traffic generators such as iperf3, netperf, or k6 [iperf3] [k6]. Benchmark results SHOULD include degradation thresholds, error rates, recovery latency, and metrics export consistency under stress to support the evaluation of CNI resilience and operational observability.

4.5. Observability and Bottleneck Detection

Observability is critical in identifying performance bottlenecks that may arise due to CNI behavior under stress conditions. Benchmarking SHOULD assess the ability of CNIs to expose metrics such as packet drops, queue lengths, or flow counts through standard telemetry interfaces (e.g., Prometheus, OpenTelemetry). Effective bottleneck detection tools and visibility into the data path are essential for root cause analysis. CNIs that provide native observability tooling (e.g., Cilium Hubble) SHOULD be benchmarked for the overhead and fidelity of these features. In federated or multi-cluster environments, observability becomes a distributed operation spanning multiple control and data planes. Benchmarking MUST therefore evaluate how CNIs and associated telemetry systems aggregate, synchronize, and correlate metrics across clusters. This includes measuring propagation delays, timestamp alignment, and aggregation accuracy when telemetry data flow through federated collectors/monitoring backends (e.g., Prometheus-Thanos, Cortex). Benchmarks SHOULD also assess the ability to localize inter-cluster bottlenecks such as congested tunnels, gateway saturation, or asymmetric routing, distinguishing local clusters from cross-cluster traffic degradation.

4.6. Kubernetes CNI topologies

Kubernetes CNI topologies refers to patterns of network connectivity in a Kubernetes environment used for testing or benchmarking CNIs [Kubernetes-docs].

- * Highly-coupled container-to-container communications
- * Pod-to-Pod communications
- * Pod-to-Service communications
- * External-to-Service communications

The benchmarking network topology must operate as an isolated test environment and MUST NOT connect to any devices that could forward test traffic into a production network or incorrectly route it to the test management network [RFC8456] and [RFC8204].

5. CNI Behavior in Federated and Multi-Cluster Environments

While existing works such as [RFC8172] and [ietf-bmwg-07] provide benchmarking methodologies for virtualized and containerized infrastructures, their scope does not extend to CNI behavior in multi-cluster or federated deployments. Architectural drafts like [draft-dwon-t2trg-multiedge-arch], [draft-si-service-mesh-dta], and [draft-ietf-wimse-workload-identity-practices] discuss aspects of multi-cluster operations and security, but do not specify CNI-focused, measurable performance parameters and considerations. Similarly, [draft-contreras-nmrg-interconnection-intents] introduces the notion of multi-cluster service deployment and intent-based interconnection, yet it does not cover CNI-level performance benchmarking across federated clusters.

5.1. Overview of Federated Networking

Federated and multi-cluster environments extend the scope of container networking beyond single operational domains. These architectures enable scalability, geographical distribution, isolation, and service proximity to end users, which are key properties for multi-domain cloud-native infrastructures. Federated CNI benchmarking is particularly relevant to Telco-Cloud and 6G scenarios, where workloads are distributed between cloud and (far-)edge IoT domains, introducing additional considerations compared to single-cluster deployments. In such environments, multiple clusters operate as autonomous domains while being interconnected through federation layers or multi-cluster networking mechanisms. Examples include popular third-party solutions such as

Submariner, Ligo, Karmada, and Open Cluster Management (OCM), which provide network connectivity, service discovery, and workload scheduling across clusters. In this context, CNIs are often extended by multi-cluster gateways or overlays to facilitate inter-cluster pod-to-pod and service-to-service communication. Such interconnections can rely on encapsulation protocols (e.g., VXLAN, IPSec, WireGuard) or Layer-7 service meshes (e.g., Istio, Linkerd, Consul, Open Service Mesh) - based on Envoy proxy and sidecars.

5.2. Benchmarking Considerations for CNIs in Federated Environments

Benchmarking CNIs in federated deployments MUST explicitly reflect how (i) architectural choices, (ii) topology and connectivity, (iii) overlay and tunneling mechanisms, (iv) synchronization, and (v) security enforcement affect network behavior for both (i) data-plane and (ii) control-plane operations. The following factors are key:

