INTRODUCTION

RDMA-Accelerated Container Infrastructure

A Scalable Architecture for Multi-Tenant Storage and Network Workloads, on a 100 GbE Private Cloud

Coming up

- Compute
- Storage
- Network
- Results

OVERVIEW

The Problem With Faster Networks

Modern private clouds need isolation, availability, and speed — at the same time

- Modern private-cloud infrastructure is expected to provide multi-tenant isolation, highly available services, replicated storage, observability, and secure remote access — all while network speeds move well past 100 GbE.
- This presentation studies NGLAB, an RDMA-accelerated, multi-tenant private cloud built on a 100 GbE Mellanox fabric, Incus containers and VMs, Open vSwitch/OVN software-defined networking, and LINSTOR/DRBD replicated storage.

The core claim

- Installing faster adapters and switches alone does not guarantee proportional gains in application or storage performance.
- The real bottleneck is usually the host networking stack, the software switching layer, the storage replication path, CPU interrupts, and overlay networks — especially once tenant traffic, management traffic, storage replication, and service access all share one physical fabric.

RESEARCH QUESTION & METHODOLOGY

What We're Asking, and How We Prove It

Primary question

How can RDMA, hardware offloading, software-defined networking, and software-defined storage be combined to design a redundant, lossless, and highly available multi-tenant private cloud?

Supporting questions

- How can a 100 GbE fabric support both cloud-management traffic and high-performance RDMA data-plane traffic?
- What physical network configuration does RoCEv2 actually require?
- How does DRBD provide replicated block storage for an Incus private cloud at high speed?
- How do Open vSwitch and OVN provide tenant isolation and overlay connectivity?
- Where do CPU overhead, TCP, software switching, and replication introduce bottlenecks — and how much can offloading and RDMA remove?
- What operational trade-offs follow from RDMA, and when does conventional TCP remain the better choice?

The proof: three configurations, one cluster

- Baseline — the entire data path runs through the kernel: every packet classified, switched, and copied by the host CPU; storage replication over plain TCP.
- Accelerated , no RDMA — packet processing offloaded to the NIC's embedded switch.
- Accelerated with RDMA — packet processing offloaded to the NIC's embedded switch; storage replication over RDMA, writing directly into remote memory. Comparing the two isolates exactly what offloading and RDMA are worth.

ARCHITECTURE

Two Sub-Clusters, Two Node Classes

A management/services layer, and a high-performance compute + storage layer

- Docker Swarm — the service group.
  - *Reverse proxies, load balancers, databases, backups, and the Grafana/Loki observability stack — kept highly available and separate from the primary workload environment.*
- Incus — the compute group.
  - *The main multi-tenant environment: containers and VMs for educational and research workloads, connected to the 100 GbE fabric via Mellanox ConnectX-5/6 Dx adapters.*
- Within Incus, six physical nodes split into two classes:
  - *Compute class — Gigabyte, Supermicro, R620 — diskless; all workload storage I/O crosses the 100 GbE fabric.*
  - *Storage/network-controller class — R730xd-1/2/3 — diskful; back the LINSTOR pools, run the LINSTOR controller and satellites, and host OVN central services.*

