POLITECNICO DI TORINO

Master's Degree in Computer Engineering

Master's Degree Thesis

Evaluating Hardware Offload of Network Functions with SmartNICs

Supervisors

Prof. Fulvio RISSO

Ph.D. Student Davide MIOLA

Candidate

Vincenzo COSI

April 2025

Summary

The increasing demand for efficient and scalable data center networks has led the industry to the exploration of hardware acceleration technologies, which allow to offload portions of the Linux network stack processing away from the main CPUs and into dedicated accelerators such as *SmartNICs*, thereby promising advancements in overall system performance and efficiency.

This thesis questions these claims by conducting a series of experiments to evaluate the performance and efficiency gains achieved by adopting NVIDIA ConnectX-7 100 Gbps SmartNICs across several server configurations. The tests covered a variety of networking scenarios, including encapsulation, IPsec encryption, and the deployment of real world cloud-native applications within a Kubernetes cluster.

Results indicate that SmartNICs can significantly reduce CPU overhead for IPsec encryption tasks. However, in other cases, improvements were only measurable on older and less powerful processors, while more modern hardware showed limited benefits.

Acknowledgements

Non sono mai stato particolarmente bravo con le parole, né tantomeno nell'esprimere i miei sentimenti. Scrivere questi ringraziamenti non è semplice, quindi perdonatemi se sarò un po' conciso.

Il mio primo grazie va al **Prof. Risso**, per avermi dato l'opportunità di svolgere questa tesi e per il supporto costante durante tutto il percorso che ne ha reso possibile la realizzazione.

Un sentito ringraziamento a **Federico e Davide**, i miei dottorandi, per avermi guidato con pazienza e competenza nei momenti più complessi che non sono certo mancati lungo il cammino.

Grazie di cuore a tutto il **Lab 9**: un ambiente raro, stimolante e accogliente, che ha trasformato le giornate al Politecnico in qualcosa di piacevole e significativo. Le pause pranzo al sushi, le serate giochi e tutti gli attimi di leggerezza hanno reso questi mesi davvero speciali.

Un ringraziamento speciale a **Fra Cappa**, per l'incredibile supporto tecnico e umano. Non so come avrei fatto senza di te... e ancora non so come farò d'ora in poi. E ovviamente grazie anche ad Attilio, per le mozzarelle, la torta polacca aversana, e – soprattutto – per il costante incoraggiamento e la tua *inesauribile positività*.

Table of Contents

List of Tables

List of Figures

Acronyms

ACL
: access control list

CNI
: container network interface

DMA
: direct memory access

HPC
: high-performance computing

IPsec
: internet protocol security

ISR
: interrupt service routine

NFV
: network function virtualization

NIC
: network interface card

OVN
: open virtual network

OVS
: open virtual switch

PF

physical function

SA

security association

SDN

software-defined network

SR-IOV

single-root input/output virtualization

TC

traffic control

TCP

transmission control protocol

VF

virtual function

VM

virtual machine

VXLAN

virtual extensible LAN

Chapter 1

Introduction

1.1 Goal of the Thesis

The primary objective of this thesis is to evaluate the viability and potential benefits of implementing hardware offload on SmartNICs within a data center environment. Specifically, this work investigates whether leveraging on SmartNICs, such as NVIDIA ConnectX-7, can provide measurable improvements in performance, efficiency, or other critical metrics that justify their adoption.

Manufacturers of SmartNICs emphasize their potential to offload network-related tasks from the CPU, thereby reducing CPU usage and enabling better scalability in data center operations. This thesis seeks to critically assess these claims by examining whether the expected CPU savings and performance enhancements are observable in practical settings. For example, the existing literature highlights the advantages of hardware offload, such as reduced CPU overhead during packet processing, improved energy efficiency, and enhanced throughput for high-performance workloads.

To achieve this, a series of experiments were conducted utilizing NVIDIA ConnectX-7 SmartNICs, focusing on their capabilities and performance under various scenarios. These experiments aim to test the hypothesis that offloading tasks to SmartNICs significantly alleviates CPU load while maintaining or improving network performance. The findings aspire to provide insights into the practical implications and trade-offs of integrating hardware offload technologies in modern data center architectures.

Chapter 2

Background

2.1 Linux Networking Stack

The Linux networking stack is responsible for handling and routing network packets between various interfaces, applications, containers, or virtual machines (VMs). Although many concepts are similar to traditional host networking, containerized and virtualized environments introduce additional abstractions, such as virtual Ethernet (*veth*) pairs, bridge devices, or Open vSwitch (OVS) interfaces. This section provides an overview of how network traffic flows within the Linux kernel, detailing the path from the physical (or virtual) network interface to the receiving application, container, or VM.

2.1.1 Network to Socket or Virtual Interface

When a packet arrives on a physical interface, such as an Ethernet card, it is transferred via Direct Memory Access (DMA) to a ring buffer in kernel memory. At this point, the hardware triggers an interrupt. To avoid performing extensive processing in the Interrupt Service Routine (ISR) of the Network Interface Card (NIC) driver, Linux divides the operation into a top half (the ISR itself) and a bottom half (a deferrable function). The top half is concise, handling only critical operations such as acknowledging the interrupt and scheduling the bottom half. Meanwhile, the bottom half executes more time-consuming tasks, including parsing the packet and delivering it to higher layers of the networking stack.

In containerized or virtualized environments, the receiving interface might not be a physical NIC but rather a virtual device (for example, a *veth* pair endpoint, a tap device connected to a hypervisor, or an SR-IOV Virtual Function). Regardless of the interface type, once the packet is in kernel memory, it follows a similar flow: the packet is inspected for protocol headers, potential filtering rules are applied (e.g., via iptables or eBPF), and the packet is ultimately routed to its destination.

If the packet is destined for a local container or VM on the same host, it may be switched via a Linux bridge, Open vSwitch, or another virtual switch mechanism. The virtual switch effectively decides whether to forward the packet to another container's *veth* interface, to an internal interface representing a VM (like a tap device), or to pass it to the host network stack. In all cases, the kernel must match the packet against the relevant forwarding rules (in bridging mode) or flow tables (in OVS) to deliver it to the correct destination interface.

2.1.2 Data Transmission in Host and Virtualized Contexts

On the transmit path, the picture is typically simpler than on the receive side, as socket writes represent the primary entry point for outbound traffic. For blocking calls such as `write` or `send`, the kernel copies data from user space into kernel space, passing it through the appropriate protocol layers. For TCP/IP, this includes operations such as segmentation, congestion control, and flow control.

In virtualized or containerized environments, outbound traffic might originate from a container's application process writing to a virtual interface, for instance, a *veth* endpoint. The kernel sends the packet to the corresponding peer device on the host side (e.g., a bridge or OVS port). If the packet is destined for a different container on the same host, the bridging or switching component routes it internally without ever touching a physical NIC. In contrast, if the packet is destined for a remote endpoint, it is forwarded through the host's physical interface driver, potentially utilizing SR-IOV or offloading mechanisms to accelerate packet processing.

While the general principles of the network stack apply to both host-based and container-based traffic, virtualization layers introduce additional complexity, particularly with regard to:

- **Interface Types:** Physical NICs, *veth* pairs, tap devices, or SR-IOV Virtual Functions.
- **Bridging vs. Routing:** Packets can be switched via a Linux bridge or OVS if the source and destination share the same host.
- **Offloading:** SmartNICs and SR-IOV can bypass parts of the kernel networking stack, potentially reducing CPU overhead.
- **Security and Isolation:** Namespaces, cgroups, and network policies ensure container traffic remains isolated.

Nonetheless, the fundamental steps remain the same: inbound packets are DMA-transferred into host memory, processed by the kernel's networking stack (which may include bridging, routing, or offloading), and handed off to the appropriate

socket or virtual interface. Outbound packets, on the other hand, move from user space into the kernel stack, optionally passing through virtualization layers or virtual switches, and ultimately exit through a physical or virtual interface to reach their destination.

2.2 Open vSwitch

Open vSwitch (OVS) [1] is an open-source multilayer virtual switch that facilitates network automation through standard management interfaces and support for various tunneling protocols. Originally developed to operate within virtualized environments, OVS provides a flexible and programmable switching fabric that integrates with cloud platforms and containerized workloads.

One of the standout features of Open vSwitch (OVS) is its capability to manage both Layer 2 and Layer 3 forwarding. It supports advanced functionalities such as flow-based forwarding, quality of service (QoS), and traffic mirroring. OVS is frequently implemented in environments that leverage Software-Defined Networking (SDN), allowing for dynamic policy enforcement and effective network segmentation.

To enhance network performance, OVS supports hardware offloading, allowing data plane operations to be offloaded from the host CPU to specialized hardware such as SmartNICs. Through integration with technologies such as SR-IOV, OVS can achieve lower latency and higher throughput by leveraging hardware acceleration.

This thesis explores how OVS hardware offloading, particularly when combined with SmartNICs, impacts the performance of virtualized and containerized networking environments. By evaluating different deployment scenarios, the study aims to determine the effectiveness of OVS in reducing CPU utilization and improving network efficiency.

2.3 IPsec

IPsec (Internet Protocol Security) is a suite of protocols that provide secure communication over IP networks by ensuring data confidentiality, integrity, and authenticity. It is widely employed in Virtual Private Networks (VPNs), secure site-to-site communications, and protecting sensitive network traffic over untrusted networks. IPsec operates at the network layer of the OSI model, making it transparent to applications and enabling security at the IP packet level.

IPsec consists of several components, including the Authentication Header (AH) and the Encapsulating Security Payload (ESP). The AH provides integrity and authentication for IP packets by ensuring that transmitted data has not been

altered in transit. ESP, on the other hand, offers encryption and authentication, ensuring that data remains confidential and secure against eavesdropping attacks.

The protocol supports two operational modes: Transport Mode and Tunnel Mode. Transport Mode encrypts only the payload of the IP packet, leaving the header untouched, which is helpful for end-to-end communications between two hosts. Tunnel Mode, in contrast, encapsulates the entire IP packet within another IP packet, making it ideal for VPNs and secure communications between entire networks.

A fundamental challenge of IPsec is the computational burden introduced by cryptographic operations such as encryption and decryption. These processes demand significant CPU resources, potentially degrading system performance, especially in high-throughput environments. Modern networking hardware, including SmartNICs, to mitigate this, supports cryptographic offloading, where dedicated hardware accelerators handle encryption and decryption operations. This reduces CPU utilization, enhances throughput, and lowers network latency, making secure communications more efficient.

The adoption of IPsec extends beyond traditional networking, as it plays a crucial role in cloud security, secure data center communications, and 5G network infrastructure.

2.4 SR-IOV

Single Root I/O Virtualization (SR-IOV) is a technology that enables a single physical network interface card (NIC) to be shared efficiently among multiple virtual machines (VMs) or containers while maintaining high performance. SR-IOV is widely used in high-performance computing, cloud infrastructures, and data center environments where low-latency and high-throughput networking are required. SR-IOV exposes multiple Virtual Functions (VFs) from a single Physical Function (PF) on a NIC. Each VF operates as an independent network interface that can be assigned directly to a VM or container, bypassing the software-based network stack. This direct assignment reduces CPU overhead, minimizes latency, and increases network throughput compared to traditional virtualized networking solutions. The key benefits of SR-IOV include:

- **Improved Performance:** By allowing direct access to hardware, SR-IOV eliminates the need for hypervisor-based network processing, reducing CPU usage and improving data transfer speeds.

- **Scalability:** SR-IOV enables efficient sharing of NIC resources among multiple workloads, optimizing resource utilization in multi-tenant environments.

- **Reduced Latency:** Since VFs can communicate directly with the physical network hardware, packet processing is significantly faster than software-based networking approaches.
- **Enhanced Security:** SR-IOV ensures isolation between different VFs, reducing the risk of network attacks between tenants in cloud environments.

Despite these advantages, SR-IOV also has limitations. It requires hardware support from the NIC and firmware, as well as compatibility with the operating system and hypervisor. Additionally, SR-IOV can be less flexible compared to software-based solutions, as VFs are statically assigned and cannot be dynamically reconfigured as easily as virtual network interfaces.

2.5 Open Virtual Network

Open Virtual Network (OVN) extends Open vSwitch (OVS) to create a distributed network virtualization system that offers logical switches, routers, and distributed ACLs capable of spanning multiple hypervisors, container hosts, and bare-metal servers. Unlike traditional setups where bridges are configured on each host individually, OVN orchestrates the entire network as a collection of unified logical components. This abstraction significantly reduces manual configuration, as administrators can define logical networks at a high level, and OVN handles the complexity of mapping them onto physical infrastructure.

OVN's architecture revolves around two key databases known as the *Northbound* and *Southbound* databases. The Northbound Database contains high-level network definitions, such as logical switches, logical routers, and security policies defined by administrators or automation tools. The *ovn-northd* daemon translates these definitions into flow-based instructions, then writes them into the Southbound Database. Each node in the cluster runs a local OVN controller, which retrieves these low-level flow rules and enforces them by configuring the local OVS instance.

When a new container or virtual machine appears, OVN registers its network configuration in the Northbound Database. The daemon *ovn-northd* maps the logical topology onto physical resources by assigning IP addresses, setting up any required routing rules, and generating tunnel endpoints if necessary. If two containers exist on different hosts, the communication is encapsulated using protocols like Geneve or VXLAN, allowing OVN to forward traffic without requiring a centralized gateway. Upon arriving at the destination host, packets are decapsulated and delivered to the target logical switch or router, ensuring seamless connectivity across distributed environments.

