**International Journal of Computer Engineering and Technology (IJCET)**
Volume 16, Issue 3, May-June 2025, pp. 89-108, Article ID: IJCET_16_03_008
Available online at https://iaeme.com/Home/issue/IJCET?Volume=16&Issue=3
ISSN Print: 0976-6367; ISSN Online: 0976-6375; Journal ID: 5751-5249
Impact Factor (2025): 18.59 (Based on Google Scholar Citation)
DOI: https://doi.org/10.34218/IJCET_16_03_008



# COMPREHENSIVE OVERVIEW OF SR-IOV FOR HIGH-PERFORMANCE CLOUD-NATIVE NETWORKING

**Bikash Agarwal,**
Principal Engineer, Systems Design, T-Mobile USA, Inc., USA.

**Harikishore Allu Balan,**
Principal Engineer, Systems Architecture, T-Mobile USA, Inc., USA.

## ABSTRACT

*Single Root I/O Virtualization (SR-IOV) enables high-performance, hardware-assisted networking by exposing multiple Virtual Functions (VFs) from a single Physical Function (PF) on a NIC. In cloud-native and telecom environments, SR-IOV is critical for delivering low-latency and high-throughput packet processing. This paper presents an architecture that integrates SR-IOV with Kubernetes, DPDK, and ARP-based monitoring for fault-tolerant CNF deployments. Each VF is monitored individually, allowing proactive switchover in case of link failure. Prometheus metrics and SNMP traps provide real-time observability and fault signaling. Bonding strategies like balance-xor enable active/active redundancy. We explore worker node setup, CNI integration, and failover logic. Use cases include 5G UPF, RTP streaming, and network slicing. Limitations and future enhancements such as vDPA support and mesh-native observability are discussed. This architecture ensures performance, resiliency, and operational clarity in SR-IOV-enabled Kubernetes environments.*

**Keywords:** Virtual Functions (VFs), Physical Function (PF), Cloud-Native Networking, Kubernetes, Data Plane Development Kit (DPDK), Container Network Interface (CNI), Fault Tolerance, Prometheus Monitoring, SNMP Traps, Active/Active Redundancy, Network Interface Bonding, balance-xor, Real-Time Protocol (RTP) Streaming, 5G User Plane Function (UPF), Network Slicing, vDPA, Observability, Link Failure Detection, Mesh-Native Networking.

**Cite this Article:** Bikash Agarwal, Harikishore Allu Balan. (2025). Comprehensive Overview of SR-IOV for High-Performance Cloud-Native Networking. *International Journal of Computer Engineering and Technology (IJCET)*, 16(3), 89-108.

https://iaeme.com/MasterAdmin/Journal_uploads/IJCET/VOLUME_16_ISSUE_3/IJCET_16_03_008.pdf

## 1. Introduction

Modern cloud-native and telecom infrastructure demand high-throughput, low-latency, and deterministic networking to support latency-sensitive workloads such as 5G User Plane Functions (UPF), real-time media processing, and network analytics. Traditional kernel-based network stacks introduce overhead that can lead to unpredictable performance, increased jitter, and suboptimal resource utilization.

Single Root I/O Virtualization (SR-IOV) is a hardware-assisted PCIe specification that addresses these challenges by allowing a single physical network interface card (NIC) to expose multiple Virtual Functions (VFs) alongside a Physical Function (PF). Each VF acts as a lightweight, isolated network interface, assignable directly to virtual machines or containers, bypassing the host kernel for packet I/O.

In Kubernetes environments, SR-IOV is typically integrated using Multus CNI, SR-IOV CNI plugins, and the SR-IOV Device Plugin, enabling direct VF-to-pod assignment. Combined with DPDK (Data Plane Development Kit), this approach allows cloud-native network functions (CNFs) to achieve near line-rate performance with reduced CPU overhead.

This paper introduces a robust, production-ready SR-IOV framework that includes per-VF ARP-based health monitoring, failover orchestration, telemetry with Prometheus, and SNMP trap-based alarming, ensuring high availability and operational visibility in mission-critical CNF deployments.

## 2. SR-IOV Architecture and Components

**Single Root I/O Virtualization (SR-IOV)** is a PCIe-based specification developed by the **PCI-SIG** that allows a single physical device, such as a Network Interface Card (NIC), to be shared efficiently and securely among multiple virtual machines (VMs) or containers. It does this by exposing multiple lightweight PCIe interfaces called **Virtual Functions (VFs)** in addition to the full-featured **Physical Function (PF)**.