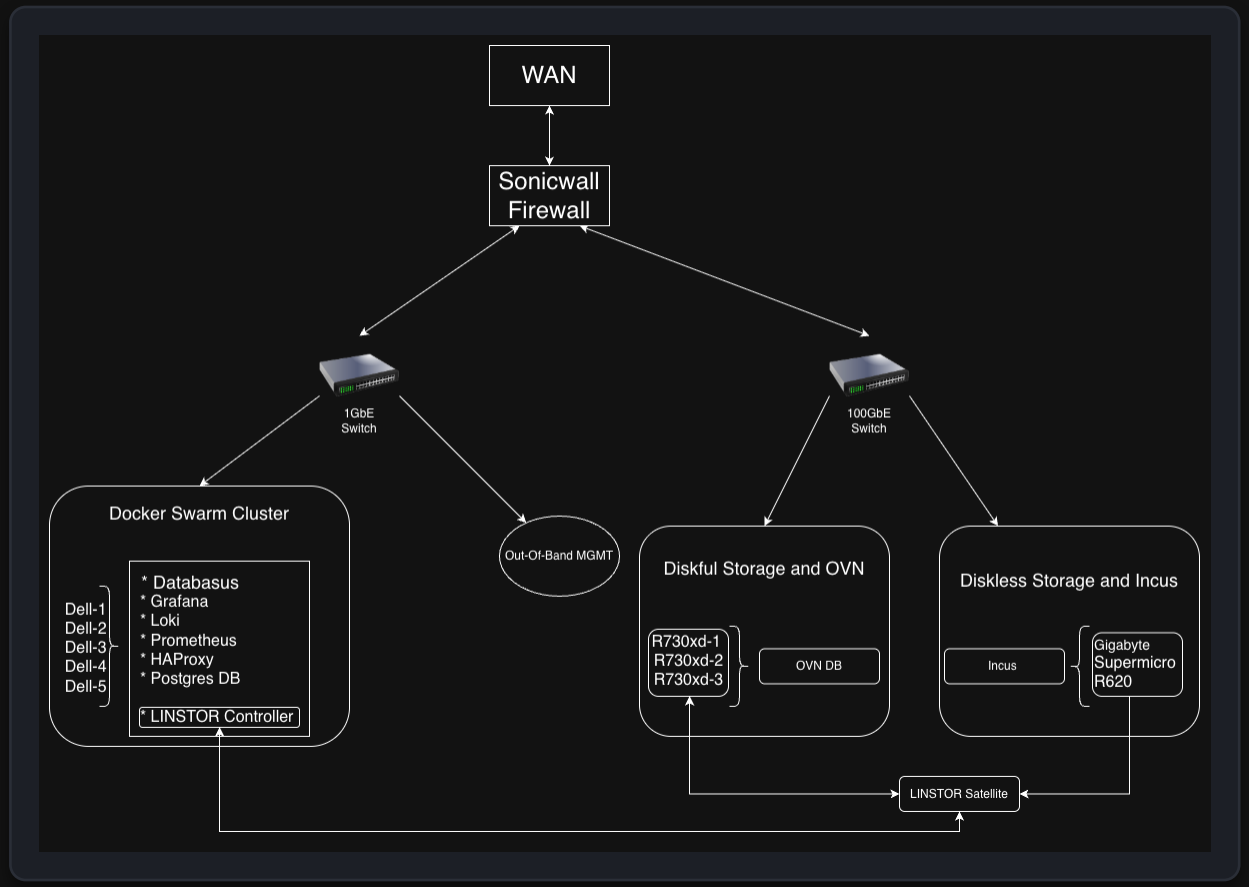


Figure 2.1 — Network Map

ARCHITECTURE

One VLAN per Traffic Class

A uniform scheme keeps every traffic class on its own Layer 2 domain

- Each VLAN carries exactly one class of traffic on its own dedicated /28 subnet — 14 usable addresses, enough for six nodes plus gateways, uplinks, and virtual IPs.
- VLANs 10–30 run on both node classes (every node replicates, is LINSTOR-managed, and joins the OVN plane). VLAN 40+ runs only on compute nodes — per-tenant uplinks are allocated on demand as new tenants are provisioned.

VLAN	Subnet	Purpose	Node Class
10	4.1.0.0/28	DRBD data layer replication traffic	Compute, Storage
20	4.2.0.0/28	LINSTOR management layer	Compute, Storage
30	4.3.0.0/28	OVN management layer	Compute, Storage
40	4.4.0.0/28	Incus cluster management layer	Compute
50	4.5.0.0/28	Infrastructure OVN uplink, WAN-routed	Compute
60	4.6.0.0/28	Admin-labs OVN tenant uplink, WAN-routed	Compute
N	4.N.0.0/28	Faculty tenant OVN uplink, WAN-routed	Compute

Table 2.1 — VLAN scheme for the 100 GbE cluster fabric

ARCHITECTURE

Perimeter Policy & Network Zones

A SonicWall NSA 4650 turns each VLAN into its own enforced zone

- Every VLAN on the 100 GbE fabric maps to its own SonicWall zone — the firewall enforces what may cross into another zone, and what may leave for the WAN.