- * **Federation and Topology Models:** CNIs may operate under hub-and-spoke [RFC4364], [RFC7024], neighboring, full-mesh [RFC4271], [RFC9181], or hierarchical [RFC7426] topologies. Each model introduces distinct path lengths and potential bottlenecks and security concerns. Benchmarks SHOULD quantify metrics like latency, jitter, and packet loss across these models.
- * **Overlay, Encapsulation, and Encryption Mechanisms:** CNIs may rely on native multi-cluster extensions (e.g., Cilium ClusterMesh) or external overlays (e.g., Submariner tunnels) with optional encryption (e.g., IPSec, WireGuard). Tests SHOULD measure the combined encapsulation and cryptographic overhead, including per-packet header size, MTU effects, CPU utilization, and throughput reduction compared to unencrypted baselines.
- * **Routing, Policy, and Synchronization Behavior:** CNIs synchronize endpoints, routes, and network policies across clusters. Benchmarking SHOULD measure propagation delay, convergence time, and consistency under dynamic conditions such as node joins, removals, or policy updates. Resource utilization (CPU, memory, and bandwidth) during synchronization SHOULD also be recorded.
- * **Cross-Cluster Connectivity and Load Balancing:** Evaluation SHOULD include one-way and RTT latency, throughput, and packet loss between pods located in different clusters. When multi-cluster services distribute requests, benchmarks SHOULD assess fairness as well as responsiveness to endpoint or cluster failures that influence path selection and recovery behavior.

- * **Quality of Service (QoS) and Policy Enforcement:** CNIs that implement QoS tagging or traffic shaping (e.g., Cilium's eBPF/EDT-based pacing, Calico's DSCP marking and policy-driven shaping, Antrea's TrafficControl, or Kube-OVN's QoS queues) SHOULD be evaluated for their ability to maintain SLA/SLO across clusters and overlays. Benchmarks SHOULD also verify that isolation and access-control policies (e.g., deny/allow rules) remain consistent across domains.
- * **Resiliency and Recovery Performance:** Benchmarking SHOULD assess CNI behavior during multi-cluster fault conditions, including inter-cluster link loss, control-plane failures, restarts, or topology reconfiguration. Measurements SHOULD include reconvergence time, packet loss, and recovery time to steady-state. Benchmarks SHOULD also evaluate route re-establishment latency and transient traffic interruption duration to characterize the CNI's overall fault-tolerance behavior.

6. Best Practice Operational Example: CODEF

CODEF is an open-source, modular benchmarking environment that supports the evaluation of containerized workloads in edge-to-cloud infrastructures. CODEF adopts a microservice-based architecture to streamline experimentation through abstraction, automation, and reproducibility. CODEF is logically divided into four functional layers, each implemented as an independent containerized microservice: Infrastructure Manager, Resource Manager, Experiment Controller, and Results' Processor, as represented in Figure 1. This modular design ensures extensibility and facilitates integration with diverse technologies across the experimentation pipeline.