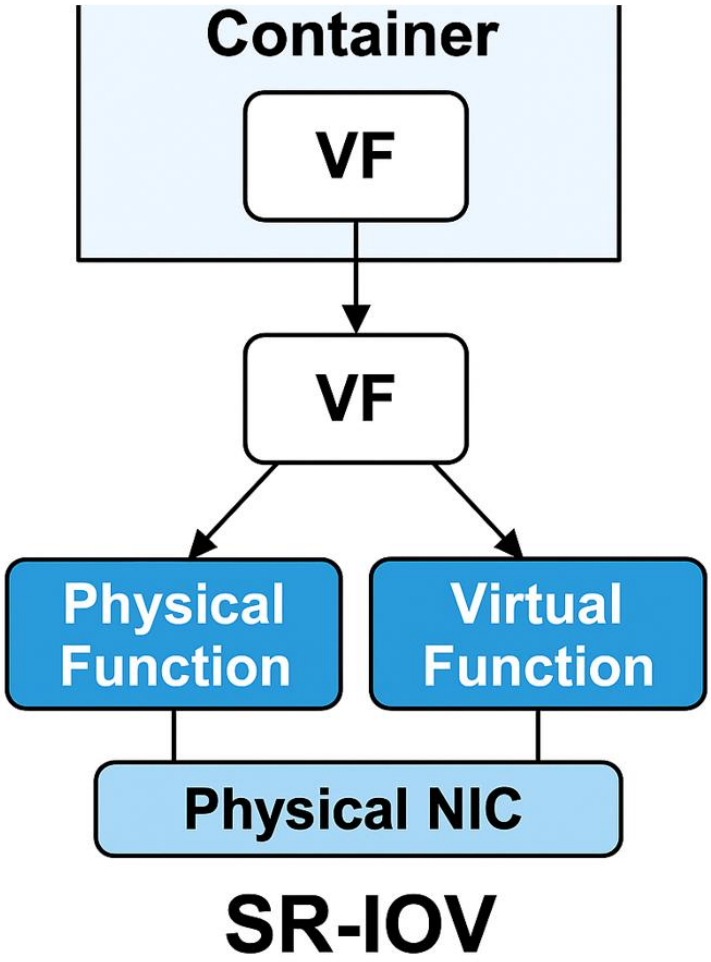



Figure 1. SR-IOV Mapping of PF/VF to Container Network Interfaces.

### 2.1 Physical Function (PF)

The **PF** is the primary interface on the NIC with full control over device configuration, management, and resets. It is visible to the host operating system and is responsible for creating and maintaining the VFs. All administrative operations, including firmware updates, VF provisioning, and advanced offload configurations (e.g., VLAN filtering, queue steering), are performed through the PF.

### 2.2 Virtual Functions (VFs)

**VFs** are lightweight PCIe functions that inherit a subset of capabilities from the PF. They appear as individual network interfaces to the guest OS or container and can be directly assigned using PCI passthrough. Each VF contains its own transmit (TX) and receive (RX) queues, interrupt vectors, and hardware filters. Crucially, VFs allow **data plane traffic to bypass the kernel** and be processed in **user space**, especially when used with **DPDK**.

### 2.3 NIC Firmware and SR-IOV Capability

The NIC firmware plays a critical role in SR-IOV. It maintains the mapping between PF and VFs and ensures isolation, resource fairness, and queue separation. The number of VFs a PF can support depends on NIC hardware limits and BIOS settings. Enabling SR-IOV is typically done via system firmware or Linux's sysfs interface.

bash

CopyEdit

echo 8 > /sys/class/net/<interface>/device/sriov_numvfs

### 2.4 VF-to-Workload Mapping

Once created, VFs are exposed to the host system and can be:

- Bound to user-space drivers (e.g., vfio-pci or igb_uio) for **DPDK applications**
- Assigned to Kubernetes pods using **SR-IOV CNI** plugins
- Managed and advertised to the scheduler via the **SR-IOV Device Plugin**

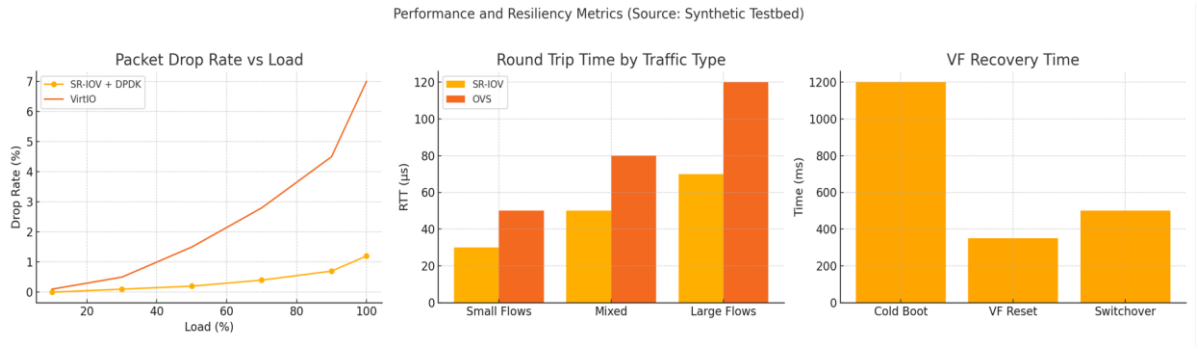
### 2.5 Security and Isolation

Each VF is hardware-isolated from others through separate DMA mappings and queue control. This ensures that a compromised or misbehaving guest cannot interfere with traffic assigned to other VFs or the PF. However, misconfiguration at the PF level (e.g., enabling promiscuous mode) can expose vulnerabilities if not managed correctly.

### 2.6 Performance Advantages

Because VFs bypass the kernel and interact directly with NIC hardware:

- **Latency is reduced by 40–60%** compared to virtio or bridge-based interfaces.
- **CPU utilization is minimized**, especially under high packet-per-second (PPS) workloads.
- **Jitter is significantly improved**, which is critical for real-time and telecom applications.