A key advantage of OVN lies in its automated adaptation to configuration changes. When subnets, security groups, or network policies change, the OVN

controllers dynamically update the local flow rules, guaranteeing that network traffic is always routed according to the latest logical definitions. This process eliminates the need for manual updates to multiple devices across the data center, significantly reducing misconfigurations. OVN also integrates natively with IPv6, enabling modern addressing schemas to be used transparently within logical networks.

Because OVN relies on Open vSwitch, all OVS offloading capabilities remain available. Flow rules designed for high-bandwidth or latency-sensitive applications can be offloaded to SmartNICs, relieving the CPU from repetitive packet processing tasks. This feature becomes crucial when operating large-scale deployments where east-west traffic dominates, as the overhead of software-based switching can degrade performance. Hardware offloading, combined with OVN's distributed control plane, results in a powerful architecture for data centers seeking both efficiency and scalability.

Notably, OVN fits well into container orchestration platforms. Kubernetes clusters that require on-demand provisioning of logical networks or advanced network policies can benefit from OVN's approach to dynamic flow management. Certain projects, such as Kube-OVN, merge Kubernetes' CNI (Container Network Interface) model with OVN's distributed control plane. By doing so, they enable pods to be attached seamlessly to logical networks orchestrated by OVN. This synergy will be explored in more detail in the subsequent sections, illustrating how Kube-OVN combines Kubernetes resource management with OVN's powerful SDN features to provide a cohesive network virtualization layer.

2.6 Kubernetes

Kubernetes [2] is an open-source container orchestration system for automating software deployment, scaling, and management.

Kubernetes has become an important component in modern data centers, providing a robust and scalable platform for orchestrating containerized workloads. Its ability to automate application deployment, scaling, and management has led to widespread adoption by enterprises, cloud providers, and high-performance computing environments. Kubernetes enables organizations to manage distributed systems efficiently, ensuring high availability and optimized resource utilization.

As data center workloads grow increasingly complex, Kubernetes is crucial in integrating networking solutions and supporting multi-cloud and hybrid-cloud deployments. Through plugins and APIs, the platform's extensibility allows for seamless integration with networking and security frameworks, making it a versatile choice for large-scale infrastructures.

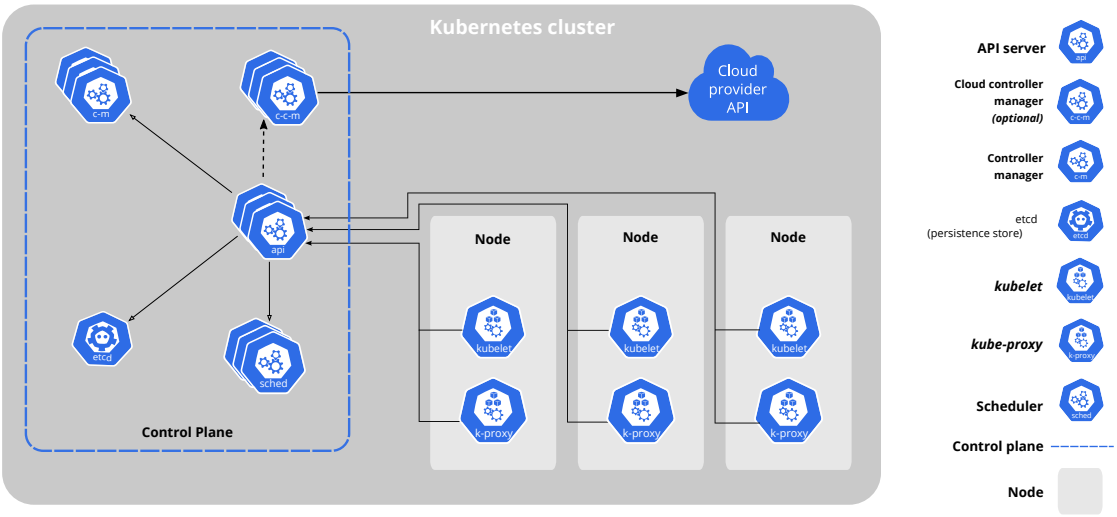

Figure 2.1: A Kubernetes cluster.

2.6.1 Kubernetes Architecture

The architecture of Kubernetes follows a distributed model composed of multiple components that work together to manage containerized workloads. At a high level, Kubernetes consists of the following key components:

- **Control Plane**: The control plane is responsible for managing the overall cluster state, making scheduling decisions, and maintaining desired application configurations. It consists of components such as the API Server, Controller Manager, Scheduler, and etcd (a distributed key-value store).
- **Worker Nodes**: Each worker node runs containerized applications and is managed by the control plane. Nodes include essential components such as the kubelet (agent responsible for managing container execution), the container runtime (such as Docker or containerd), and the kube-proxy (responsible for networking and service discovery).
- **Networking and Storage**: Kubernetes provides a pluggable networking model that allows integration with various networking solutions. Storage solutions are also managed dynamically using Persistent Volumes and Storage Classes.

A crucial aspect of Kubernetes networking is the Container Network Interface (CNI), a standardized framework that facilitates the configuration of networking for containers. CNI is designed to provide a dynamic and extensible approach to networking in containerized environments. Instead of relying on static networking

configurations, CNI allows containers to be connected to different network topologies based on application requirements.

CNI abstracts networking complexities by providing a standardized API for container orchestrators like Kubernetes to assign IP addresses, configure routes, and enforce policies dynamically. This enables seamless network integration, ensuring that workloads can communicate efficiently regardless of their underlying infrastructure. The framework supports a wide range of networking architectures, from simple bridge networks to more advanced overlay and underlay networking solutions.

One of the key advantages of CNI is its modularity, allowing administrators to swap out or extend networking solutions based on their specific needs. By decoupling the networking stack from the container runtime, CNI enhances flexibility, scalability, and interoperability across different environments, making it a foundational component of Kubernetes networking.

2.6.2 Kube-OVN

Kube-OVN [3] is a comprehensive Container Network Interface (CNI) implementation for Kubernetes that integrates Open vSwitch (OVS) to provide advanced networking capabilities, including overlay networking, security policies, and network segmentation. Designed to enhance Kubernetes networking, Kube-OVN combines the flexibility of software-defined networking (SDN) with the performance benefits of hardware acceleration, making it an ideal choice for cloud-native applications and high-performance workloads.

One of the key advantages of Kube-OVN is its ability to support both underlay and overlay networks, allowing for efficient routing and communication between pods, nodes, and external services. By leveraging OVS, Kube-OVN provides fine-grained network control, enabling administrators to define policies for traffic shaping, Quality of Service (QoS), and security enforcement. Additionally, Kube-OVN supports integration with hardware offloading technologies, such as SmartNICs and SR-IOV, to further improve network performance by offloading processing tasks from the CPU to dedicated networking hardware.

2.6.3 Multus

Multus [4] is a Kubernetes networking plugin that enables the use of multiple network interfaces in a single pod. By default, Kubernetes assigns a single network interface to each pod, but in many high-performance and network-sensitive applications, additional interfaces are needed to support specialized networking requirements.

Multus extends Kubernetes' Container Network Interface (CNI) capabilities,

allowing pods to connect to multiple networks simultaneously. This is particularly beneficial for scenarios such as network function virtualization (NFV), storage networking, and high-performance computing (HPC), where separating control, data, and management traffic onto different interfaces can enhance performance and security.

When combined with SmartNICs, Multus enables advanced networking configurations, including direct hardware-accelerated paths for traffic-intensive applications. By allowing network traffic to bypass the kernel and be processed directly on the NIC, Multus can further reduce latency and CPU overhead in Kubernetes environments. This makes it an essential tool for optimizing the networking stack in cloud-native infrastructures, particularly in data centers leveraging SmartNIC technology.

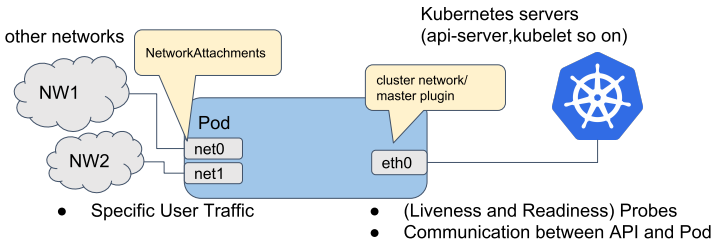

Figure 2.2: Multiple network interfaces attached to a single pod with Multus.

2.6.4 SR-IOV network device plugin

The SR-IOV Network Device Plugin [5] is Kubernetes device plugin for discovering and advertising networking resources in the form of:

- SR-IOV virtual functions (VFs)
- PCI physical functions (PFs)
- Auxiliary network devices, in particular Subfunctions (SFs)

which are available on a Kubernetes host.

2.7 Google Online boutique

Google Online Boutique [6] is an open-source microservices-based e-commerce application designed to simulate real-world cloud-native workloads. It consists of multiple independent services, each responsible for a specific function, such as

Service	Language	Description
frontend	Go	Exposes an HTTP server to serve the website. Does not require signup/login and generates session IDs for all users automatically.
cartservice	C#	Stores the items in the user's shopping cart in Redis and retrieves it.
productcatalogservice	Go	Provides the list of products from a JSON file and ability to search products and get individual products.
currencyservice	Node.js	Converts one money amount to another currency. Uses real values fetched from European Central Bank. It's the highest QPS service.
paymentservice	Node.js	Charges the given credit card info (mock) with the given amount and returns a transaction ID.
shippingservice	Go	Gives shipping cost estimates based on the shopping cart. Ships items to the given address (mock).
emailservice	Python	Sends users an order confirmation email (mock).
checkoutservice	Go	Retrieves user cart, prepares order and orchestrates the payment, shipping and the email notification.
recommendationservice	Python	Recommends other products based on what's given in the cart.
adservice	Java	Provides text ads based on given context words.
loadgenerator	Python/Locust	Continuously sends requests imitating realistic user shopping flows to the frontend.

Table 2.1: List of services in the Google Online Boutique, their programming languages, and descriptions.

frontend rendering, product catalog management, user authentication, and payment processing. These services communicate over the network using HTTP and gRPC, making the application a useful benchmark for evaluating networking performance in cloud environments.

The modular nature of Google Online Boutique makes it an ideal testbed for

studying the impact of hardware acceleration technologies like SmartNICs. Since microservices often generate a significant amount of east-west traffic within a data center, optimizing network performance through offloading mechanisms can lead to substantial efficiency gains. By analyzing network latency, throughput, and CPU utilization when running Google Online Boutique with and without SmartNIC acceleration, researchers can assess the practical benefits of offloading network processing from the host CPU to dedicated hardware.

This application is also widely used to evaluate the performance of service meshes and Kubernetes networking configurations. Integrating SmartNICs into such environments allows for an in-depth analysis of how offloading affects containerized networking, application response times, and overall system scalability.

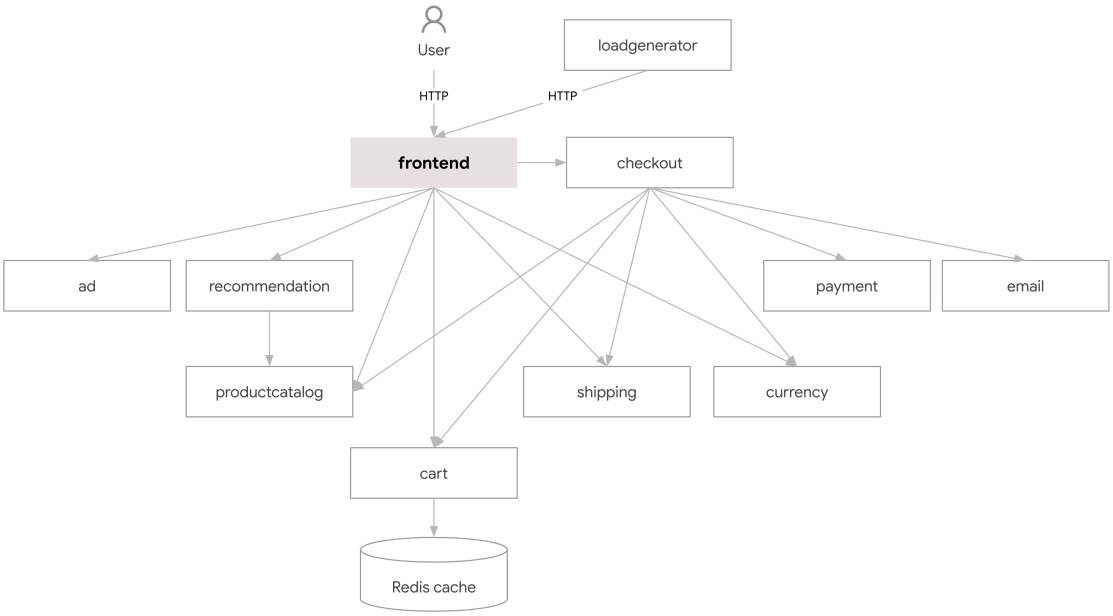

Figure 2.3: Google online boutique pods schema.

2.8 Related Works

The study of SmartNICs and hardware offloading has gained significant attention in recent years due to the increasing need for high-performance networking in cloud data centers, network function virtualization (NFV), and secure communications. This section reviews key research contributions related to SmartNIC architectures, their role in accelerating network functions, and the impact on data center performance.

Kfoury et al. [7] present a comprehensive survey on SmartNICs, providing an

extensive classification of different SmartNIC architectures, including ASIC-based, FPGA-based, and SoC-based implementations. Their work highlights the primary use cases of SmartNICs, such as packet processing, security enforcement, network telemetry, and virtualization acceleration. One of their key findings is the trade-off between performance and programmability across different SmartNIC architectures, which is a crucial consideration when selecting the appropriate offloading mechanism for specific workloads.