– *Management zones (VLANs 10, 20, 30, 40 — DRBD, LINSTOR, OVN, Incus control): fully restrictive. No traffic leaves the zone toward any other zone or the WAN — the four control-plane classes stay mutually isolated even while sharing one physical fabric.*

– *Tenant uplink zones (VLAN 50+): permitted to egress toward the WAN, so a project's containers and VMs get real internet access — but inter-tenant traffic between these zones stays prohibited.*

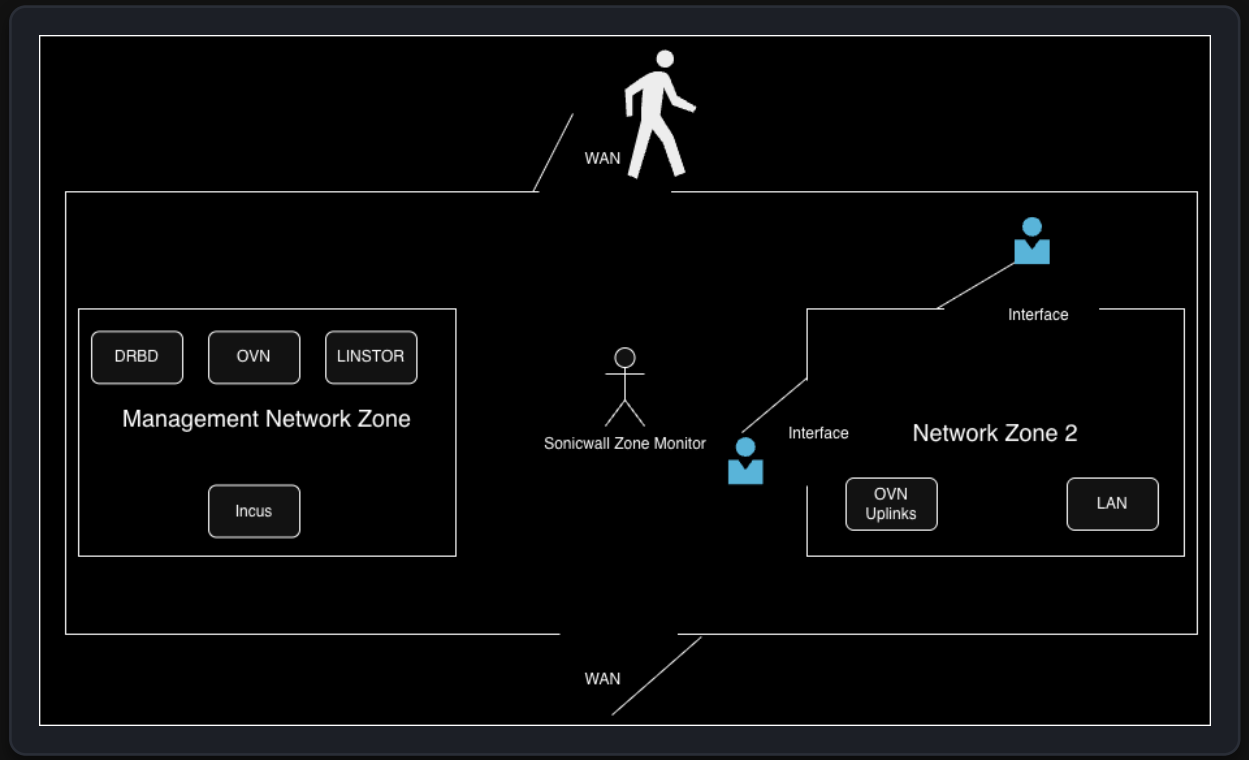


Figure 2.2 — SonicWall Network Zones

THE SERVICE GROUP

Purpose & Applications

A modest 5-node group that carries every control-plane dependency the cluster leans on

- Distinct from the high-performance Incus group — 5 nodes on a gigabit switch, dominated by control-plane traffic, configuration access, and observability data rather than heavy bandwidth flows.
- The Docker Swarm overlay network hosts every platform service the rest of the infrastructure depends on, reachable through a single load-balanced Virtual IP.

PostgreSQL

Persistent state for Guacamole's connection store and other services' metadata.

Nginx Proxy Manager

Terminates inbound TLS at the VIP and routes to the correct backend service.

LINSTOR Controller

The authoritative controller for both the service group and the Incus group.

Prometheus · Loki · Grafana

The observability stack — metrics, logs, and a unified dashboard.

Databasus

Apache-2.0 backup automation — encrypted database backups shipped to Google Drive.

Bind9 + Auth0 (OIDC)

Bind9 resolves per-tenant DNS zones; Auth0 provides single sign-on across Incus, Grafana, and the LINSTOR web UI.