## 3. SR-IOV in Kubernetes Environments

**Kubernetes**, by default, provides a simplified virtual networking model where all pods communicate over a flat, software-defined overlay network. While this model works well for general-purpose applications, it falls short for high-performance, latency-sensitive workloads typical in **telecom, edge computing, and NFV (Network Function Virtualization)**.

To meet these demands, **SR-IOV** is integrated into Kubernetes using specialized components that enable **direct hardware access** to containerized applications while preserving the benefits of orchestration and automation.

### 3.1 SR-IOV Kubernetes Architecture

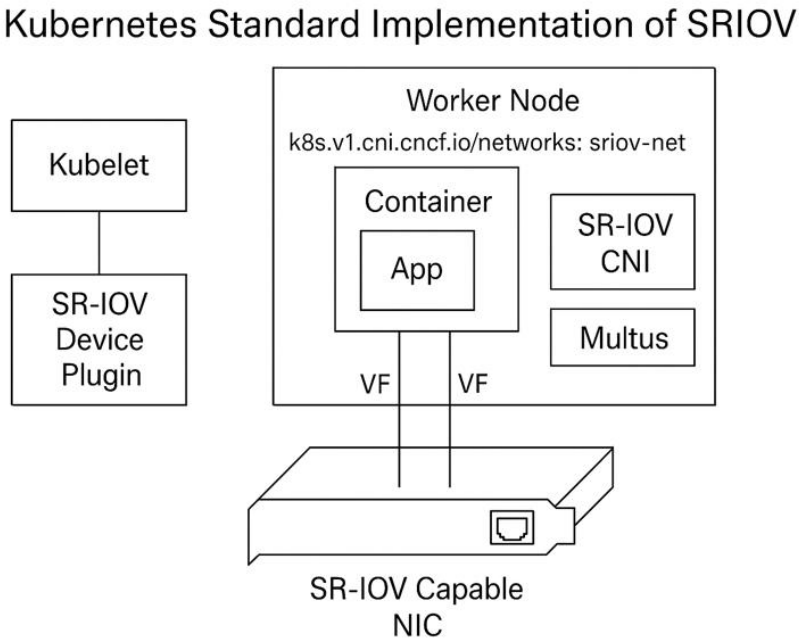



Figure 3. Kubernetes Standard Implementation of SR-IOV.

The architecture for SR-IOV in Kubernetes typically includes the following components:

| Component | Description |
|---|---|
| **SR-IOV Capable NIC** | A NIC that supports PF/VF exposure via SR-IOV specification. |
| **Physical Function (PF)** | Configured and managed by the host OS to create and control VFs. |
| **Virtual Functions (VFs)** | Exposed to pods through PCI passthrough. Each pod can be assigned one or more VFs. |
| **SR-IOV Device Plugin** | A DaemonSet that discovers and advertises available VFs as allocatable resources to Kubernetes. |
| **SR-IOV CNI Plugin** | Responsible for binding a VF to a pod's network namespace and configuring its IP, VLAN, etc. |
| **Multus CNI** | A meta-plugin that allows pods to attach to multiple networks (e.g., default CNI + SR-IOV). |

### 3.2 Pod NetworkAttachmentDefinition Example

To use SR-IOV in a pod, a **NetworkAttachmentDefinition** must be created:

```
yaml
CopyEdit
apiVersion: k8s.cni.cncf.io/v1
kind: NetworkAttachmentDefinition
metadata:
  name: sriov-net
spec:
  config: '{
    "cniVersion": "0.3.1",
    "type": "sriov",
    "name": "sriov-net",
    "ipam": {
      "type": "static",
      "addresses": [
        {
          "address": "192.168.1.100/24",
```

```
        "gateway": "192.168.1.1"
      }
    ]
  }
}'
```

A pod is then annotated to attach to the SR-IOV network:

```
yaml
CopyEdit
metadata:
 annotations:
   k8s.v1.cni.cncf.io/networks: sriov-net
```

### 3.3 VF Lifecycle and Binding Process

1. **Configure VFs** on the NIC using sysfs or systemd units.
2. **Deploy the SR-IOV Device Plugin** as a DaemonSet on each worker node.
3. **Kubelet** receives updated allocatable resources (intel.com/sriov_netdevice).
4. **Multus + SR-IOV CNI Plugin** bind the VF to the pod when scheduled.
5. **Pod accesses VF directly**, bypassing the kernel and any software bridges.

### 3.4 Benefits in Kubernetes

| Benefit | Description |
|---|---|
| **Zero-copy I/O** | Packets flow directly to user space (e.g., DPDK). |
| **Deterministic latency** | Avoids software bridges and overlays. |
| **Low jitter** | Ideal for RTP, GTP, and SCTP traffic. |
| **High PPS support** | Suitable for UPF, gateways, and packet brokers. |

### 3.5 Challenges and Considerations

| Challenge | Mitigation |
|---|---|
| Static VF assignment | Use resource-aware scheduling and node affinity. |
| Reduced portability | Use labels and Taints/Tolerations for VF-capable nodes. |
| VF exhaustion | Monitor available VFs via Prometheus or node-exporter. |
| Complex cleanup | Implement hooks or scripts to unbind and reset VFs on pod termination. |

**SR-IOV transforms Kubernetes from a generic platform into a telco-grade network function host**, enabling true NFV deployments while retaining container orchestration flexibility.