Khan et al. [8] explore the integration of SmartNICs in Kubernetes environments, focusing on offloading network functions such as IPsec encryption, virtual switching, and tunneling. Their research demonstrates how BlueField-2 SmartNICs can significantly reduce CPU utilization by handling encryption and packet forwarding directly on the NIC, leading to improved throughput and lower network latencies. The study further discusses the potential integration of hardware offloading with Kubernetes networking plugins, particularly in high-performance computing (HPC) and NFV use cases.

Liu et al. [9] conduct a detailed performance characterization of the BlueField-2 SmartNIC, analyzing its ability to offload various network functions from the host CPU. Their results indicate that while the host CPU can saturate high-speed network links, the embedded processors on the SmartNIC exhibit performance limitations when handling kernel-space packet processing. However, operations such as cryptographic acceleration and inter-process communication achieve significant performance improvements when offloaded to the SmartNIC. This study underscores the importance of selecting the right offloading strategy based on workload requirements and hardware capabilities.

Efraim et al. [10] focus on the integration of SR-IOV with Open vSwitch (OVS), detailing recent advancements in the Linux kernel that enable efficient hardware offloading while maintaining software-defined networking (SDN) flexibility. Their work describes the challenges of managing SR-IOV-enabled devices within virtualized environments and proposes solutions that allow flow-based policy enforcement while retaining the performance benefits of direct hardware access. This approach is particularly relevant for cloud providers seeking to optimize their virtualized network stacks without sacrificing manageability.

Overall, these studies collectively emphasize the growing importance of SmartNICs in modern networking environments, demonstrating their potential to enhance performance, security, and energy efficiency. However, they also highlight key challenges such as programmability, integration complexity, and workload-specific optimizations, which must be carefully considered when deploying SmartNIC-based solutions in production environments.

Chapter 3

SmartNICs

Smart Network Interface Cards (SmartNICs) represent a significant advancement in networking hardware, enabling the offloading of various network functions from the host CPU to dedicated processing units within the NIC. These specialized network cards enhance performance, reduce CPU utilization, and improve efficiency in data centers, cloud infrastructures, and high-performance computing environments. SmartNICs are typically categorized into three main types based on their architecture: **ASIC-based**, **SoC-based**, and **FPGA-based** implementations. Each type offers distinct advantages and trade-offs, making them suitable for different use cases and workload demands.

3.1 ASIC-based

Application-Specific Integrated Circuit (ASIC)-based SmartNICs are designed with purpose-built hardware to accelerate specific network functions. These SmartNICs offer high efficiency and low power consumption but are less flexible compared to other architectures. They are optimized for tasks such as packet processing, load balancing, and network security enforcement, providing deterministic performance with minimal overhead.

One of the most widely used ASIC-based SmartNICs is the **NVIDIA ConnectX-7**, which is specifically engineered for high-throughput, low-latency networking applications.

3.1.1 NVIDIA ConnectX-7

The **NVIDIA ConnectX-7** is an advanced ASIC-based SmartNIC designed to provide high-speed networking with hardware acceleration capabilities. While it supports up to **400Gbps** of network bandwidth, a **100Gbps** version was used

during testing. It includes features such as **RDMA over Converged Ethernet (RoCE)**, **GPUDirect** for optimized data transfer in AI workloads, and built-in support for **IPsec and TLS encryption offloading**.

Key capabilities of the ConnectX-7 include:

- **Hardware-accelerated packet processing:** Reduces CPU load by offloading network tasks such as tunneling, encryption, and flow steering.
- **Dynamic congestion control:** Enhances network efficiency by reducing packet loss and improving throughput in high-performance environments.
- **SR-IOV and VirtIO support:** Enables direct assignment of virtual network functions to VMs and containers for improved performance in cloud-native architectures.

ConnectX-7 is widely adopted in cloud data centers, HPC environments, and AI workloads due to its ability to accelerate high-bandwidth, low-latency communications while ensuring secure and efficient data movement.

3.2 SoC-based

System-on-Chip (SoC)-based SmartNICs integrate general-purpose processing units, such as ARM cores, alongside dedicated networking hardware. This architecture allows SmartNICs to support programmable offloading, making them highly adaptable to a variety of network functions, including virtual switching, firewalling, and deep packet inspection.

3.2.1 NVIDIA BlueField Family

The **NVIDIA BlueField** family represents a series of SoC-based SmartNICs that combine high-performance networking with embedded computing capabilities. BlueField SmartNICs include multiple **ARM cores**, a programmable data path, and dedicated accelerators for encryption, storage, and AI-driven analytics.

Key features of BlueField SmartNICs include:

- **DPU (Data Processing Unit) Architecture:** Allows for offloading of security functions, telemetry, and AI-driven network monitoring.
- **Integration with Kubernetes and SDN solutions:** Provides seamless support for containerized and virtualized workloads.
- **Zero-Trust Security Enforcement:** Enables advanced access control and microsegmentation directly at the SmartNIC level.

The BlueField SmartNICs are particularly well-suited for use in cloud computing, cybersecurity applications, and AI-driven analytics, providing extensive programmability and offloading capabilities.

3.3 FPGA

Field-Programmable Gate Array (FPGA)-based SmartNICs offer the highest degree of programmability, enabling the development of custom network functions optimized for specific workloads. Unlike ASIC-based solutions, which are fixed in functionality, FPGA SmartNICs can be reprogrammed dynamically to accommodate evolving network requirements.

FPGA-based SmartNICs are commonly used for:

- **Custom network acceleration:** Tailored to specific applications such as financial trading platforms, where ultra-low latency is required.
- **High-performance packet filtering and inspection:** Ideal for cybersecurity and deep packet inspection use cases.
- **Real-time data analytics and monitoring:** Provides the ability to process and analyze network traffic with minimal delay.

While FPGA SmartNICs offer unmatched flexibility, they tend to have higher power consumption and require expertise in hardware programming for optimal utilization. However, their ability to be reconfigured makes them a valuable tool for organizations that need adaptable, high-performance networking solutions.

Overall, SmartNICs are transforming modern networking by offloading critical functions from the host system, improving efficiency, and enhancing security. Their diverse architectures cater to different operational needs, making them a crucial component in next-generation data centers and cloud environments.

Chapter 4

Architecture and Offloading Techniques

4.1 Hardware and Software Architecture

The integration of SmartNICs in modern data centers requires a well-defined hardware and software architecture to fully leverage their offloading capabilities. The overall architecture consists of multiple components, including compute nodes, network interfaces, and software stacks optimized for offloading network functions.

4.1.1 Hardware Components

The hardware architecture includes the following key elements:

- **Compute Nodes:** High-performance servers equipped with SmartNICs, supporting both virtualized and containerized workloads.
- **SmartNICs:** Network interface cards capable of offloading processing tasks such as packet switching, encryption, and tunneling.
- **Network Infrastructure:** High-speed interconnects, including 100Gbps links using QSFP+ cables, supporting efficient data transfer between nodes.

The experiments were conducted on two different hardware setups:

First Test Setup

- **Servers:** Two machines with different CPU configurations:
  - **Polycube-server:** Intel(R) Xeon(R) Gold 5120 CPU @ 2.20GHz, **28 cores**.

  - **Fall-server:** Intel(R) Xeon(R) Silver 4116 CPU @ 2.10GHz, **24 cores**.

- **SmartNICs:** NVIDIA ConnectX-7 (100Gbps) installed on both machines.
- **Interconnect:** Direct connection using 100Gbps QSFP+ cables.
- **PCIe Configuration:** Both servers use a motherboard with PCIe Gen 3 x8, which does not fully utilize the ConnectX-7 bandwidth, as it requires PCIe Gen 4 x16 for maximum performance.
- **Operating System:** Ubuntu Server 24.04.

Second Test Setup

- **Servers:** Two identical machines:
  - Intel(R) Xeon(R) Gold 6442Y CPU, **96 cores**.
- **SmartNICs:** NVIDIA ConnectX-7 (100Gbps) installed on both machines.
- **Interconnect:** Direct connection using 100Gbps QSFP+ cables.
- **PCIe Configuration:** Motherboards equipped with PCIe Gen 4 x16 slots, which fully support the bandwidth requirements of the ConnectX-7 SmartNICs.
- **Operating System:** Ubuntu Server 24.04.

4.1.2 Software Components

The software stack plays a crucial role in managing and configuring SmartNIC offloading. The main components include:

- **Operating System:** Ubuntu Server 24.04.
- **Open vSwitch (OVS):** A widely used virtual switch that integrates with SmartNICs for offloading virtual networking functions described in the previous chapter.
- **Kubernetes and CNI Plugins:** Networking solutions such as Multus and Kube-OVN enable advanced networking features for containerized workloads.
- **SR-IOV and VF Management:** Allows for direct assignment of virtual network interfaces to applications, bypassing software-defined network stacks.

4.2 Offloading Mechanisms

Offloading mechanisms refer to the delegation of networking tasks from the host CPU to specialized hardware within SmartNICs. This approach enhances performance, reduces CPU overhead, and ensures efficient resource utilization. The following types of offloading are commonly used:

- **Packet Processing Offload:** The SmartNIC processes incoming and outgoing packets, reducing software stack latency.
- **Tunnel Offloading:** Protocols such as VXLAN and Geneve are handled directly by the SmartNIC, bypassing the host CPU.
- **Security Offload:** IPsec and TLS encryption/decryption are executed on the SmartNIC to improve cryptographic performance.
- **Virtual Switching Offload:** Open vSwitch flow rules are offloaded to SmartNIC hardware, reducing CPU-bound switching tasks.

4.2.1 Offload on ConnectX-7

The **NVIDIA ConnectX-7** provides advanced offloading capabilities, enabling high-performance networking with minimal CPU involvement. Key offloading features include:

- **OVS Hardware Offload:** Flow-based packet processing is executed directly on the SmartNIC, reducing the need for software switching.
- **VXLAN and Geneve Acceleration:** SmartNIC-integrated tunneling support for high-speed overlay networking.
- **SR-IOV Virtual Functions (VFs):** Allows namespaces, virtual machines and Kubernetes pods to bypass the hypervisor, achieving near-native performance.
- **IPsec Offloading:** Hardware-accelerated encryption and decryption improve secure communications while maintaining high throughput.

These features make the ConnectX-7 a strong candidate for data-intensive workloads, where traditional CPU-based processing may become a bottleneck.

4.2.2 Offloading Implementation

The implementation of SmartNIC offloading involves multiple steps, ensuring that network traffic is properly redirected to the NIC hardware for processing. The key steps include:

1. **SR-IOV Configuration:** Enable Virtual Functions (VFs) on the SmartNIC and assign them to applications or containers.

2. **Integration with Open vSwitch:** Configure OVS with hardware offloading enabled to allow flow-based acceleration.

3. **Kubernetes Networking Setup:** Use Multus to attach multiple network interfaces to pods, leveraging hardware acceleration.

4. **Security and Performance Optimization:** Enable cryptographic acceleration for IPsec workloads and monitor performance using benchmarking tools.

Hardware Offload Workflow

Offloading in SmartNICs follows a well-defined workflow that involves traffic redirection from the CPU to the NIC hardware. This process is managed by several key components [10]:

1. Representor Ports: In SR-IOV environments, each Virtual Function (VF) is associated with a *representor port*. This representor acts as a proxy between the VF and the SmartNIC's embedded switch (e-switch). The representor port allows the host system to monitor, control, and offload network flows at the hardware level.

2. Devlink and Switchdev: The Linux kernel provides tools such as *devlink* and *switchdev* to configure hardware offloading. Devlink enables interaction with the SmartNIC firmware, while switchdev allows for flow-based rule programming that offloads switching decisions to the SmartNIC rather than the host CPU.

3. TC (Traffic Control) Offloading: The *Traffic Control (TC)* subsystem is used to manage network traffic and can offload rules to SmartNICs. With TC Flower filters, administrators can define flow-based policies, which are then pushed down to the hardware, allowing efficient forwarding, VLAN tagging, and tunneling operations such as VXLAN and Geneve encapsulation.

4. Hybrid Offloading Approach: Not all traffic is offloaded immediately. The system follows a *hybrid approach*, where:

- The first packet of a flow is handled by the software-based OVS, which determines the forwarding decision.

- If the flow is suitable for offloading, the flow rule is installed in the SmartNIC hardware via TC.
- Subsequent packets follow the hardware-defined path, reducing CPU involvement and improving performance.

Practical Implementation in OVS

To integrate these offloading mechanisms into Open vSwitch, the following steps are performed:

- Configure the SmartNIC's **e-switch** to operate in **switchdev mode**, allowing hardware-accelerated switching.
- Use **devlink** to set the appropriate operating mode for SR-IOV and enable representor ports.
- Establish **TC Flower** rules to direct traffic flows efficiently while enabling hybrid offloading.
- Monitor *flow aging* and offloaded counters to ensure the SmartNIC efficiently manages active flows.

IPsec Offload

IPsec full offload enables cryptographic operations to be handled entirely by the SmartNIC, significantly reducing the CPU overhead associated with encryption and decryption.

The packet processing in IPsec offloading follows these steps [11]:

1. Incoming packets are received by the SmartNIC, which checks if they match an existing Security Association (SA).
2. If a matching SA is found, the SmartNIC performs decryption, authentication, and integrity verification directly in hardware.
3. The decrypted packet is then forwarded to the host system or switched through the SmartNIC's embedded switch, depending on the configuration.
4. For outgoing traffic, the SmartNIC encrypts packets and applies authentication headers before transmission.
5. The processed packets are then forwarded directly to the network, bypassing the host CPU.

This approach ensures minimal latency and improved throughput for encrypted communication, making it ideal for high-performance and secure networking environments.