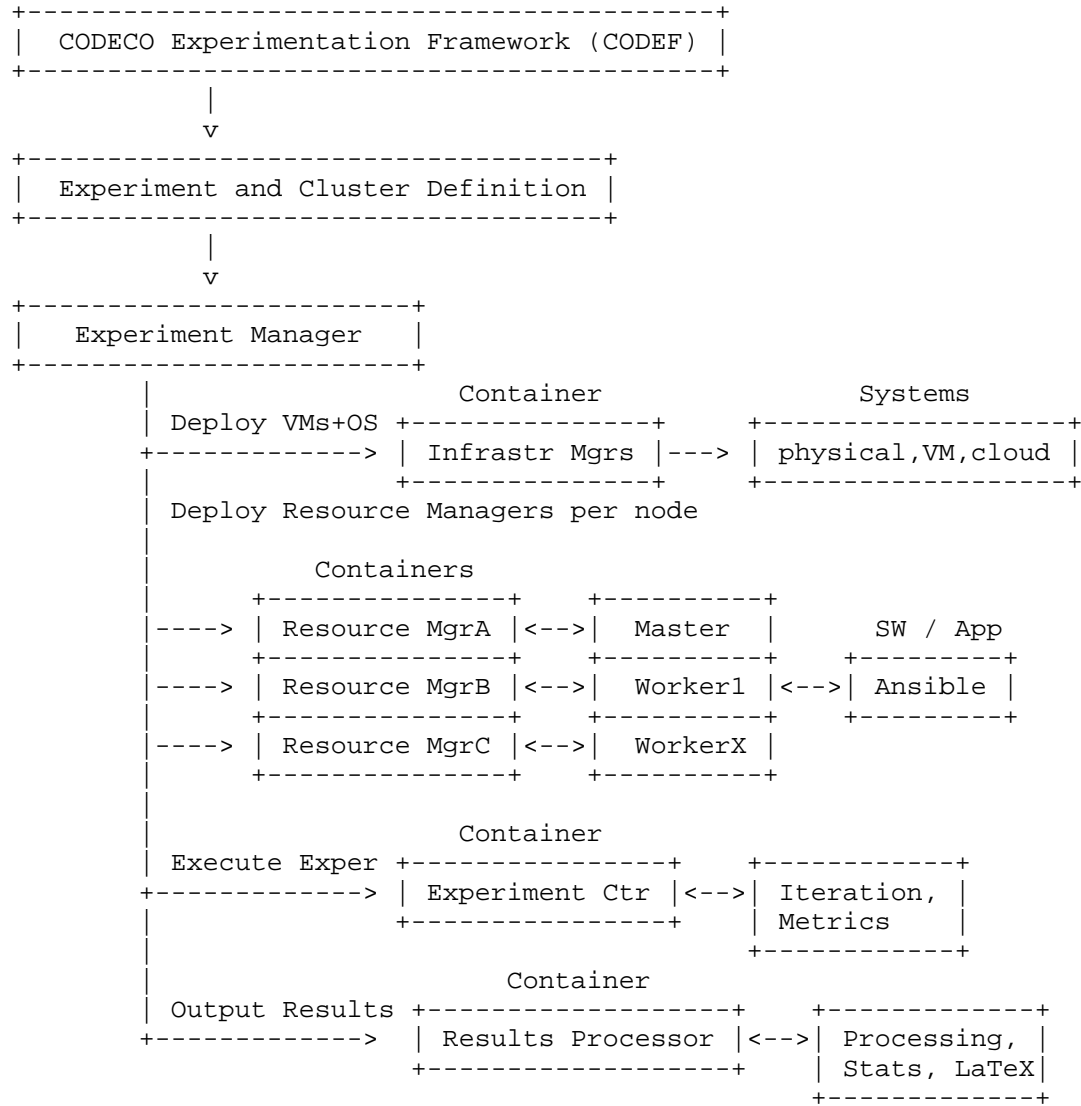


Figure 1: CODEF and its components.

- * The Infrastructure Manager layer provisions cluster resources across heterogeneous environments, including bare-metal nodes, hypervisor-based virtual machines (e.g., VirtualBox, XCP-ng), and public or academic cloud testbeds (e.g., AWS, CloudLab, EdgeNet).

- * The Resource Manager deploys software components on each node using parameterized Ansible playbooks. A dedicated instance of the Resource Manager operates per node to guarantee consistent, automated software setup.
- * The Experiment Controller coordinates workload execution, manages experimental iterations, collects measurement data, and invokes benchmarks.
- * The Results' Processor performs statistical analysis and post-processing to generate structured outputs, including visualization and reporting artifacts.

CODEF supports full automation of the experimentation lifecycle, from cluster instantiation to metric analysis. Each cluster is provisioned from clean operating system images to ensure consistency, repeatability, and environmental isolation across benchmark runs. This approach eliminates state leakage between tests and enhances comparability. The framework also provides low-level parameterization options for various networking and security configurations. These include tunneling and encapsulation mechanisms (e.g., VXLAN, Geneve, IP-in-IP), encryption protocols (e.g., IPsec, WireGuard), and Linux kernel-based datapath acceleration features (e.g., eBPF and XDP). Such flexibility supports the emulation of production-grade deployments across a wide range of container network interfaces (CNIs) and infrastructure types.

6.1. CODEF Benchmarking and CNI Support

CODEF addresses the need for repeatable, infrastructure-agnostic benchmarking across the edge-to-cloud continuum. It supports a broad spectrum of third-party CNIs plugins, including Antrea [antrea], Calico [calico], Cilium [cilium], Flannel [flannel], Weave Net [weavenet], Kube-Router [kube-router], Kube-OVN [kube-ovn], and Multus, as well as emerging solutions such as L2S-M [L2S-M]. These CNIs can be deployed and benchmarked across multiple Kubernetes distributions, including upstream Kubernetes (vanilla), lightweight variants such as K3s, K0s, and MicroK8s, and production-grade clusters. Each CNI plugin employs distinct architectural strategies at the network layer, such as underlay versus overlay models, use of encapsulation protocols (e.g., VXLAN, Geneve), encryption mechanisms (e.g., WireGuard, IPsec), and programmable datapaths (e.g., eBPF/XDP). Additionally, the degree of support for network policy enforcement, observability, and integration with Kubernetes-native APIs varies significantly across implementations. These differences introduce variability in performance, scalability, and resource utilization depending on workload and deployment characteristics. CODEF enables the consistent application of benchmarking procedures

across this heterogeneity by offering a unified, declarative methodology. It abstracts infrastructure-specific details and enforces environmental consistency through repeatable provisioning, workload orchestration, and result normalization. Accordingly, any benchmarking methodology targeting CNIs in diverse Kubernetes environments SHOULD account for these dimensions: CNI architecture, Kubernetes distribution, infrastructure type, and test scenario configuration to ensure meaningful, comparable, and reproducible results.