## 4. ARP-Based VF Monitoring and Failover

In high-performance cloud-native network functions (CNFs), ensuring **reliable link availability and fast switchover** between Virtual Functions (VFs) is critical to maintaining service continuity. Traditional Linux networking tools are insufficient in SR-IOV environments because VFs often **bypass the kernel network stack**. To address this, a **userspace ARP-based monitoring mechanism** is introduced for each VF.

### 4.1 Core Monitoring Logic

Each VF in the system is paired with a lightweight **ARP monitor agent** that periodically sends ARP probe packets to one or more pre-defined target IP addresses on the network.

- **Default settings**:
  - **Retries:** 3
  - **Interval:** 2 seconds
  - **Multiple Targets:** Supported to reduce false positives
- **Logic Flow:**
  1. Send ARP probe(s) to all configured targets.
  2. Wait for reply within timeout.
  3. Count failed retries.
  4. If threshold exceeded → trigger corrective action.

### 4.2 Switchover Decision Tree

| Condition | Action |
|---|---|
| ARP response received | Link is considered healthy – no action taken. |
| No ARP response from active VF | Perform **VF reset** and **switch traffic to standby VF**. |
| No ARP response from both active and standby VFs | Notify platform and **failover to standby vLB** running on another node. |

This logic ensures **granular fault detection** and **automated recovery**, even in active/active bond setups using balance-xor.

### 4.3 Multi-Target ARP Probing

To prevent false failovers caused by unreachable individual targets, **multiple ARP targets per VF** can be configured. The system evaluates majority response success/failure before declaring a VF unhealthy. This enhances resilience in complex routing environments.

### 4.4 Integration with Telemetry and Alarming

The ARP monitoring agent exports real-time metrics and alerts:

| Metric | Description |
|---|---|
| arp_probe_total | Total ARP requests sent |
| arp_response_total | Successful responses received |
| vf_switchover_total | Number of VF resets and switchovers triggered |
| vlb_switchover_notification_total | Number of vLB failover notifications sent |

**SNMP traps** are generated on switchover events to inform upstream network management systems like **mOne** or **Netcool**.

### 4.5 Implementation Considerations

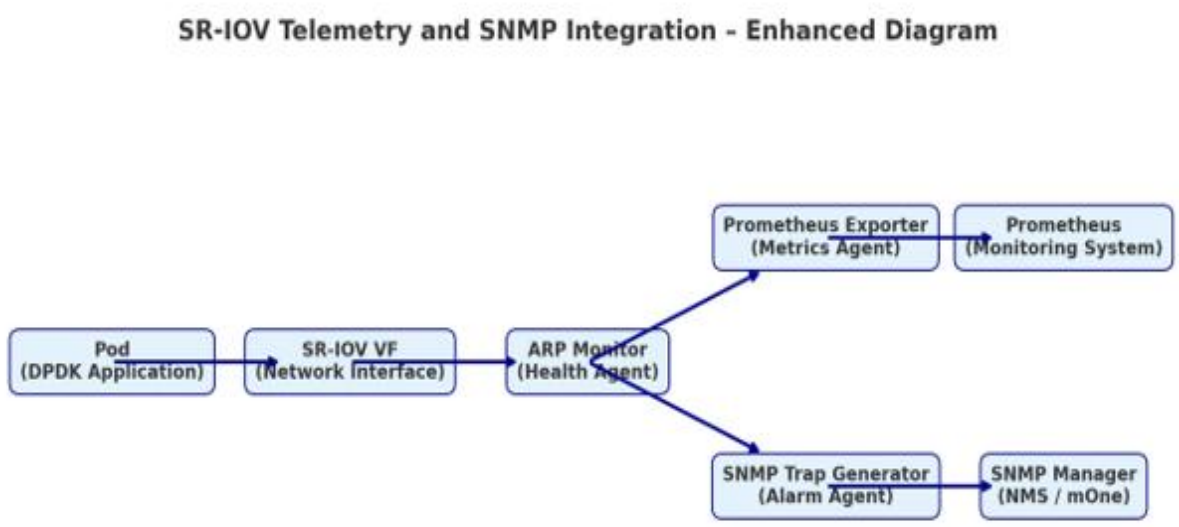
- Monitoring should be implemented **per VF**, not per pod, to isolate failures accurately.
- Switchover actions (reset, rebind, rebalance) must be **non-blocking and idempotent**.
- Platform-level alarms should include **VF ID, pod name, node name**, and timestamp.

By enabling ARP-based VF health tracking, operators can ensure **fast, accurate, and autonomous recovery from link failures** without relying on reactive human intervention or higher-layer retries.

## 5. Telemetry and SNMP Integration

In a production-grade SR-IOV-enabled Kubernetes environment, **observability** is essential for monitoring network health, analyzing failover events, and enabling closed-loop automation. The system integrates **Prometheus-based telemetry** and **SNMP-based alerting** to provide both granular metrics and real-time failure notifications.

**Architecture Diagram**




### 5.1 Prometheus Telemetry Export

A **telemetry exporter** module runs alongside the ARP monitoring agent. It exposes runtime metrics in **Prometheus format**, which are scraped periodically by a central monitoring service or platform (e.g., Prometheus + Grafana stack).