By combining these techniques, SmartNICs can offload large portions of network processing, significantly reducing CPU load and improving throughput for high-performance workloads.

Chapter 5

Experimental Evaluation

This section presents the configurations of tests conducted to evaluate the performance impact of SmartNIC hardware offloading in different networking scenarios. The experiments were designed to compare traditional CPU-based packet processing with offloading mechanisms provided by the NVIDIA ConnectX-7 SmartNIC. The goal is to assess improvements in throughput, latency, and CPU utilization when leveraging hardware acceleration.

As shown in chapter 4, two server setups were employed:

- *First Setup* (older hardware, Gen 3 x8)
- *Second Setup* (newer hardware, Gen 4 x16)

5.1 Open vSwitch Hardware Offload Tests

The first set of experiments focuses on Open vSwitch (OVS) hardware offloading. OVS is widely used in virtualized and containerized environments to manage network traffic, and its performance can be significantly improved by leveraging SmartNIC offload capabilities.

OVS was configured in two modes: In the first all packet processing is performed in software by the host CPU. In the second flow rules are offloaded to the SmartNIC, reducing CPU overhead. Each test was executed multiple times to ensure consistent and reproducible results.

5.1.1 Virtual Ethernet vs SR-IOV

Figures 5.3 and 5.4 focus on the first setup, illustrating CPU usage and throughput with `iperf3` TCP tests across varying numbers of parallel streams. Although

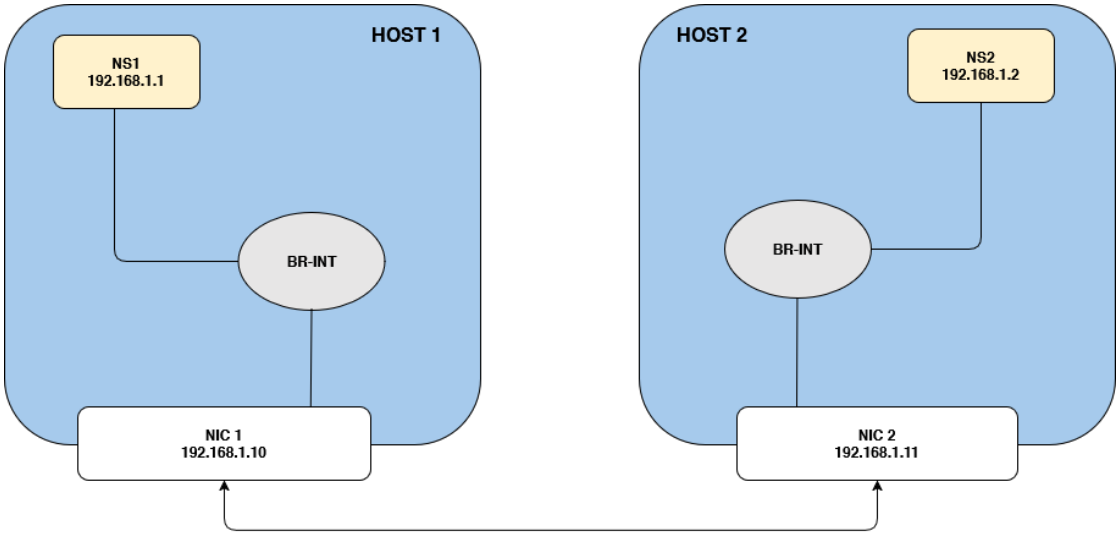

Figure 5.1: Configuration for the test without offload.

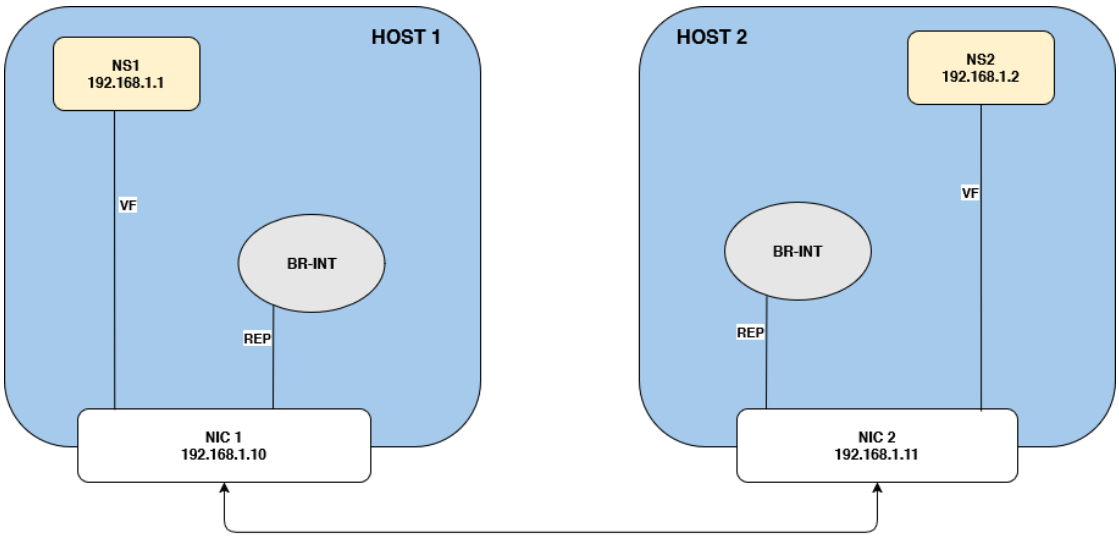

Figure 5.2: Configuration for the test with offload.

throughput often levels off around 64Gbps, SR-IOV offloading significantly reduces CPU utilization when many streams are active.

Figure 5.3 shows that, at low stream counts, CPU usage remains modest irrespective of configuration. As concurrency rises (12 or 24 streams), the *non-offloaded* client setup experiences a steep climb in CPU consumption, whereas SR-IOV offloading maintains lower overhead across the board. Meanwhile, Figure 5.4 indicates that both approaches start with comparable throughput, yet SR-IOV

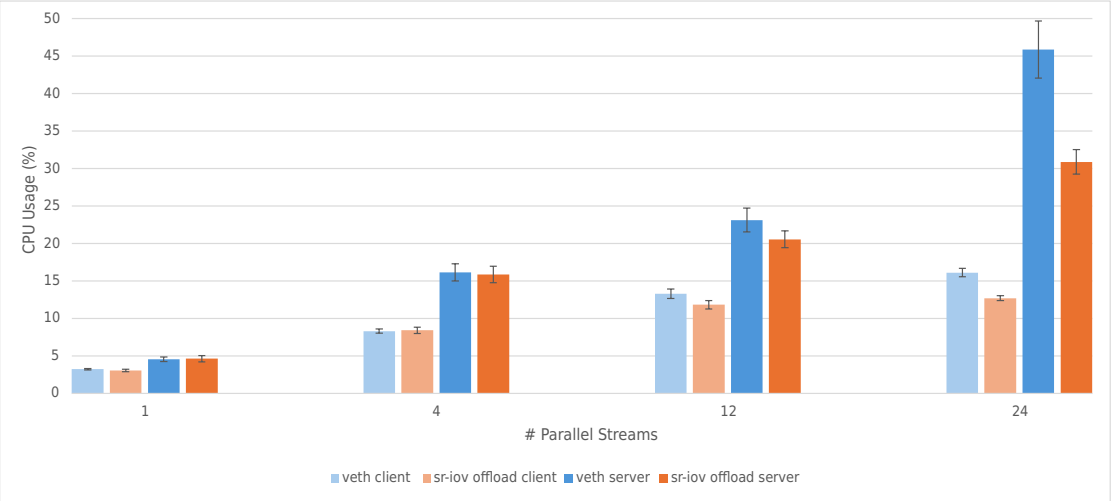

Figure 5.3: CPU usage for different numbers of streams on the first setup, without encapsulation or encryption.

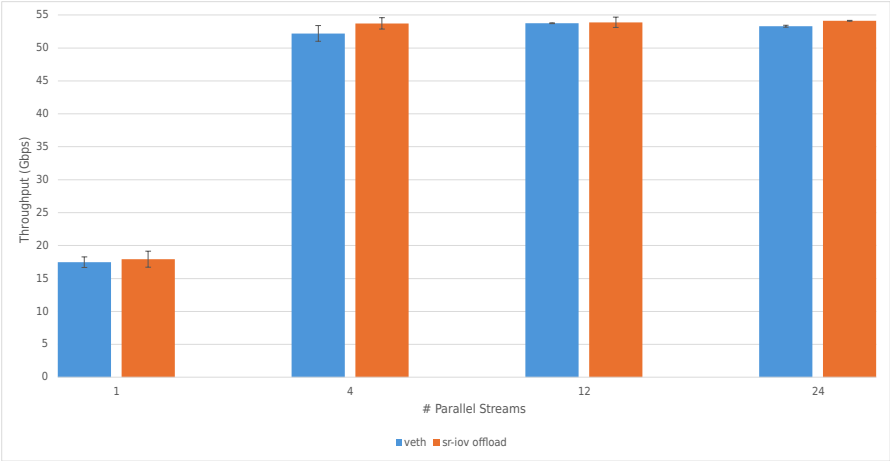

Figure 5.4: Throughput for different numbers of streams on the first setup, without encapsulation or encryption.

stays near 64Gbps under heavier loads, stabilizing performance despite hitting PCIe limits.

To determine if these observations arise from hardware constraints, the same tests were run on the *second setup*, which eliminates the PCIe bottleneck and provides more CPU resources. Figures 5.5 and 5.6 summarize outcomes on the newer machines, revealing that ConnectX-7 can achieve near line-rate operation and alleviate CPU utilization more effectively under higher concurrency.

In Figure 5.5, offloading continues to yield tangible CPU savings as parallel

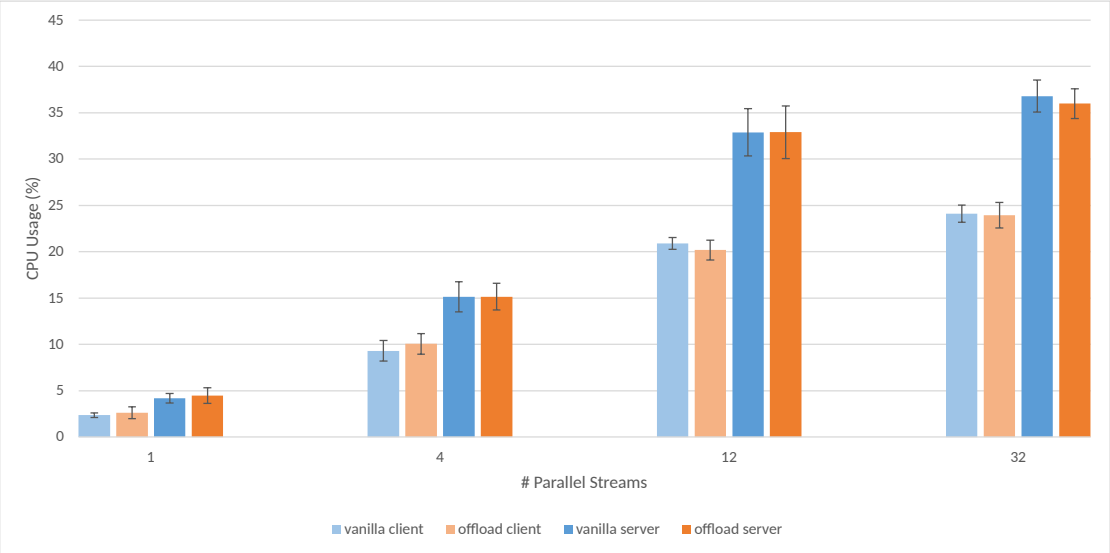

Figure 5.5: CPU usage for different numbers of streams on the second setup, without encapsulation or encryption.

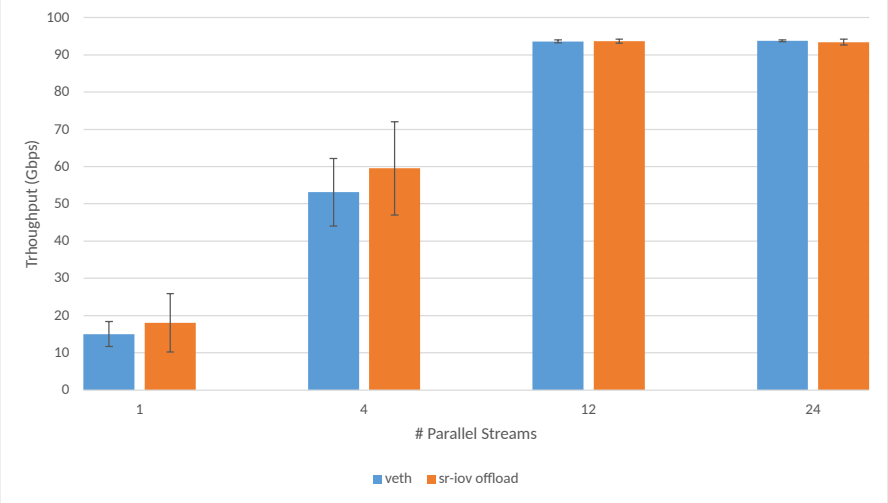

Figure 5.6: Throughput for different numbers of streams on the second setup, without encapsulation or encryption.

streams increase. According to Figure 5.6, the newer setup allows throughput to climb closer to the SmartNIC's maximum, confirming that removing PCIe bottlenecks and employing a robust CPU design helps SR-IOV maintain high data rates without saturating the host CPU.