THE SERVICE GROUP · TOPOLOGY

All Five Nodes, All Managers

A manager-only Swarm trades a little overhead for an entire class of failure modes

- Every one of the five nodes is configured as a manager rather than a worker — deliberately, for two reasons.
  - *The service count is small enough that the extra cost of five managers is minor next to the operational benefit.*
  - *It eliminates the failure mode where losing one dedicated manager leaves the group unable to accept new orchestration commands.*
- The cluster tolerates the loss of up to two of the five managers while still accepting incoming connections.
- Swarm management traffic rides its own dedicated VLAN — isolated from both LINSTOR management and Swarm data operations.
- A shared Virtual IP is maintained by keepalived (VRRP): if a node fails, the election rebinds the VIP to a survivor within seconds, keeping every client's address stable.

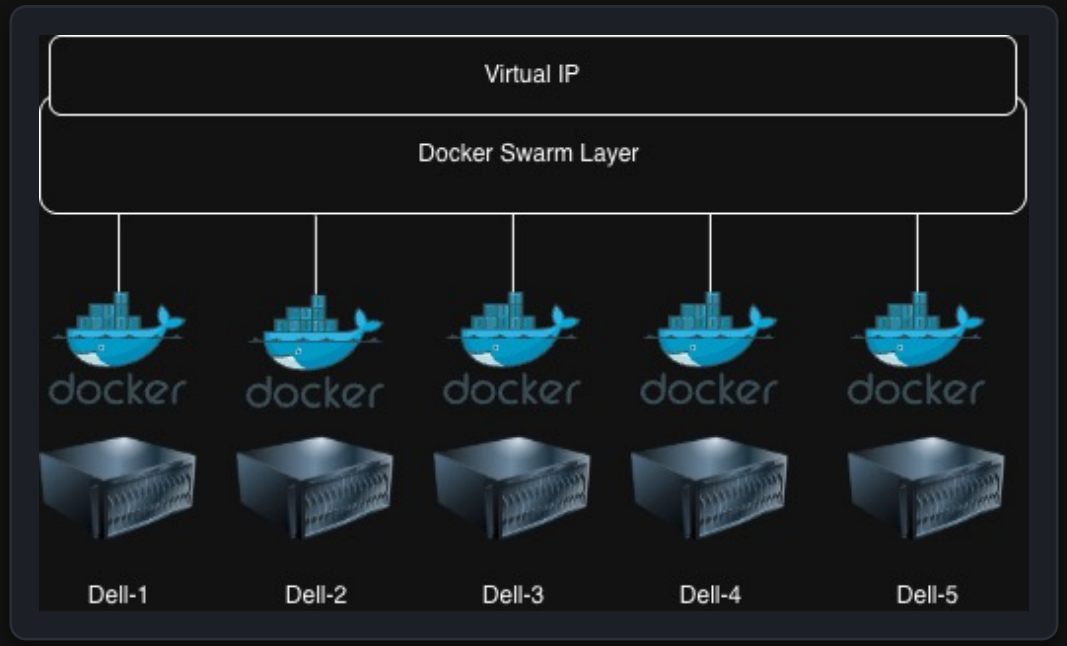


Figure 3.1 — Docker Swarm

Storage backend: LINSTOR + DRBD via a Docker volume plugin. Each node runs a LINSTOR satellite — storage pools here are independent of the Incus group's.

FROM HERE

Walking Up the Stack

With the service group and the physical architecture in place, the rest of this presentation follows the Incus group up from the metal

Compute

Incus, KVM/QEMU, GPU sharing

Storage

DRBD, quorum, LINSTOR

Network

OVN, ASAP², RDMA/RoCEv2

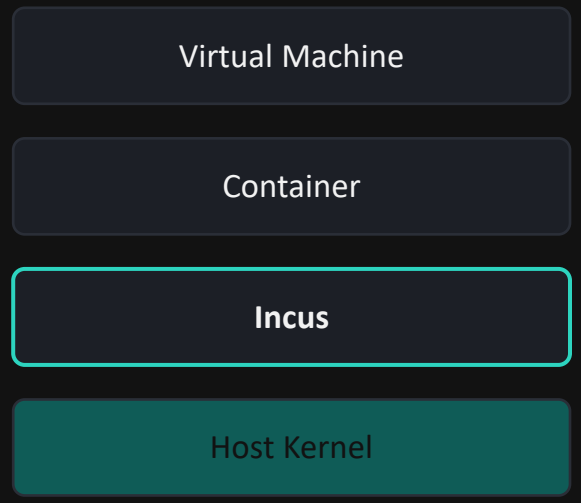
Results

Baseline vs. accelerated

PART 1 OF 3 · COMPUTE · STORAGE · NETWORK

Compute

Incus virtualization, KVM/QEMU, and GPU sharing inside the cluster infrastructure



FOUNDATIONS

The Incus Group

A unified control plane built on two kernel primitives

- Incus is a workload orchestrator and unified control plane for system containers (LXC), OCI application containers, and QEMU virtual machines.
- Incus itself is not a hypervisor — it delegates VM execution to QEMU/KVM and container containment to LXC, exposing one uniform API and CLI on top.