6.2. Environment Configuration Aspects

In addition to the functional differences among CNI plugin implementations, benchmarking methodologies SHOULD account for the architectural and physical characteristics of the deployment environment. Key variables include the type of infrastructure such as virtualized environments (e.g., VM or hypervisor-based) versus bare-metal deployments and the test topology, including intra-node (same host) versus inter-node (across hosts) communication. Benchmarks SHOULD also distinguish between distributions designed for general-purpose Kubernetes (e.g., vanilla K8s) and those optimized for constrained edge deployments (e.g., MicroK8s, K3s). Hardware heterogeneity introduces further variability. Performance results can be significantly influenced by CPU architecture (e.g., x86_64 vs. ARM), number of cores and threads, memory speed and hierarchy, cache layout, NUMA topology, and network interface characteristics (e.g., NIC model, offload capabilities, and firmware version). Low-level system configuration options, including MTU size, tunneling mode (e.g., VXLAN, IP-in-IP), and kernel datapath tuning (e.g., eBPF or XDP parameters), MAY also affect observed performance. Empirical results from experiments conducted with CODEF under a variety of scenarios including intra- and inter-cluster configurations, hardware with diverse specifications, and a range of Kubernetes distributions demonstrated measurable performance differences across CNI plugins. Notably, significant disparities were observed not only between different CNI implementations, but also within the same CNI when deployed on different Kubernetes distributions or system architectures. Contrary to expectation, deploying lightweight CNI plugins on edge-optimized distributions does not always result in improved efficiency. In some cases, plugins reduce their resource footprint by sacrificing performance (e.g., selecting a simpler encapsulation mechanism), while others achieve better throughput when paired with more capable general-purpose distributions at the expense of increased overhead. These trade-offs SHOULD be explicitly captured in benchmarking outcomes. Importantly, the optimal CNI and distribution pairing is often workload-dependent. A configuration that appears suboptimal in terms of raw resource usage MAY outperform a lightweight alternative for certain traffic patterns, application

behaviors, or network policies. As such, benchmarking methodologies intended for heterogeneous edge-cloud scenarios, in particular mobile scenarios and IoT scenarios, where embedded devices are a main part of the overall networking infrastructure, SHOULD incorporate these dimensions and evaluate plugin behavior across representative workloads and system conditions.

6.3. Measurement Tools

CODEF relies on Ansible playbooks to provision a suite of software tools supporting both workload generation and measurement. Benchmarking configurations may include lightweight and comprehensive traffic generators such as [iperf3], [netperf], and [sockperf], as well as the [k8s-bench-suite]. These tools enable detailed measurements of network bandwidth, packet throughput, latency, and fragmentation behavior across TCP and UDP protocols, with varying message sizes. Resource usage metrics such as CPU load, memory consumption, and disk utilization are collected at both node and container granularity. Observability stacks based on Prometheus and Grafana are integrated for real-time metric capture, historical trend visualization, and alerting capabilities. These facilities support traceability of system behavior during experiments and assist in identifying anomalous performance characteristics. For scalability and resilience benchmarking, CODEF integrates load and stress testing tools such as the CNCF [kube-burner] and chaos engineering platforms (e.g., Chaos Mesh or Litmus). These tools simulate dynamic workloads, rapid pod scaling, and fault injection to evaluate system performance under adverse or bursty conditions. Such orchestrated testing scenarios are essential to reveal bottlenecks, performance degradation points, and recovery latency under operational stress. Power consumption profiling is optionally supported through empirical estimation models or telemetry-based measurement frameworks such as [kepler]. However, their accuracy SHOULD be evaluated critically, as results may vary depending on the availability and quality of hardware-level counters (e.g., Intel RAPL) and the characteristics of the execution platform, particularly in virtualized or non-Intel environments.