Overall, this *veth vs. SR-IOV* comparison highlights that offloading consistently

relieves CPU pressure, although the extent of those improvements can differ between the two setups. On older hardware, SR-IOV provides a noticeable advantage by sustaining a steady 64Gbps under heavier concurrency while significantly reducing CPU load. In the second setup, which removes the PCIe bottleneck and benefits from more powerful CPUs, offloading still lowers CPU usage, but the difference compared to the veth configuration is not as pronounced. The newer servers are inherently more capable of handling high data rates, so while SR-IOV helps, the net gains in performance or CPU savings appear to be more modest.

In both environments, the core takeaway is that offloading does not always boost raw TCP throughput beyond the available hardware limits but does allow higher concurrency without disproportionately taxing the CPU. When system resources align closely with the SmartNIC's capabilities, offloading can yield predictable performance at scale with minimal overhead. By contrast, when hardware resources are constrained, SR-IOV often demonstrates clearer advantages in keeping CPU usage under control.

5.1.2 Geneve Tunneling Encapsulation

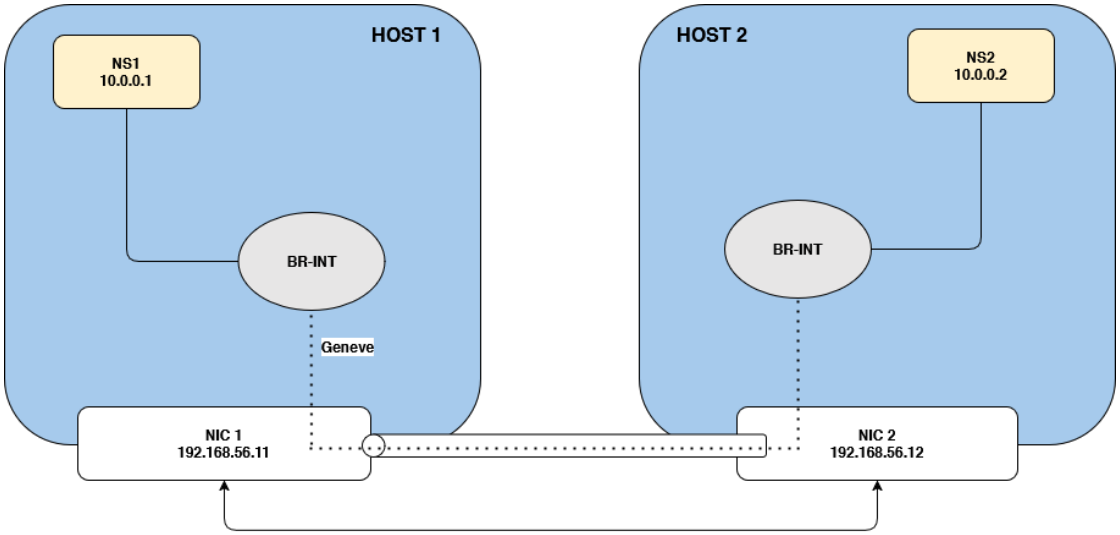

Figure 5.7: Configuration for the test with Geneve encapsulation and without offload.

A further set of tests was conducted to assess how Geneve tunneling impacts offloading results under the *first setup* (older hardware) and the *second setup* (newer, more capable servers). Geneve introduces an additional layer of encapsulation, thereby adding overhead in the packet processing path. The experiments aim to

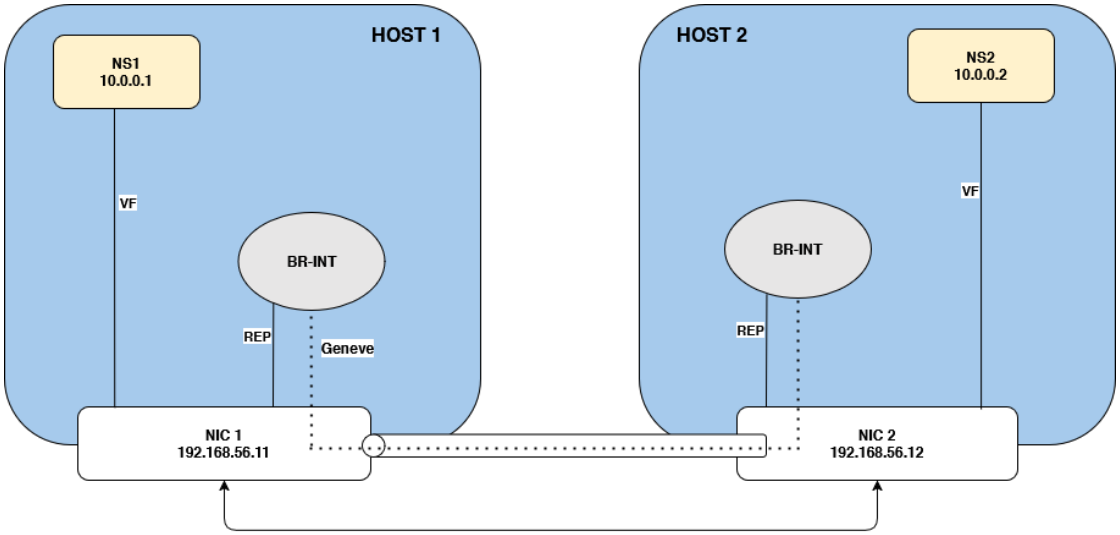

Figure 5.8: Configuration for the test with Geneve encapsulation and offload.

determine whether SR-IOV offloading continues to reduce CPU usage or boost throughput once the complexity of tunneling is taken into account.

Figures 5.9 and 5.10 show results from the older machines. CPU usage rises noticeably for the virtual Ethernet (*non-offloaded*) configuration when parallel streams increase, largely due to the extra encapsulation overhead. SR-IOV offloading helps contain CPU consumption, especially at higher stream counts, although overall throughput remains constrained around 60–64Gbps. The key advantage of hardware offloading in this scenario is preventing the CPU from becoming overwhelmed by Geneve's tunneling operations.

Figure 5.9 highlights how *non-offloaded* configurations incur a marked CPU penalty as concurrency scales, whereas SR-IOV consistently keeps utilization lower. In Figure 5.10, both approaches follow a similar throughput profile, with SR-IOV slightly mitigating the performance dip introduced by tunneling. The net benefit is most pronounced on the older hardware, where any form of overhead is amplified by the limited PCIe bandwidth and CPU capacity.

The second setup, summarized in Figures5.11 and 5.12, differs in a surprising way from the first. At low concurrency (1 and 4 streams), offloading actually results in reduced throughput, even though CPU usage is slightly lower than in the *non-offloaded* mode. When the number of streams increases to 12 or 32, the throughput for offloaded and non-offloaded configurations becomes practically the same, yet offloading shows a marginal rise in CPU usage. These counterintuitive results suggest that, in the newer and more capable environment, some overheads in the offload path or traffic steering mechanism outweigh the typical benefits of

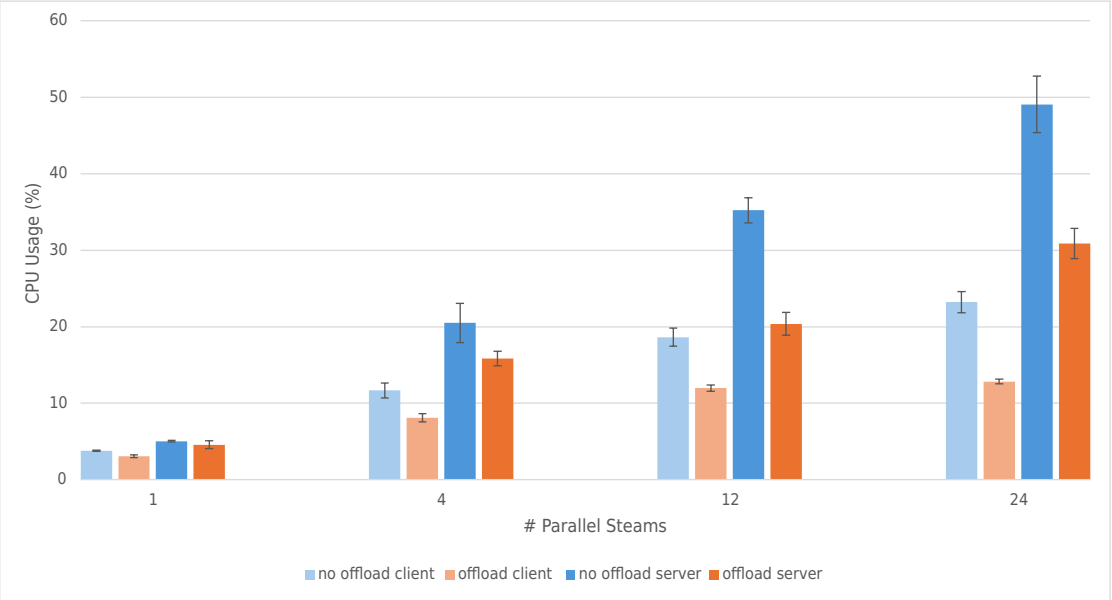

Figure 5.9: CPU usage for different numbers of streams on the first setup, with Geneve encapsulation.

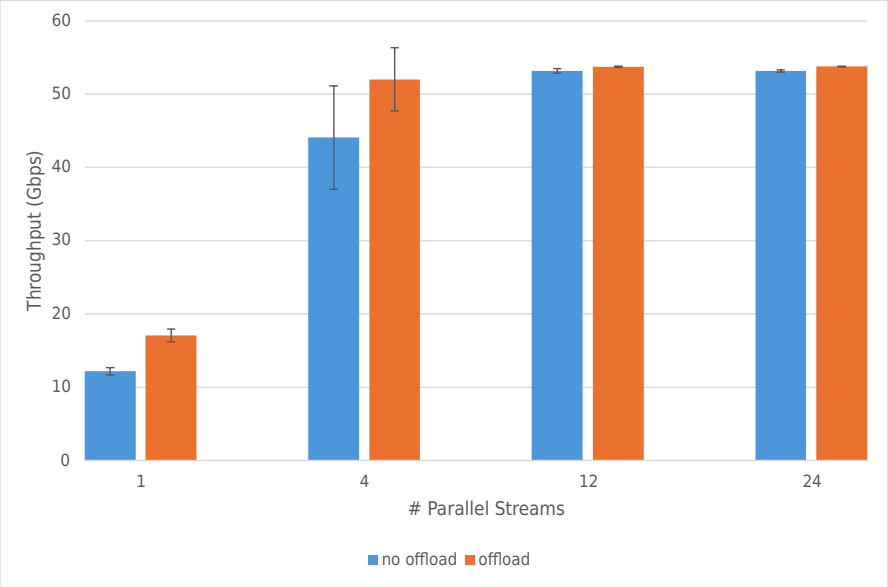

Figure 5.10: Throughput for different numbers of streams on the first setup, with Geneve encapsulation.

hardware acceleration, particularly at low stream counts, where the baseline CPU load is already minimal.

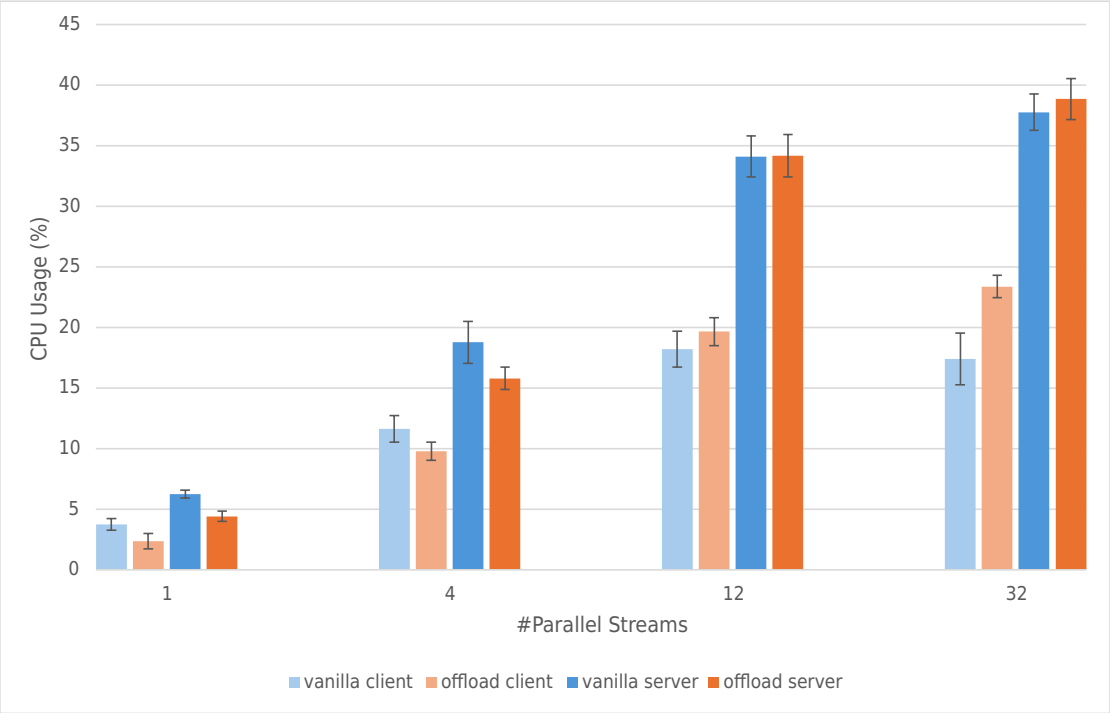

Figure 5.11: CPU usage for different numbers of streams on the second setup, with Geneve encapsulation.

In Figure 5.11, offloading reduces CPU usage at lower concurrency, but this advantage is overshadowed by the noticeable drop in throughput visible in Figure 5.12. At higher concurrency, CPU usage converges or even becomes slightly higher under SR-IOV, while throughput remains essentially the same compared to the *non-offloaded* configuration. Consequently, in contrast to the first setup, the second setup does not exhibit a clear advantage for offloading under Geneve tunneling.