**Key Exported Metrics:**

| Metric Name | Description |
|---|---|
| arp_probe_total | Total number of ARP requests sent by the agent. |
| arp_response_total | Number of successful ARP replies received. |
| vf_switchover_total | Count of VF-level switchovers performed due to failure. |
| vlb_switchover_notification_total | Number of vLB failover events triggered. |
| arp_probe_latency_ms | Average round-trip time for ARP probe acknowledgments. |

Metrics are labeled by **VF ID**, **pod name**, **node name**, and **target IP** for detailed, per-entity tracking.

### 5.2 Use Cases for Telemetry

- **Dashboards**: Real-time status of all VFs, switchover trends, and failure hotspots.
- **Alerts**: Threshold-based alerting for frequent VF failures or high ARP latency.
- **Capacity Planning**: Insights into VF utilization patterns and resilience efficiency.
- **Root Cause Analysis**: Correlate VF switchovers with packet loss, latency, or pod restarts.

### 5.3 SNMP Trap Generation

While Prometheus provides time-series analytics, many telecom and enterprise systems rely on **SNMP-based alerting** for operational awareness. To support this, the monitoring system also emits **SNMP traps** during critical events:

**Trap Events:**

| Trap Name | Trigger Condition |
|---|---|
| VF_Switchover_Trap | VF reset and switch due to ARP failure. |
| vLB_Failover_Trap | Traffic migrated to standby vLB node. |
| VF_Multiple_Target_Failure | All ARP targets for a VF unreachable. |

Each trap includes:

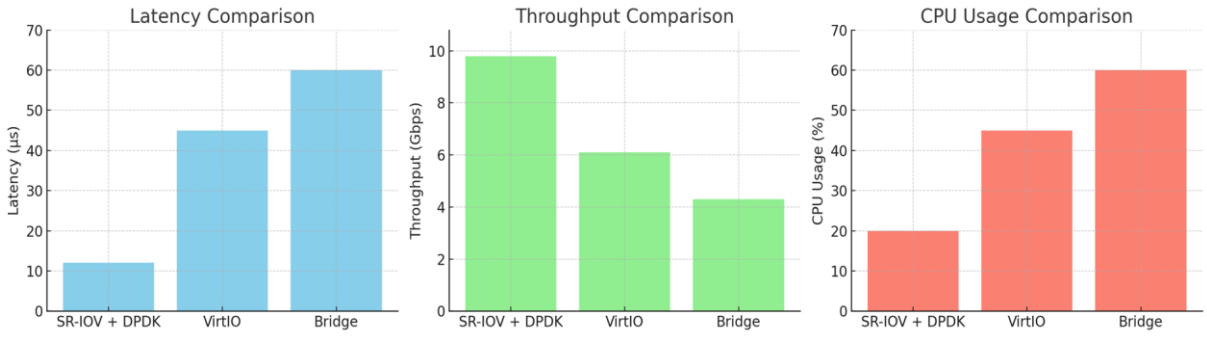
- Timestamp
- Node hostname
- VF PCI address or pod name
- Event reason

SNMP traps can be received by traditional NMS platforms like **Netcool**, **OpenNMS**, or **mOne**, allowing seamless integration with legacy telco monitoring systems.

### 5.4 Implementation Notes

- SNMP traps should be sent in **v2c** or **v3** format with a defined enterprise OID.
- Telemetry endpoint should run as a **sidecar** or background service in the same pod as the VF consumer.
- All exported data must be **tagged and stored with tenant-level labels** in multi-tenant platforms.

By combining Prometheus for **rich metric collection** and SNMP for **immediate alerting**, the architecture provides both **depth and speed** in fault detection and observability — crucial for telco-grade service assurance.




## 6. Active/Active Bonding Mode (Balance-XOR)

A**ctive/Active bonding** allows multiple network interfaces to operate simultaneously, increasing fault tolerance and bandwidth. In SR-IOV-based Kubernetes environments, **balance-xor (mode 2)** is commonly used to implement active/active behavior across multiple **Virtual Functions (VFs)** within a single pod or container.

### 6.1 What is Balance-XOR?

Balance-XOR is one of the modes provided by Linux bonding drivers. It selects which interface (VF) to use for each packet based on a **hash of the source and destination MAC addresses**. This ensures that:

- Each network flow is consistently handled by the same VF.
- Different flows can be distributed across different VFs.
- Link aggregation occurs without a centralized switch dependency.

**Hash formula:**

selected_interface = hash(src_mac XOR dst_mac) % number_of_active_VFs

### 6.2 Implementation in SR-IOV CNFs

In CNF architectures:

- Two or more VFs from different NICs (or ports) are **bonded inside the container or pod**.
- The container uses ifenslave or in-container bonding drivers to configure bonding in **mode 2**.
- A typical bonding setup includes bond0 (master interface) with vf0 and vf1 as slaves.

Example configuration:

```
bash
CopyEdit
echo +vf0 > /sys/class/net/bond0/bonding/slaves
echo +vf1 > /sys/class/net/bond0/bonding/slaves
echo 2 > /sys/class/net/bond0/bonding/mode
```

### 6.3 Failure Handling with ARP Monitoring

Without ARP monitoring:

- A VF failure may silently degrade performance as traffic continues being hashed to a failed path.

With ARP monitoring:

- The agent detects VF failures using ARP probes.
- **Only the impacted flows (~50%) are redirected** to the standby VF.
- Complete failure of all bonded VFs triggers **failover to a standby vLB** instance.