7. Kubernetes CNI Benchmarking Telco-Cloud Methodology

This section defines a set of best practice guidelines for benchmarking Kubernetes CNI plugins in telco-cloud and edge-cloud environments. The approach is aligned with IETF BMWG, emphasizing reproducibility, transparency, comparability. The benchmarking recommendations presented herein aim to be applicable across a wide range of deployment scenarios, Kubernetes distributions, and CNI implementations. While selected operational workflows and experiences from CODEF are considered to illustrate practical

implementation of these best practices, the methodology itself is designed to remain tool-agnostic and aligned with standardized benchmarking guidance. The practices focus on controlled environment setup, test repeatability, performance metric collection, observability, and result reporting. Attention is given to relevant characteristics for telco and edge environments, including resource constraints, deployment diversity, and protocol behavior under stress. The goal is to provide a consistent and extensible benchmarking methodology for CNIs operating in dynamic, distributed, and microservice-oriented infrastructure environments.

7.1. Controlled Test Environments

Benchmarking SHOULD be conducted in isolated testbeds with no extraneous traffic or workloads. The following practices help reduce environmental noise and increase determinism:

- * Use bare-metal or dedicated VMs for benchmarking to avoid cross-tenant interference.
- * Ensure consistent CPU pinning and disable power-saving features or CPU frequency scaling to stabilize performance measurements.
- * Synchronize clocks across test nodes using NTP or PTP for accurate latency and jitter measurement.

7.2. Standardized Test Configurations

Benchmarking SHOULD adhere to pre-defined configurations to enable comparability across CNIs and platforms, aligning with [RFC2544][RFC6815]. The following elements MUST be documented:

- * Kubernetes version and distribution.
- * CNI plugin version and configuration parameters.
- * Kernel version and system tunables (e.g., MTU size, sysctl options).
- * CPU model, memory size, and network interface type.

7.3. Test Repeatability and Statistical Significance

Each experiment SHOULD be repeated a minimum of five times. For latency and throughput metrics, results MUST be reported using:

- * Minimum, average (median), maximum.

- * at least 90th, and 95th percentile values.

Furthermore, adequate warm-up times when starting test runs, and cool-down periods between test runs SHOULD be included to prevent thermal bias or residual resource contention. Where possible, automation frameworks (e.g., CODEF, Ansible) SHOULD be used to ensure that each experiment is launched from a clean state.

7.4. Traffic Generators, Traffic Models and Load Profiles

Traffic generators MUST support multiple transport protocols (e.g., TCP, UDP) and varying packet sizes as well as interrarrival packet rates. Benchmarking tools such as iperf3, netperf, and sockperf are RECOMMENDED. For realistic CNI evaluation:

- * TCP_RR, TCP_CRR, and UDP_RR SHOULD be used to measure latency, jitter, and throughput.
- * Multiple flows and concurrent connections SHOULD be tested to simulate microservice interactions.

Benchmarks SHOULD include traffic profiles reflecting real-world microservice communications, such as:

- * Short-lived TCP connections (request/response).
- * Persistent streaming (large payloads, high throughput).
- * Burst UDP traffic for latency and packet loss analysis.

7.5. Workload Simulation, Emulation, and Stress Testing

To evaluate performance under real-world loads, benchmarking MUST include scenarios with:

- * Small, average, high pod churn rates (creation/deletion).
- * Concurrent service access and policy enforcement.
- * Synthetic network and node failure

Tools such as kube-burner, chaos-mesh, and tc-netem are RECOMMENDED to orchestrate these scenarios, aligning with stress test guidance in [RFC8239].

7.6. Observability and Resource Instrumentation

CNIs SHOULD expose internal metrics (e.g., policy hits, flow counts, packet drops). Benchmarks MUST capture:

- * CPU and memory usage per CNI pod/process via for instance Prometheus.
- * NIC statistics.
- * Network path visibility (e.g., using Cilium Hubble or Calico flow logs)

Experimental and open-source examples on how such metrics can be captured at a node and network level can be checked in the CODECO project [codeco_d10] and respective code [codeco_d12]. Resource metrics MUST be collected at both node-level and pod-level granularity.

7.7. Result Reporting and Output Format

Benchmarking outputs SHOULD:

- * Use machine-readable formats (e.g., JSON, YAML, YANG).
- * Clearly label all test parameters and metrics.
- * Include system logs, configuration manifests, and tool versions.

A common results schema SHOULD be developed to support comparative analysis and long-term reproducibility, in line with goals in [RFC6815].

8. IANA Considerations

This document has no IANA considerations.