One hypothesis is that the overhead of managing offloaded flows under Geneve may introduce additional control-plane or data-plane complexity that negates the typical CPU savings. The broader PCIe bandwidth and high core count may also make software-based processing sufficiently efficient, leaving less headroom for SR-IOV to provide improvements. As a result, offloading can occasionally lead to suboptimal throughput at low concurrency or a slight CPU increase at higher concurrency.

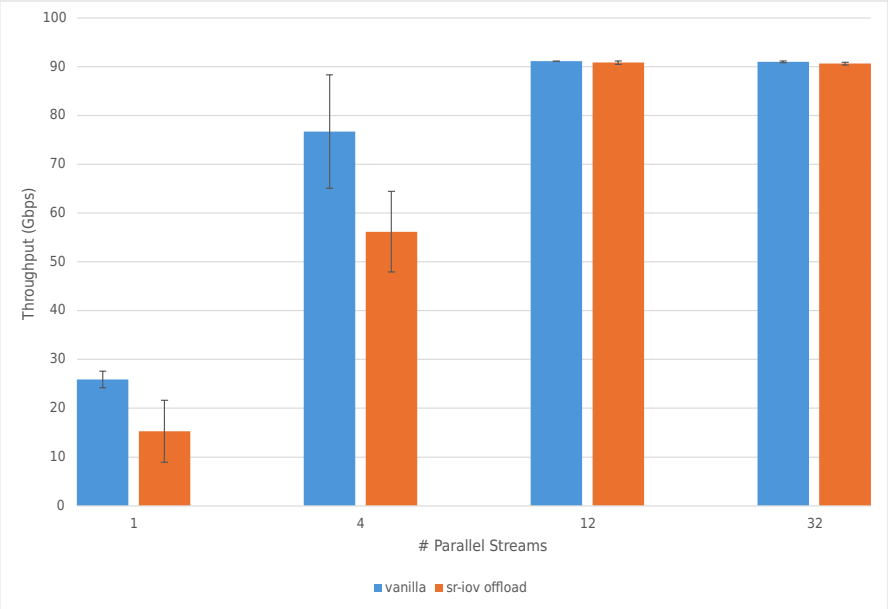

Figure 5.12: Throughput for different numbers of streams on the second setup, with Geneve encapsulation.

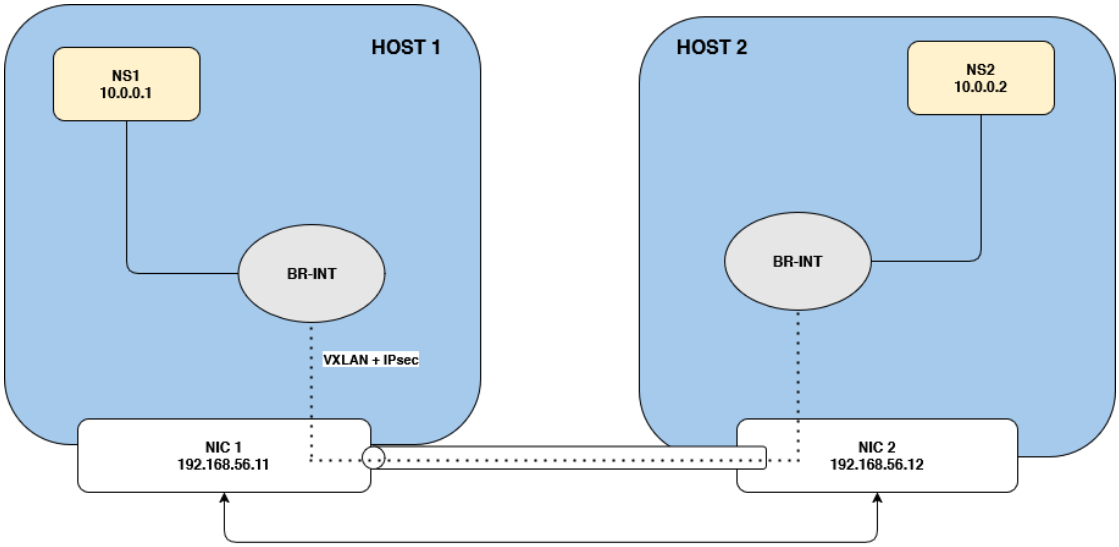

Figure 5.13: Configuration for the test with VXLAN encapsulation + IPsec encryption and without offload.

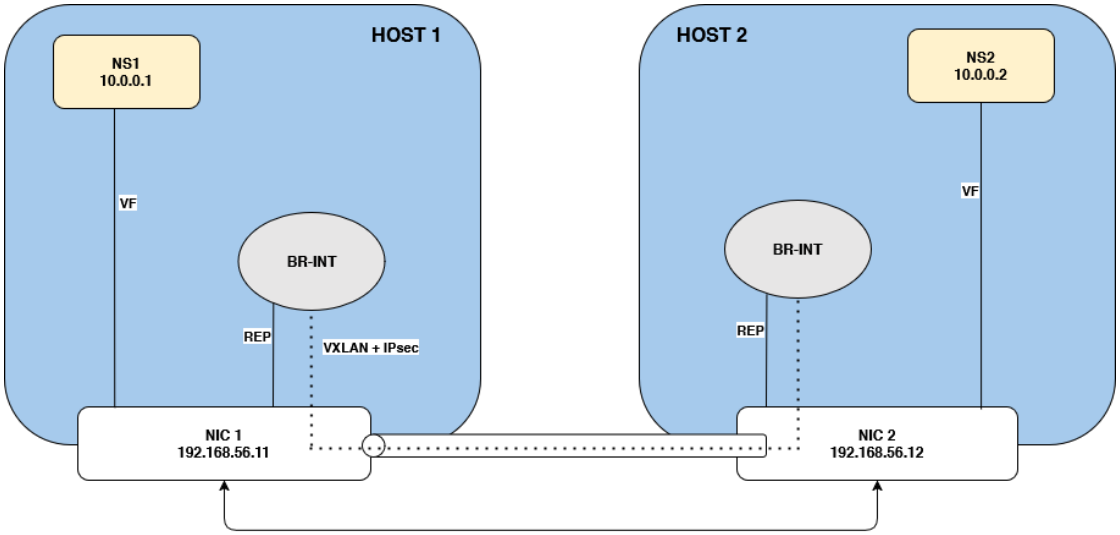

Figure 5.14: Configuration for the test with VXLAN encapsulation + IPsec encryption and offload.

5.1.3 VXLAN + IPsec Offloading

A further series of tests examined the combined effect of VXLAN tunneling and IPsec encryption. In contrast to earlier scenarios, this configuration pushes significant cryptographic overhead onto the system, which often limits throughput when handled purely by software. By moving encryption into the ConnectX-7 SmartNIC, offloading attempts to keep throughput high despite CPU constraints.

Figures 5.15 and 5.16 outline the results. Notably, offloading in this context increases CPU usage relative to the *non-offloaded* configuration; however, throughput jumps from around 2.5Gbps up to nearly 27Gbps, amounting to an **approximate 980% improvement**. This ten-fold increase in transmission speed indicates that hardware-accelerated encryption provides a much more meaningful gain than might be inferred solely from CPU usage metrics.

As shown in Figure 5.15, the offloaded configuration consumes more CPU than expected, likely due to the overhead of managing IPsec flows on the SmartNIC, along with additional control-plane interactions. However, Figure 5.16 confirms that offloading nevertheless drives throughput sharply upward, from a baseline of just a few gigabits per second to well above 20Gbps. In other words, even if CPU usage is not minimized, the system overall delivers a vastly higher packet processing rate by delegating cryptographic operations away from the host.

These findings underscore that hardware offloading is particularly valuable when the workload involves encryption or other computationally expensive functions. While network-centric offloads may not always raise throughput under simpler use

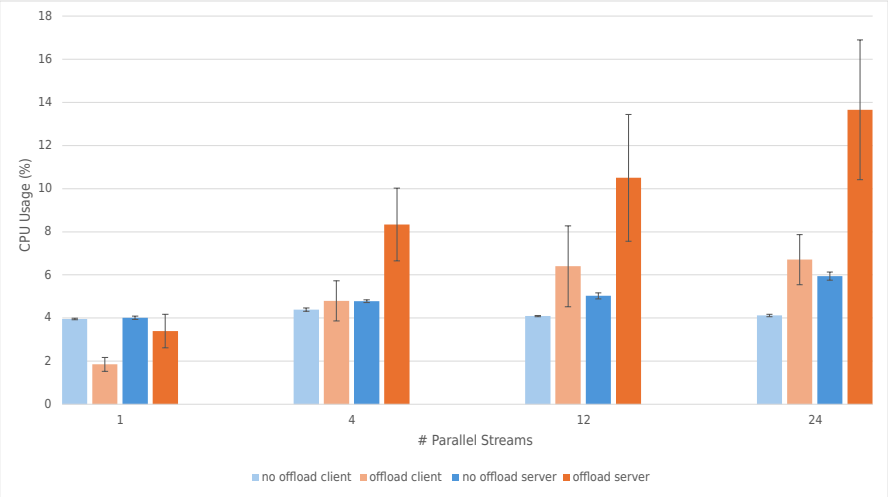

Figure 5.15: CPU usage under VXLAN + IPsec on the first setup.

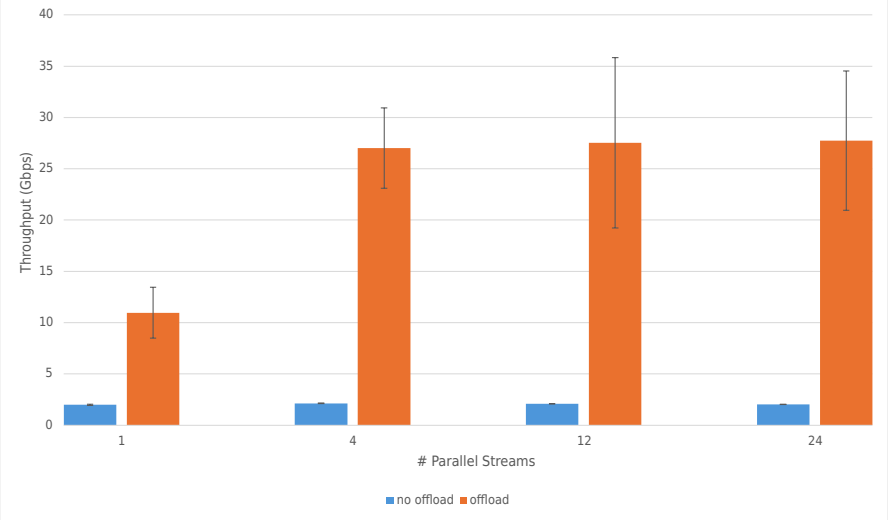

Figure 5.16: Throughput under VXLAN + IPsec on the first setup.

cases, tasks like IPsec encryption can see dramatic gains in data rate. Rather than capping out at a fraction of line speed due to software-based ciphers, offloading facilitates an order-of-magnitude improvement, allowing the system to sustain far greater concurrency while maintaining acceptable performance. In many real-world deployments, this trade-off of slightly higher CPU usage for a large throughput boost is highly beneficial, as it ensures encrypted communications remain both secure and efficient.

The same test was also performed on the second setup with broader PCIe

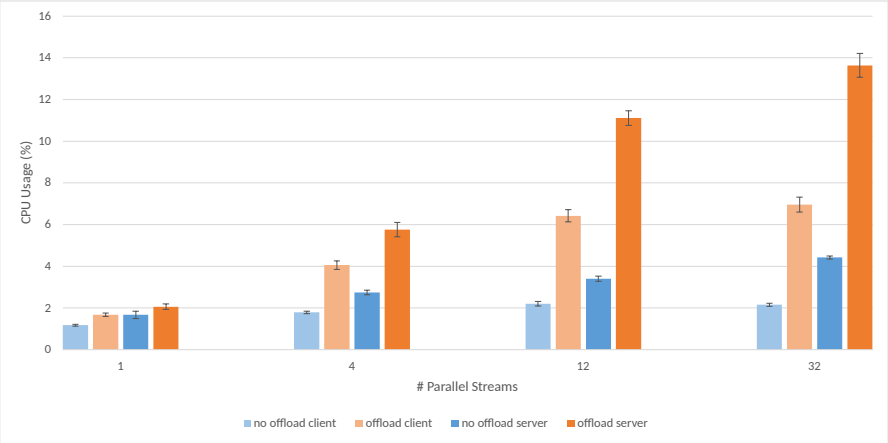

Figure 5.17: CPU usage under VXLAN + IPsec on the second setup.