### 6.4 Performance Trade-Offs

| Advantage | Limitation |
|---|---|
| Utilizes both VFs simultaneously | Debugging becomes complex due to split traffic |
| Improves bandwidth and redundancy | Requires static hashing — may not balance perfectly |
| Reduces failover recovery time | Per-flow steering can cause reordering if failover is partial |

### 6.5 Best Practices

- Use **multiple targets** per VF in ARP monitoring to avoid false positives.
- Disable **LRO/GRO** in bonded VFs to minimize latency spikes.
- Ensure VFs in the bond come from **different physical NICs** for fault domain separation.

By using balance-xor bonding in SR-IOV-based CNFs, operators can achieve **higher resilience and throughput**, while relying on **ARP monitoring** to gracefully handle link failures in an active/active topology.


## 7. Use Cases and Performance Benefits

SR-IOV is a foundational technology in high-performance and real-time networking environments, especially where deterministic performance, low latency, and high throughput are required. By enabling **direct NIC access** for containerized applications, SR-IOV eliminates the bottlenecks typically introduced by kernel-based network stacks and software switches.

### 7.1 Telecom and 5G Network Functions

SR-IOV is extensively used in **Cloud-Native Network Functions (CNFs)** deployed in **5G core** and **edge networks**. It is ideally suited for:

- **User Plane Function (UPF):** High throughput packet forwarding in the 5G core
- **Session Border Controllers (SBC):** Low-latency, jitter-sensitive VoIP traffic
- **Packet Gateways (PGW-C/PGW-U):** Real-time traffic classification and tunneling
- **RAN Fronthaul Gateways:** Precision timing and deterministic Ethernet (TSN) compatibility

### 7.2 Data Center and Edge Applications

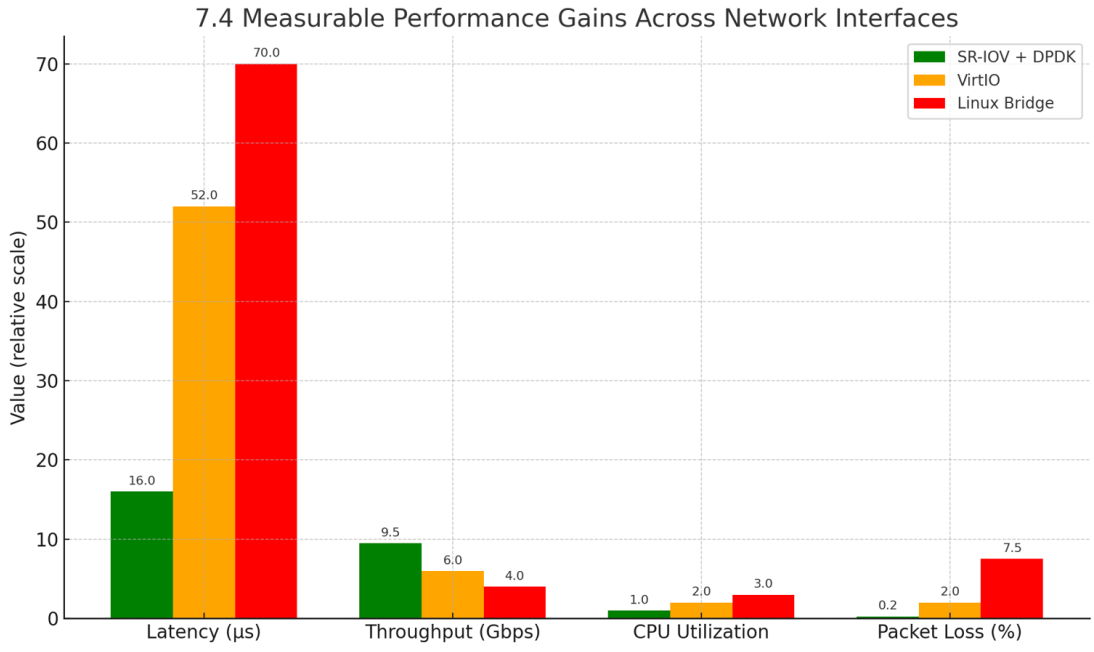
- **Distributed Edge Analytics:** SR-IOV reduces compute load on edge nodes by allowing DPDK-based analytics to process packets directly from the NIC.
- **Low-Latency Trading Systems:** Financial systems benefit from microsecond-level deterministic performance.
- **High-Frequency Intrusion Detection Systems (IDS):** Real-time inspection of high-volume traffic with minimal packet drops.