9. Security Considerations

Benchmarking tools and automation frameworks may introduce risk vectors such as elevated container privileges or misconfigured network policies. Experiments involving stress tests or fault injection should be performed in isolated environments. Benchmarking outputs SHOULD NOT expose sensitive cluster configuration or node-level details.

10. References

10.1. Normative References

- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<https://www.rfc-editor.org/info/rfc2119>>.
- [RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <<https://www.rfc-editor.org/info/rfc8174>>.
- [RFC7312] Fabini, J. and A. Morton, "Advanced Stream and Sampling Framework for IP Performance Metrics (IPPM)", RFC 7312, DOI 10.17487/RFC7312, August 2014, <<https://www.rfc-editor.org/info/rfc7312>>.
- [RFC2285] Mandeville, R., "Benchmarking Terminology for LAN Switching Devices", RFC 2285, DOI 10.17487/RFC2285, February 1998, <<https://www.rfc-editor.org/info/rfc2285>>.
- [RFC2544] Bradner, S. and J. McQuaid, "Benchmarking Methodology for Network Interconnect Devices", RFC 2544, DOI 10.17487/RFC2544, March 1999, <<https://www.rfc-editor.org/info/rfc2544>>.
- [RFC1242] Bradner, S., "Benchmarking Terminology for Network Interconnection Devices", RFC 1242, DOI 10.17487/RFC1242, July 1991, <<https://www.rfc-editor.org/info/rfc1242>>.
- [RFC8172] Morton, A., "Considerations for Benchmarking Virtual Network Functions and Their Infrastructure", RFC 8172, DOI 10.17487/RFC8172, July 2017, <<https://www.rfc-editor.org/info/rfc8172>>.
- [RFC6808] Ciavattone, L., Geib, R., Morton, A., and M. Wieser, "Test Plan and Results Supporting Advancement of RFC 2679 on the Standards Track", RFC 6808, DOI 10.17487/RFC6808, December 2012, <<https://www.rfc-editor.org/info/rfc6808>>.
- [RFC8239] Avramov, L. and J. Rapp, "Data Center Benchmarking Methodology", RFC 8239, DOI 10.17487/RFC8239, August 2017, <<https://www.rfc-editor.org/info/rfc8239>>.
- [RFC6815] Bradner, S., Dubray, K., McQuaid, J., and A. Morton, "Applicability Statement for RFC 2544: Use on Production Networks Considered Harmful", RFC 6815, DOI 10.17487/RFC6815, November 2012, <<https://www.rfc-editor.org/info/rfc6815>>.

- [RFC5481] Morton, A. and B. Claise, "Packet Delay Variation Applicability Statement", RFC 5481, DOI 10.17487/RFC5481, March 2009, <<https://www.rfc-editor.org/info/rfc5481>>.
- [RFC9315] Clemm, A., Ciavaglia, L., Granville, L. Z., and J. Tantsura, "Intent-Based Networking - Concepts and Definitions", RFC 9315, DOI 10.17487/RFC9315, October 2022, <<https://www.rfc-editor.org/info/rfc9315>>.
- [RFC4364] Rosen, E. and Y. Rekhter, "BGP/MPLS IP Virtual Private Networks (VPNs)", RFC 4364, DOI 10.17487/RFC4364, February 2006, <<https://www.rfc-editor.org/info/rfc4364>>.
- [RFC8204] Tahhan, M., O'Mahony, B., and A. Morton, "Benchmarking Virtual Switches in the Open Platform for NFV (OPNFV)", RFC 8204, DOI 10.17487/RFC8204, September 2017, <<https://www.rfc-editor.org/info/rfc8204>>.
- [RFC7024] Jeng, H., Uttaro, J., Jalil, L., Decraene, B., Rekhter, Y., and R. Aggarwal, "Virtual Hub-and-Spoke in BGP/MPLS VPNs", RFC 7024, DOI 10.17487/RFC7024, October 2013, <<https://www.rfc-editor.org/info/rfc7024>>.
- [RFC8456] Bhuvaneshwaran, V., Basil, A., Tassinari, M., Manral, V., and S. Banks, "Benchmarking Methodology for Software-Defined Networking (SDN) Controller Performance", RFC 8456, DOI 10.17487/RFC8456, October 2018, <<https://www.rfc-editor.org/info/rfc8456>>.
- [RFC9181] Barguil, S., Gonzalez de Dios, O., Ed., Boucadair, M., Ed., and Q. Wu, "A Common YANG Data Model for Layer 2 and Layer 3 VPNs", RFC 9181, DOI 10.17487/RFC9181, February 2022, <<https://www.rfc-editor.org/info/rfc9181>>.
- [RFC7426] Haleplidis, E., Ed., Pentikousis, K., Ed., Denazis, S., Hadi Salim, J., Meyer, D., and O. Koufopavlou, "Software-Defined Networking (SDN): Layers and Architecture Terminology", RFC 7426, DOI 10.17487/RFC7426, January 2015, <<https://www.rfc-editor.org/info/rfc7426>>.
- [RFC4271] Rekhter, Y., Ed., Li, T., Ed., and S. Hares, Ed., "A Border Gateway Protocol 4 (BGP-4)", RFC 4271, DOI 10.17487/RFC4271, January 2006, <<https://www.rfc-editor.org/info/rfc4271>>.