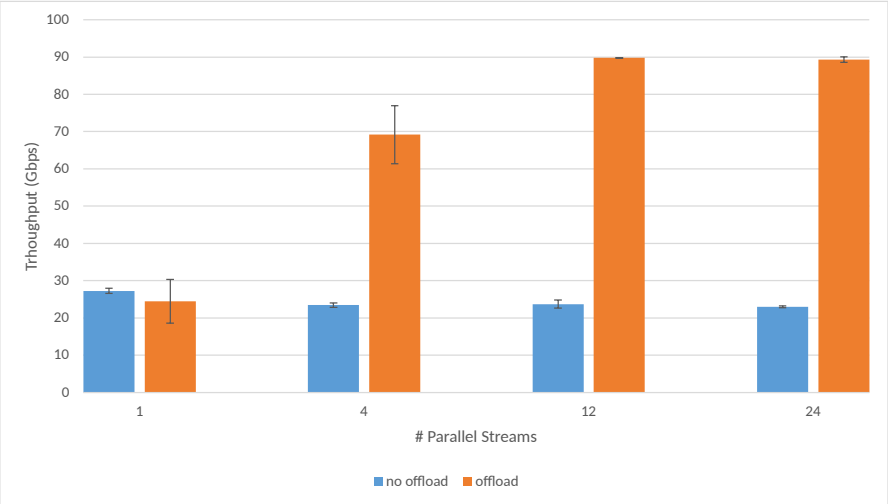

Figure 5.18: Throughput under VXLAN + IPsec on the second setup.

bandwidth and more powerful CPUs, as shown in Figures 5.17 and 5.18. Despite the hardware's ability to handle higher base throughput, IPsec offloading still leads to a noteworthy jump in performance. However, the net gain is smaller in relative terms than on the first setup, since the modern CPUs can process a fair amount of cryptographic load in software without saturating. *Non-offloaded* configurations in this environment remain viable, but offloading continues to show value by preventing CPU overhead from escalating under heavy concurrency, even if it no longer yields the tenfold increase observed on older machines.

5.2 Google Online Boutique Hardware Offload Tests

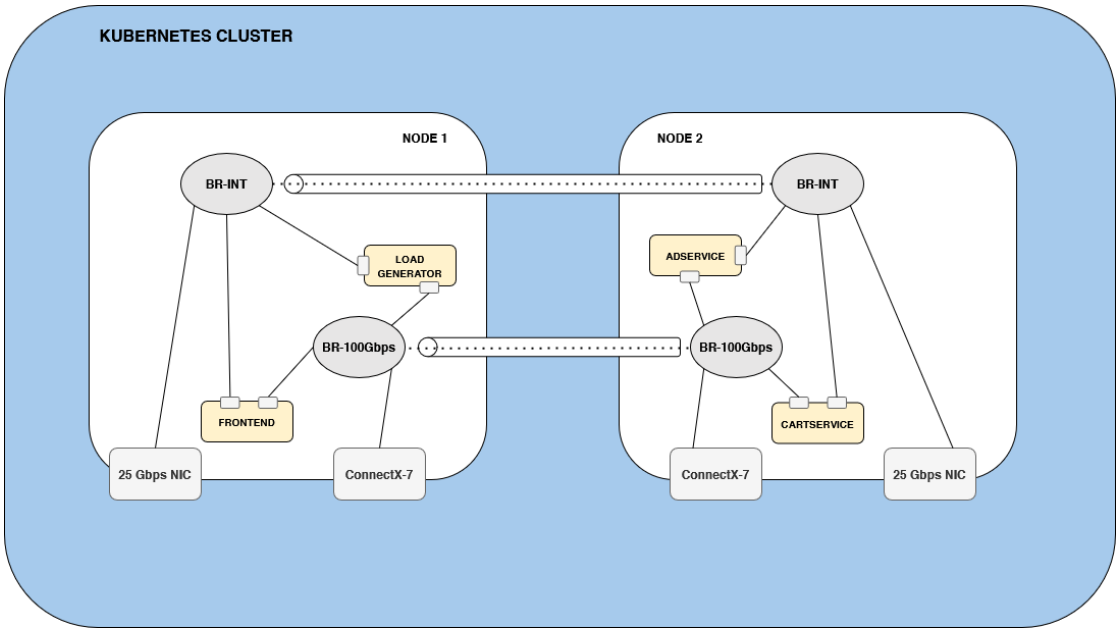

Figure 5.19: Configuration for the test with the Google Online Boutique in a Kubernetes cluster without offload.

To demonstrate the impact of offloading in a more realistic microservices environment, the Google Online Boutique demo was deployed on the *second setup*, where each server has enough CPU resources and PCIe bandwidth to handle high levels of concurrency with minimal software overhead. Pods for each microservice were distributed across the two nodes (Table 5.1) in such a way that the *frontend* and *loadgenerator* services ran on the same host, heavily stressing its resources.

Table 5.1: Pod Placement for the Google Online Boutique

Node	Microservices / Pods
node 1	frontend, loadgenerator, adservice, cartservice, checkoutservice, currencyservice, emailservice, shippingservice, redis-cart
node 2	paymentservice, productcatalogservice, recommendationservice

A custom script was added to the *loadgenerator* container, leveraging the Locust framework to generate HTTP requests toward the frontend. Because Locust itself

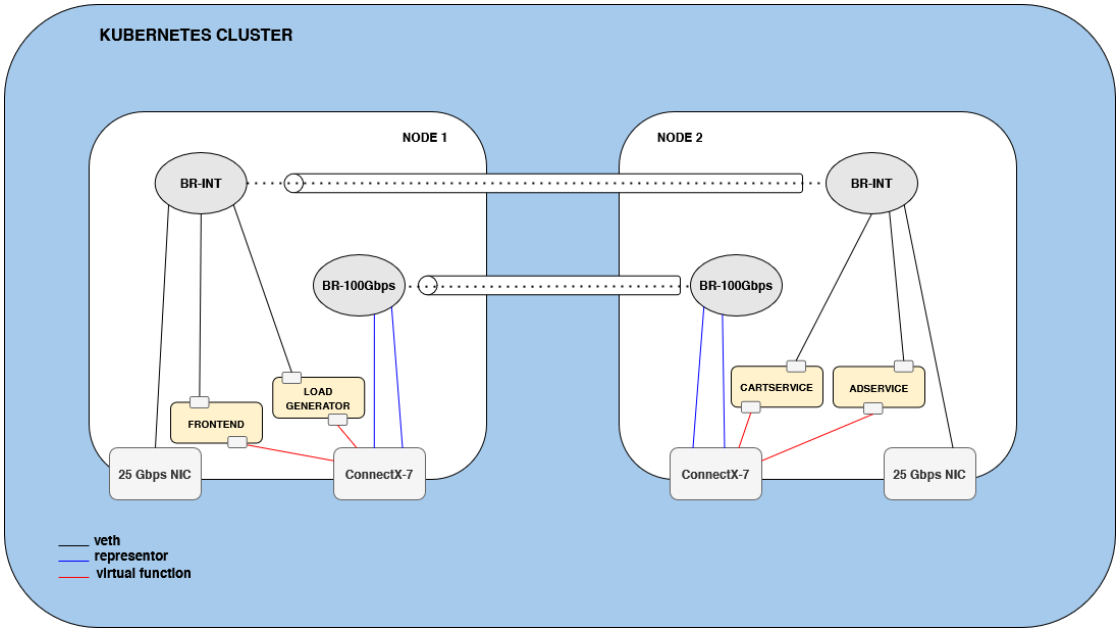

Figure 5.20: Configuration for the test with the Google Online Boutique in a Kubernetes cluster with offload.

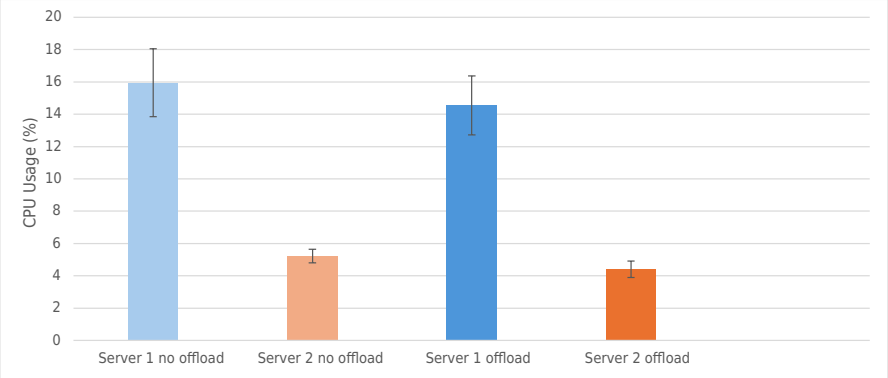

Figure 5.21: CPU usage under Google Online Boutique test. Frontend pod and loadgenerator pod ran both on Server 1.

does not maximize efficiency at high concurrency, multiple parallel Locust instances were launched: ten total processes, each simulating 10,000 users, for a cumulative load of 100,000 virtual users. In both the *offloaded* and *non-offloaded* cases, the back-end microservices were unchanged; the only difference was that, for the offload scenario, each pod's *secondary* interface was bound to an SR-IOV Virtual Function (VF) through Multus, whereas in the baseline scenario all pods shared a 100Gbps virtual Ethernet device. It is worth noting that in both scenarios, the ConnectX-7

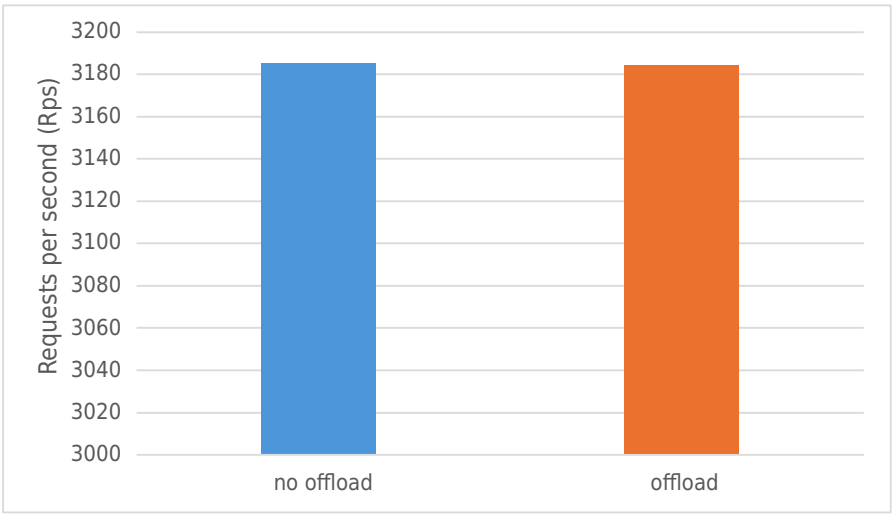

Figure 5.22: Requests per second under Google Online Boutique test.

card served as the secondary NIC on each pod, while the *primary* NIC was a separate 25Gbps device. Moreover, the creation and assignment of VFs in the offloaded setup were automated through the *SR-IOV Network Device Plugin*.

Figures 5.21 and 5.22 summarize the results, focusing on CPU usage and request rates (requests per second). Surprisingly, the difference in performance with or without SR-IOV offload remains minimal. Both approaches sustain roughly the same number of requests per second, and the CPU utilization is very similar across the pods, including those handling critical paths such as the frontend. The *loadgenerator* itself hits its own internal limits regardless of whether hardware offload is active. These observations align with what was reported in [12], where typical CPU overhead for networking stands around 8% in containerized workloads.

Ultimately, the Online Boutique tests demonstrate that, on powerful modern hardware, the networking subsystem represents a smaller share of overall CPU usage, and offloading does not always translate into dramatic performance improvements for highly local microservice communications. However, once workloads involve additional overhead, such as inter-node encryption or large-scale cross-node requests, SmartNIC offload remains a promising route to scale up secure networking without exceeding CPU limits.

Chapter 6

Results

The tests performed across different network configurations and hardware setups highlight the nuanced impact of SmartNIC offloading. While some scenarios yield notable gains in throughput or CPU savings, others demonstrate more modest or even counterintuitive results. This chapter consolidates the main findings from all experiments, emphasizing how hardware offloading behaves under varying levels of concurrency, encapsulation, and cryptographic overhead.

6.1 Comparing First and Second Setups

Two test setups were employed to explore offloading on older vs. newer hardware. The *first setup* uses servers with limited PCIe bandwidth (Gen 3 x8) and fewer CPU cores, whereas the *second setup* benefits from Gen 4 x16 and more powerful CPUs. The differences between these environments underpin much of the variability in results.

6.1.1 Offloading in the First Setup

In the older configuration, we observed a consistent pattern: offloading through SR-IOV often did not increase raw TCP throughput beyond 60–64Gbps, yet it substantially reduced CPU utilization when many parallel streams were active. In simpler tests without encapsulation, this phenomenon became especially clear. The CPU overhead for a virtual Ethernet (*veth*) client soared at higher concurrency, whereas SR-IOV effectively capped CPU usage. Although line rate (100Gbps) was never fully reached, the ability to maintain stable throughput while cutting CPU consumption proved advantageous. In data centers, avoiding full CPU saturation is often as critical as chasing maximum throughput.

When additional protocols were introduced, such as Geneve tunneling, offloading

helped mitigate the extra overhead. The cost of encapsulating traffic in software is non-trivial, especially under concurrency. SR-IOV offloading partially moved that burden into the ConnectX-7, limiting the CPU's involvement. While the difference in throughput at times remained modest, the CPU relief was typically more substantial. Therefore, on older hardware, any overhead, encapsulation or otherwise, becomes more significant, thereby enhancing the perceived benefit of hardware offloads.

In the most demanding scenario tested, combining VXLAN tunneling with IPsec encryption, the results became more dramatic. Despite raising CPU usage compared to a purely software-based approach, the net throughput soared from a baseline of 2.5Gbps to almost 27Gbps (an approximately 980% improvement). In this situation, the encryption engine on the SmartNIC essentially removes the cryptographic bottleneck from the CPU's workload. Although the processor ends up managing some control-plane tasks and partial flow steering, the heavy lifting of per-packet encryption is delegated to the ConnectX-7. This highlights that, in real-world use cases, particularly those involving computationally expensive security protocols, SmartNIC offloading can be the difference between an impractically low data rate and a throughput close to line rate.