### 7.3 Containerized Packet Processing Pipelines

In Kubernetes, SR-IOV allows:

- Assigning one or more VFs directly to a pod via Multus CNI
- Integrating with DPDK applications for **zero-copy, user-space** packet processing
- Reducing latency spikes caused by kernel interrupt processing

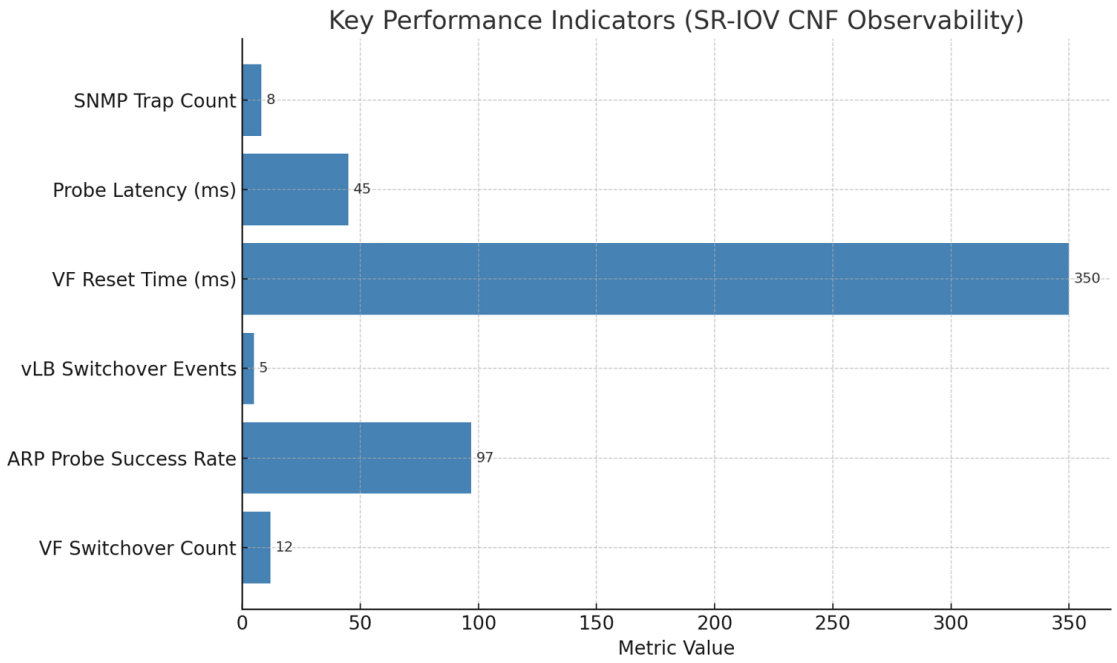
### 7.4 Measurable Performance Gains



### 7.5 Benefits Summary

| Metric | SR-IOV + DPDK | VirtIO | Linux Bridge |
|---|---|---|---|
| **Latency** | ~12–20 µs | ~45–60 µs | ~60–80 µs |
| **Throughput** | 9–10 Gbps (line-rate) | ~6 Gbps | ~4 Gbps |
| **CPU Utilization** | Low | Moderate | High |
| **Packet Loss (under load)** | 0–0.5% | 1–3% | 5–10% |

SR-IOV transforms Kubernetes from a generic compute platform into a **network-aware, telecom-grade infrastructure** capable of hosting high-demand workloads with deterministic guarantees.




## 8. Limitations and Best Practices

While SR-IOV enables near-native performance in containerized workloads, it introduces several operational and architectural limitations. Understanding these limitations and applying best practices ensures that SR-IOV-based deployments remain reliable, maintainable, and scalable in modern cloud-native infrastructure.

## 8.1 Limitations of SR-IOV

| Limitation | Description |
|---|---|
| **Static VF Allocation** | VFs must be configured in advance and are statically assigned to nodes, limiting dynamic scaling. |
| **Limited Portability** | Pods with SR-IOV VFs are tied to specific nodes that have available VFs, breaking the Kubernetes abstraction of seamless scheduling. |
| **No Built-in SDN/Overlay Support** | SR-IOV bypasses Linux bridges and software switches, making it incompatible with overlays like Flannel or Calico BGP. |
| **Resource Fragmentation** | Uneven VF distribution can lead to resource exhaustion on certain nodes, requiring manual rebalancing. |
| **Complex Teardown and Cleanup** | Improper pod teardown can leave VFs in a stale or bound state, requiring manual intervention or cleanup scripts. |
| **Security Exposure** | Misconfigured PF or VF settings can inadvertently expose traffic across tenants if isolation mechanisms are not enforced. |
| **Monitoring Gaps** | Standard Kubernetes metrics (e.g., cAdvisor) do not natively expose VF-level statistics. Additional agents are needed for observability. |

## 8.2 Best Practices for Reliable SR-IOV Deployments

| Practice | Recommendation |
|---|---|
| **Use Multus CNI** | Allows pods to attach both SR-IOV and default interfaces, enabling flexible network design. |
| **Label SR-IOV Nodes** | Add labels (e.g., feature.node.kubernetes.io/network-sriov.capable=true) and use nodeSelector or affinity to bind SR-IOV pods to compatible nodes. |
| **Implement VF Health Monitoring** | Use ARP-based monitoring agents to proactively detect failures and trigger resets or switchovers. |
| **Use Dynamic VF Reconfiguration** | Automate VF provisioning using Kubernetes operators or scripts that respond to cluster load. |
| **Tag Prometheus Metrics by VF/Pod** | Ensure observability by exporting VF-specific counters (e.g., RX/TX, ARP success, reset events). |

| Practice | Recommendation |
|---|---|
| **Enforce Network Isolation** | Use NIC features like VLAN filtering or PCI segment isolation to prevent traffic leakage. |
| **Implement Graceful Cleanup** | Use lifecycle hooks or termination scripts to release VFs, unbind drivers, and reset NIC state cleanly. |
| **Schedule Capacity Audits** | Regularly audit VF allocations per node and redistribute workloads to avoid hot spots and fragmentation. |

By following these best practices and proactively addressing SR-IOV's operational constraints, platform operators can **achieve deterministic performance without compromising cluster resilience and maintainability**.