10.2. Informative References

- [codef] Koukis et al., G. and CODECO Consortium, "CODECO Experimental Framework", 2024, <<https://gitlab.eclipse.org/eclipse-research-labs/codeco-project/experimentation-framework-and-demonstrations/experimentation-framework>>.
- [codeco_d12] Samaras et al., G. and CODECO Consortium, "CODECO D12 - Basic Operation Components and Toolkit version 2.0.", 2024, <<https://doi.org/10.5281/zenodo.12819424>>.
- [draft-ea-ds] C. Sofia et al., R., "Energy-aware Differentiated Services (EA-DS). IETF draft draft-sofia-green-energy-aware-diffserv-00, active", 2025, <<https://datatracker.ietf.org/doc/draft-sofia-green-energy-aware-diffserv/>>.
- [codeco_d10] C. Sofia et al., R. and CODECO Consortium, "CODECO Deliverable D10: Technological Guidelines, Reference Architecture, and Open-source Ecosystem Design", CODECO D10, 2024, <<https://doi.org/10.5281/zenodo.12819444>>.
- [ietf-bmwg-07] Ngoc et al., T., "Considerations for Benchmarking Network Performance in Containerized Infrastructures, draft-ietf-bmwg-containerized-infra-07, active", 2025, <<https://datatracker.ietf.org/doc/draft-ietf-bmwg-containerized-infra/07/>>.
- [antrea] Antrea Project, "Antrea CNI", 2024, <<https://antrea.io>>.
- [calico] Tigera, Inc., "Project Calico", 2024, <<https://www.tigera.io/project-calico/>>.
- [cilium] Cilium Authors, "Cilium: eBPF-based Networking, Security, and Observability", 2024, <<https://cilium.io>>.
- [Kubernetes-docs] Kubernetes Authors, "Kubernetes Documents-Cluster Networking", 2024, <<https://kubernetes.io/docs/concepts/cluster-administration/networking/>>.
- [flannel] flannel-io, "Flannel CNI Plugin", 2024, <<https://github.com/flannel-io/flannel>>.

- [kube-ovn] Kube-OVN Project, "Kube-OVN: A Cloud-Native SDN for Kubernetes", 2024, <<https://github.com/kubeovn/kube-ovn>>.
- [kube-router] Kube-Router Community, "Kube-Router: All-in-One CNI, Service Proxy, and Network Policy", 2024, <<https://github.com/cloudnativelabs/kube-router>>.
- [weavenet] Weaveworks (archived), "Weave Net: Fast, Simple Networking for Kubernetes", 2024, <<https://github.com/weaveworks/weave>>.
- [cilium-bench] Cilium Authors, "Cilium Benchmarking Tools", 2024, <<https://docs.cilium.io/en/latest/operations/performance/>>.
- [TNSM21-cni] Koukis et al., G., "Benchmarking Kubernetes Container Network Interfaces: Methodology, Metrics, and Observations", January 2024, <<https://arxiv.org/abs/2401.07674>>.
- [aws-vpc-cni-docs] Amazon Web Services, "Amazon EKS Pod Networking with the AWS VPC CNI", 2024, <<https://docs.aws.amazon.com/eks/latest/userguide/pod-networking.html>>.
- [prometheus-docs] Prometheus Authors, "Prometheus Monitoring System Overview", 2024, <<https://prometheus.io/docs/introduction/overview/>>.
- [cadvisor-docs] Google, "cAdvisor: Container Advisor", 2024, <<https://github.com/google/cadvisor>>.
- [tc-netem] Linux Foundation, "tc-netem: Network Emulation", 2024, <<https://man7.org/linux/man-pages/man8/tc-netem.8.html>>.
- [kube-burner] Cloud-Bulldozer Project, "Kube-Burner: Kubernetes Performance and Scalability Tool", 2024, <<https://github.com/cloud-bulldozer/kube-burner>>.
- [iperf3] ESnet / Lawrence Berkeley National Lab, "iPerf3: Network Bandwidth Measurement Tool", 2024, <<https://iperf.fr/>>.