6.1.2 Offloading in the Second Setup

Moving to the newer servers changed the picture somewhat. With faster CPUs and full PCIe Gen 4 x16 bandwidth, the host already manages a high volume of traffic with minimal CPU strain. Under standard TCP tests (without encryption or encapsulation), the offload advantage became narrower than in the first setup. In some tests, offloading even proved counterproductive for throughput at lower concurrency levels, possibly because the overhead of diverting flows into hardware overshadowed any CPU relief. At higher concurrency, CPU usage differences sometimes reversed or became negligible, suggesting that the improved baseline performance of the second setup leaves less room for offloading to shine.

In Geneve-encapsulated tests on newer hardware, a similar trend persisted. While SR-IOV occasionally reduced CPU usage, the difference between offloaded and non-offloaded configurations remained smaller compared to the older environment. Because the CPU and PCIe bandwidth can handle large traffic volumes in software, offloading yields only marginal benefits. In certain runs, throughput either stagnated around the same level for both modes or was even slightly lower for the offloaded configuration, likely reflecting the overhead of maintaining hardware flows.

Microservices Scenario: Google Online Boutique

The Google Online Boutique microservices demo on the *second setup*, provides insight into offloading under a more realistic containerized workload. In stark contrast to the encryption-centric VXLAN + IPsec case, here the CPU overhead for networking was comparatively small, leaving less room for hardware acceleration to shine. The throughput, measured in requests per second (RPS), remained nearly identical with or without SR-IOV offload, and overall CPU consumption varied by only a few percentage points. These findings suggest that in well-provisioned, microservice-oriented environments, network overhead may account for a modest slice of total usage, thus diminishing the incremental gains from offload.

Unlike high-bandwidth or heavily encrypted data streams, each service in the Boutique mainly exchanges moderate-sized requests and responses. The short and bursty traffic pattern does not push the ConnectX-7 SmartNIC to its limits, and the higher PCIe bandwidth available in the second setup further reduces the chance of hitting CPU or I/O bottlenecks in software.

Viewed in context with the other benchmarks, the Boutique scenario reinforces the observation that offloading is most impactful when the networking subsystem bears a large fraction of the system load. If the majority of cycles are already spent in application logic, or if local container-to-container traffic remains in software loops, hardware acceleration brings marginal returns.

Several factors may explain the muted impact of offloading in this scenario. Because many of the microservices involved in the Boutique communicate within the same node, traffic does not necessarily benefit from a direct hardware data path: in the offload configuration, pods on the same node must still traverse the physical NIC, which adds overhead for local traffic that might otherwise stay in software. Conversely, the non-offloaded case uses a purely virtual switch path for pods co-located on the same host, bypassing the physical device and thus avoiding the overhead of SR-IOV for local communications.

This result implies that offload can be less effective when the bulk of communication happens among containers on the same host, since the SmartNIC must handle traffic that might never leave the local machine. Potential resolutions to this problem could involve:

- **Topology-aware Scheduling**: Placing pods on different nodes when encryption or complex tunneling benefits are desired, ensuring that offloading hardware is fully utilized for cross-node traffic.
- **Refined Data Path Configurations**: Allowing pods on the same node to use a software path, while directing cross-node (or encrypted) flows to the VF. This might require advanced policy definitions in Multus or Kube-OVN.
- **Balancing CPU vs. Security/Isolation Requirements**: If the goal

is robust isolation or IPsec-level security, occasionally incurring additional overhead is a fair trade-off. In that case, high concurrency encryption might still benefit from the SmartNIC's crypto engine, but pure local container-to-container traffic could remain on a virtual switch path.

In sum, while the Google Online Boutique illustrates how high-level microservices can reduce the relative influence of raw packet processing overhead, it does not negate the role of offloading in more demanding or encryption-intensive use cases. Operators deploying multi-node clusters and requiring secure cross-node communications, or anticipating sustained high concurrency with large data transfers, could still find significant advantages in offloading. However, for lightweight, largely intra-node microservice traffic on modern hardware, SR-IOV offloading may yield minimal improvements in either CPU usage or total request-handling capacity.

6.2 Synthesis of Findings

In general, these experiments illustrate that hardware offloading is not universally beneficial in all workloads. Its impact depends heavily on the specific interplay between concurrency, protocol overhead, cryptographic requirements, and the underlying hardware's capacity to process packets in software.

CPU Gains vs. Throughput Gains. For many network-centric tasks (e.g., raw TCP streaming), offloading frequently translates into CPU savings rather than heightened throughput. The system remains capped by either PCIe bandwidth or the inherent overhead of TCP. However, relieving CPU resources can be vital for large-scale deployments, where servers run numerous services in parallel, and hitting a CPU bottleneck can limit concurrency.

Encryption as a Game-Changer. The clearest advantage of offloading arises in encryption-heavy workloads, as demonstrated by the VXLAN + IPsec scenario on the first setup. Software-based ciphers can drastically hamper performance. Delegating them to the SmartNIC allows throughput to climb by an order of magnitude, albeit sometimes with a minor increase in CPU usage. For organizations prioritizing secure communications at scale, hardware offloading can thus be essential.

Impact of Better Hardware. On more modern servers (Gen 4 x16), the improved baseline performance can reduce the relative gains from offloading. Indeed, offload overhead might slightly hinder throughput at low concurrency. At the same

time, the potential for CPU reduction remains valuable at higher loads, particularly when advanced protocols like Geneve or IPsec are involved. However, those benefits no longer appear as pronounced as they do on systems with more constrained resources.

6.3 Practical Considerations

The diverse outcomes across test setups underscore the necessity of tailoring offloading strategies to specific data center environments. When CPU capacity or PCIe bandwidth is limited, SR-IOV can deliver substantial improvements—especially for tasks involving encapsulation and encryption. In contrast, in an already well-provisioned environment, the incremental advantages may be smaller, and some overheads of hardware offloading might even hinder throughput in certain corner cases.

Still, the experiment with IPsec encryption shows that, whenever a workload involves computationally intensive tasks beyond simple packet forwarding, moving those functions to dedicated SmartNIC hardware provides tangible benefits. Enterprises seeking to secure east-west traffic at scale or handle surging concurrency without overwhelming their CPU fleet, would likely profit from hardware-based encryption. By contrast, organizations deploying modern servers with high core counts and broad PCIe lanes might see less difference in basic tunneling or bridging scenarios.

These results collectively suggest that hardware offloading should be viewed as a flexible tool, which is beneficial for a number of areas where CPU overheads hamper performance (e.g. encryption or limited PCIe bandwidth), yet not always necessary in high-end, lightly loaded servers. Deciding whether to enable offloading depends on balancing the cost and complexity of configuring SR-IOV with the potential for efficiency gains under real-world workloads.

6.4 Hardware Offloading with ConnectX-7 limits

Although ConnectX-7 provides robust support for offloading certain tasks, its capabilities are inherently restricted to the protocols and operations implemented in hardware. The card's hardware engines excel at accelerating IPsec encryption, TCP offload, and certain tunneling mechanisms, but do not accommodate every emerging protocol. WireGuard, for instance, is increasingly adopted for secure networking but remains incompatible with ConnectX-7's offload pipeline. This lack of wire-speed offloading for newer or less standardized protocols reflects an inherent inflexibility: once the ASIC design is finalized, introducing or updating features becomes considerably more challenging.

Even when ConnectX-7 handles supported protocols efficiently, its offload approach offers less elasticity compared to SoC-based SmartNICs, like the BlueField family. Unlike BlueField, which runs an ARM-based operating system and supports custom code execution on the NIC itself, ConnectX-7 mainly accelerates pre-programmed functions defined by its ASIC. As a result, ConnectX-7 users benefit from reduced CPU overhead when leveraging its built-in offload functions, such as IPsec, but cannot easily extend hardware offloading to proprietary or fast-evolving network protocols. For large-scale or highly dynamic data centers that frequently update their security and networking features, this limitation underscores the importance of evaluating not just raw throughput or CPU savings, but also the long-term adaptability of a particular SmartNIC solution.

Chapter 7

Conclusions

7.1 Current Limitations

While the results demonstrate clear advantages in utilizing SmartNIC offloading in scenarios like packet encryption, there are several limitations to this study that must be acknowledged. One key limitation is the reliance on synthetic workloads for performance evaluation. Most of the tests were conducted using tools such as `iperf3` and controlled traffic generators, which, while effective for measuring raw throughput and CPU utilization, do not fully capture the complexity of real-world data center workloads. The only test involving a real-world application was performed with Google Online Boutique, which provides some insight into how offloading impacts containerized microservices, but remains a limited representation of broader production environments.

Another limitation of this study is the focus on a single SmartNIC model, the NVIDIA ConnectX-7. While this hardware provides extensive offloading capabilities, the results cannot be generalized across all SmartNIC architectures, such as FPGA-based or SoC-based solutions. Different SmartNIC designs may offer varying levels of performance improvements depending on workload characteristics and network stack integration.

Additionally, the experiments were conducted on a limited number of hardware configurations, specifically two test setups with distinct CPU and PCIe architectures. While this approach allowed for a comparison between constrained and optimal PCIe bandwidth environments, a broader range of hardware setups would be necessary to fully understand how offloading scales across different data center architectures.

Finally, the evaluation primarily focused on networking-related metrics such as CPU utilization and throughput. Other potential benefits of SmartNIC offloading,

such as energy efficiency and impact on overall system performance in multi-tenant cloud environments, were not explored in this study. These aspects remain important in determining the viability of SmartNIC deployment in large-scale production environments.

7.2 Future Work

There are several directions for future research and development aimed at addressing the limitations identified in this study. One crucial area is the optimization of SmartNIC performance in environments with constrained PCIe bandwidth. Investigating compression techniques, workload-aware flow scheduling, or alternative interconnect solutions could help alleviate the impact of PCIe bottlenecks on performance.

Another promising direction is improving software integration for SmartNICs within widely used networking stacks such as Open vSwitch and Kubernetes. Enhancements to automation frameworks could simplify the configuration process, reducing the technical barrier to adopting SmartNIC offloading in real-world data center environments. Developing intelligent orchestration mechanisms that dynamically adjust offloading strategies based on workload profiles could further optimize efficiency.

A more detailed analysis of offloading performance in diverse networking scenarios would also be valuable. Future studies could explore SmartNIC impact on microservices communication patterns, network security functions, and distributed machine learning workloads. Extending the evaluation to include multiple SmartNIC vendors and architectures, such as FPGA-based offloading solutions, could provide deeper insights into trade-offs between different hardware implementations.

Additionally, exploring the role of SmartNICs in accelerating emerging networking paradigms, such as software-defined networking (SDN) and network function virtualization (NFV), could open new avenues for improving scalability and efficiency in cloud and edge computing environments. As SmartNIC technology continues to evolve, further optimizations and refinements will be necessary to maximize its impact across a broader range of applications.

Another fruitful line of research lies in **policy-based** traffic steering for container environments. In particular, solutions like Multus could evolve to transparently decide whether a given flow should be offloaded or routed via a software path, depending on whether the traffic stays within the same host or must traverse the physical network. If pods on the same node can communicate purely via virtual Ethernet, they might avoid needless overhead of traveling to the SmartNIC. Conversely, external or cross-node flows would naturally benefit from hardware offloading, especially if they require encryption or tunneling. By integrating

advanced workload awareness and network policies, future Multus implementations could dynamically attach pods to either a fully offloaded SR-IOV interface or a software-based veth interface, thereby striking an optimal balance between performance, CPU usage, and scalability.

Bibliography

[1] OvS Docs. URL: https://www.openvswitch.org/ (cit. on p. 4).

[2] Kubernetes Docs. URL: https://kubernetes.io/docs/concepts/overview/components/ (cit. on p. 7).

[3] Kube-OVN Docs. URL: https://kubeovn.github.io/docs/stable/en/ (cit. on p. 9).

[4] Multus Docs. URL: https://github.com/k8snetworkplumbingwg/multus-cni (cit. on p. 9).

[5] SR-IOV network device plugin Docs. URL: https://github.com/k8snetworkplumbingwg/sriov-network-device-plugin (cit. on p. 10).

[6] Google Online Boutique Docs. URL: https://github.com/GoogleCloudPlatform/microservices-demo (cit. on p. 10).

[7] Elie F. Kfoury, Samia Choueiri, Ali Mazloum, Ali AlSabeh, Jose Gomez, and Jorge Crichigno. «A Comprehensive Survey on SmartNICs: Architectures, Development Models, Applications, and Research Directions». In: *IEEE Access* 12 (2024), pp. 107297–107336. DOI: 10.1109/ACCESS.2024.3437203 (cit. on p. 12).

[8] Rashid Khan and Rony Efraim. *Implementing virtual network offloading using open source tools on BlueField-2*. 2021. URL: https://www.nvidia.com/en-us/on-demand/session/gtcspring21-s31380/ (cit. on p. 13).

[9] Jianshen Liu, Carlos Maltzahn, Craig Ulmer, and Matthew Leon Curry. «Performance Characteristics of the BlueField-2 SmartNIC». In: (2021). URL: https://arxiv.org/abs/2105.06619 (cit. on p. 13).

[10] Rony Efraim and Or Gerlitz. «Using SR-IOV offloads with Open-vSwitch and similar applications». In: (2023). URL: https://netdevconf.org/1.2/papers/efraim-gerlitz-sriov-ovs-final.pdf (cit. on pp. 13, 20).

[11] NVIDIA Docs. URL: https://docs.nvidia.com/networking/display/mlnxofedv24010331/ipsec+full+offload (cit. on p. 21).

[12] Davide Miola, Fulvio Risso, and Federico Parola. «Measuring the Cost of the Linux Network Stack in Real-Time». In: *2024 IEEE 10th International Conference on Network Softwarization (NetSoft)*. 2024, pp. 295–303. DOI: 10.1109/NetSoft60951.2024.10588891 (cit. on p. 37).