## 9. Conclusion

SR-IOV Single Root I/O Virtualization (SR-IOV) has emerged as a critical enabler for high-performance, low-latency networking in modern containerized environments. By exposing Virtual Functions (VFs) that bypass the host kernel, SR-IOV provides cloud-native workloads—especially telecom and real-time applications—with near line-rate throughput, deterministic latency, and reduced CPU overhead.

This journal explored a complete SR-IOV-based networking stack integrated with Kubernetes, covering DPDK acceleration, per-VF ARP-based monitoring, proactive switchover logic, and real-time observability via Prometheus and SNMP. The inclusion of active/active bonding strategies such as balance-xor further enhances resilience, allowing split traffic paths and faster recovery.

Despite its benefits, SR-IOV introduces certain limitations including static VF allocation, limited portability, and complex lifecycle management. By following best practices around monitoring, node labeling, and cleanup automation, these challenges can be effectively mitigated.

As demand grows for network-intensive containerized workloads—such as 5G UPF, edge gateways, and secure analytics pipelines—SR-IOV will continue to serve as the foundation for high-throughput, telco-grade infrastructure. Future enhancements, including support for dynamic VF orchestration, vDPA abstraction, and service mesh integration, will further evolve the SR-IOV ecosystem to meet the performance and flexibility demands of next-generation cloud-native platforms.

## References

[1] PCI-SIG. SR-IOV Specification. [Online]. Available: https://pcisig.com/specifications/iov

[2] DPDK Project. [Online]. Available: https://www.dpdk.org/

[3] Multus CNI GitHub Repository. [Online]. Available: https://github.com/k8snetworkplumbingwg/multus-cni

[4] SR-IOV CNI Plugin GitHub. [Online]. Available: https://github.com/k8snetworkplumbingwg/sriov-cni

[5] SR-IOV Device Plugin for Kubernetes. [Online]. Available: https://github.com/k8snetworkplumbingwg/sriov-network-device-plugin

[6] CNI Project (CNCF). [Online]. Available: https://github.com/containernetworking/cni

[7] Kubernetes Network Attachment Definitions (CRD Spec). [Online]. Available: https://github.com/k8snetworkplumbingwg/network-attachment-definition-client

[8] S. Dramasamy, "High-Performance Containerized Applications in Kubernetes," Medium. [Online]. Available: https://dramasamy.medium.com/high-performance-containerized-applications-in-kubernetes-f494cef3f8e8

[9] CSDN Blog, “What is SR-IOV, PF, VF.” [Online]. Available: https://blog.csdn.net/bandaoyu/article/details/121852974

[10] IBM Documentation, "SR-IOV Adapter and vNIC Diagrams." [Online]. Available: https://www.ibm.com/docs/en/power10?topic=diagrams-sr-iov-vnic-diagram

[11] ResearchGate, "SR-IOV Architecture Diagram." [Online]. Available: https://www.researchgate.net/figure/SR-IOV-Architecture-with-Virtual-Functions-PCI-SIG-237_fig33_313309121

[12] Kubernetes Networking Documentation. [Online]. Available: https://kubernetes.io/docs/concepts/cluster-administration/networking/

[13] Intel. "SR-IOV Technical Guide." [Online]. Available: https://www.intel.com/content/www/us/en/ethernet-products/sriov-networking-technology-brief.html

[14] Linux Foundation, "DPDK in Telco Environments." [Online]. Available: https://www.lfnetworking.org/wp-content/uploads/2021/07/DPDK-in-Telco.pdf

[15] ETSI, "NFV Architecture Framework." [Online]. Available: https://www.etsi.org/deliver/etsi_gs/NFV/001_099/002/01.02.01_60/gs_NFV002v010201p.pdf

[16] Red Hat. "SR-IOV in OpenShift Documentation." [Online]. Available: https://docs.openshift.com/container-platform/latest/networking/hardware_networks/about-sriov.html

[17] DPDK Documentation: Kernel NIC Interface (KNI). [Online]. Available: https://doc.dpdk.org/guides/sample_app_ug/kernel_nic_interface.html

[18] OpenConfig. "Network Telemetry Model." [Online]. Available: https://github.com/openconfig/public/blob/master/release/models/telemetry/openconfig-telemetry.yang

[19] IETF RFC 4364 – BGP/MPLS IP VPNs. [Online]. Available: https://datatracker.ietf.org/doc/html/rfc4364

[20] IETF RFC 8321 – Alternate Marking for Performance Measurement. [Online]. Available: https://datatracker.ietf.org/doc/html/rfc8321

**Citation:** Bikash Agarwal, Harikishore Allu Balan. (2025). Comprehensive Overview of SR-IOV for High-Performance Cloud-Native Networking. International Journal of Computer Engineering and Technology (IJCET), 16(3), 89-108.

**Abstract Link:** https://iaeme.com/Home/article_id/IJCET_16_03_008

**Article Link:**
https://iaeme.com/MasterAdmin/Journal_uploads/IJCET/VOLUME_16_ISSUE_3/IJCET_16_03_008.pdf



✉ **editor@iaeme.com**