- [k6] Grafana Labs, "k6: Modern Load Testing Tool", 2024, <<https://k6.io/docs/>>.
- [L2S-M] Universidad Carlos 3 de Madrid, "L2S-M: Lightweight Layer 2 Switching for Microservice Networks", September 2023, <<https://github.com/Networks-it-uc3m/L2S-M>>.
- [netperf] Hewlett Packard Enterprise, "Netperf: Network Performance Benchmark", 2024, <<https://hewlettpackard.github.io/netperf/>>.
- [sockperf] NVIDIA Mellanox, "SockPerf: RDMA and TCP/UDP Latency Benchmark", 2024, <<https://github.com/Mellanox/sockperf>>.
- [k8s-bench-suite] CNCF CNF Test Suite, "Kubernetes Bench-Suite", 2024, <<https://github.com/cnf-testsuite/testsuite>>.
- [kepler] CNCF, "Kepler: Kubernetes-based Power Estimation and Reporting", 2024, <<https://kepler.sustainable.computing.dev/>>.
- [I-D.irtf-nmrg-energy-aware] Chiaraviglio, L., Pentikousis, K., Kutscher, D., and C. Pignataro, "Energy-Aware Networked Systems for a Sustainable Future", Work in Progress, Internet-Draft, draft-irtf-nmrg-energy-aware-04, March 2024, <<https://datatracker.ietf.org/doc/html/draft-irtf-nmrg-energy-aware-04>>.
- [draft-dwon-t2trg-multiedge-arch] Dwon et al., J., "Multi-Edge Architecture for the Internet of Things (IoT). IETF draft draft-dwon-t2trg-multiedge-arch-02, Expired", 2025, <<https://datatracker.ietf.org/doc/draft-dwon-t2trg-multiedge-arch/>>.
- [draft-si-service-mesh-dta] Si et al., Z., "Service Mesh-based Data Transfer Architecture. IETF draft draft-si-service-mesh-dta-01, Expired", 2025, <<https://datatracker.ietf.org/doc/draft-si-service-mesh-dta/>>.
- [draft-ietf-wimse-workload-identity-practices] Richardson et al., M., "Workload Identity Best Practices. IETF draft draft-ietf-wimse-workload-identity-practices-00, active", 2025, <<https://datatracker.ietf.org/doc/draft-ietf-wimse-workload-identity-practices/>>.

[draft-contreras-nmrg-interconnection-intents]

Contreras et al., L., "Interconnection Intents for Network Services. IETF draft draft-contreras-nmrg-interconnection-intents-05, Expired", 2025,
<<https://datatracker.ietf.org/doc/draft-contreras-nmrg-interconnection-intents/>>.

Acknowledgements

This work has been funded by The European Commission in the context of the Horizon Europe CODECO project under grant number 101092696, and by SGC, Grant agreement nr: M-0626, project SemComIIoT.

We thank Minh-Ngoc Tran for his contributions towards alignment with the draft[ietf-bmwg-07] , and suggestions for the removal of the former section 4, which provided a CNI summary only.

Appendix A. Change Log

-Since draft-samizadeh-bmwg-cni-benchmarking-00:

- * Section 4 and 5 were removed.
- * Added details about CNI Behavior in Federated and Multi-Cluster Environments.
- * Added details about Observability and Bottleneck Detection in multi-cluster or federated environments.
- * Revised references to Kubernetes network model and IETF drafts.
- * Minor editorial updates and formatting corrections.

Authors' Addresses

Tina Samizadeh
fortiss GmbH
Guerickestr. 25
80805 Munich
Germany
Email: samizadeh@fortiss.org

George Koukis
ATHENA RC
University Campus South Entrance
67100 Xanthi
Greece

Email: George.Koukis@athenarc.gr

Rute C. Sofia
fortiss GmbH
Guerickestr. 25
80805 Munich
Germany
Email: sofia@fortiss.org
URI: www.rutesofia.com

Lefteris Mamatas
University of Macedonia
Egnatias 156
54636 Thessaloniki
Greece
Email: emamatas@uom.edu.gr

Vassilis Tsaoussidis
ATHENA RC
University Campus South Entrance
67100 Xanthi
Greece
Email: vassilis.tsaoussidis@gmail.com