1

Namespaces

Isolate what a process can see: its own PID tree, network stack, mount points, and user IDs — the basis of container identity.

2

cgroups

Isolate what a process can use: enforce limits on CPU shares, memory, and block I/O per group of processes.

Together, namespaces + cgroups let an Incus container behave like a lightweight VM: a complete, resource-bounded Linux environment sharing the host kernel.

FOUNDATIONS

KVM & QEMU

The virtualization backend that Incus manages for VM workloads

What is QEMU?

- Quick EMUlator — a user-space program that provides device emulation, hardware passthrough, and machine virtualization.
- Incus depends on QEMU whenever it runs a virtual-machine workload.
- Presents an emulated or paravirtualized device model to the guest OS.

What is KVM?

- Kernel-based Virtual Machine — a kernel module exposing the CPU's hardware virtualization extensions (Intel VT-x / AMD-V) to user space.
- Once loaded, it turns the Linux kernel into a hypervisor that manages transitions between host and guest execution.
- Represents each VM as a normal user-space process, scheduled and memory-managed by the kernel like any other.

QEMU handles device emulation. KVM handles hardware-accelerated execution. Together they form the virtualization engine behind every Incus VM.

What Is a Virtual Machine?

- A software-defined computer: an isolated environment whose guest OS believes it is running on dedicated physical hardware.
- Each instance is provisioned with its own vCPU cores, vRAM, virtual disk, network interfaces, and boot firmware.
- The hypervisor and user-space VMM intercept privileged instructions and mediate access to physical resources.
- The guest kernel operates as though it exclusively owns the machine — resources are abstracted by the host across all VMs.

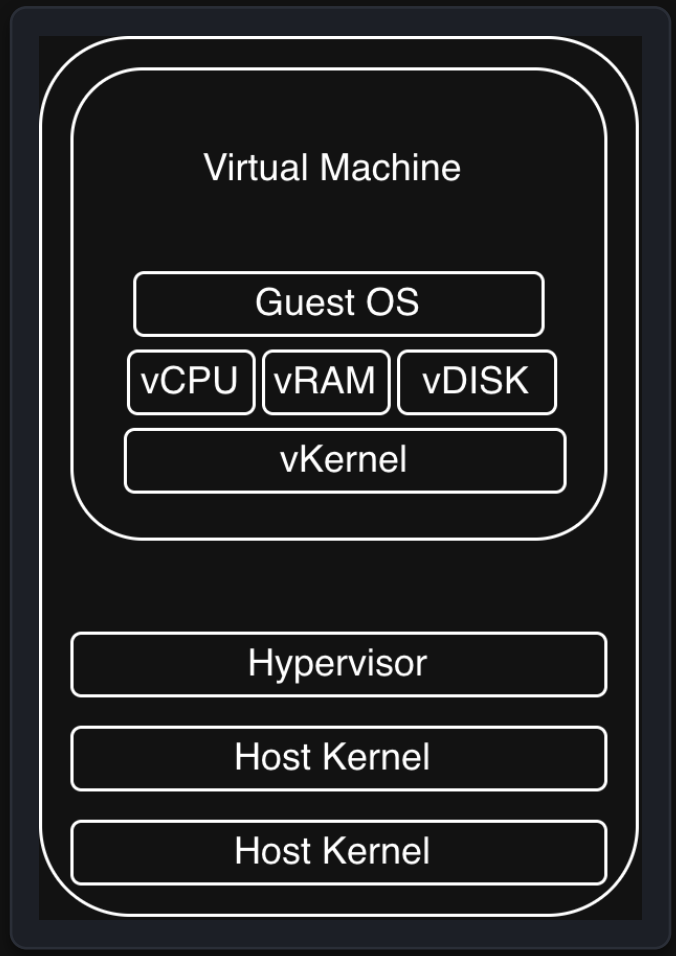


Figure 4.1 — Virtual Machine

FOUNDATIONS

What Is an Incus Container?

- Incus implements OS-level virtualization — a system container, distinct from single-service app containers like Docker or Podman.
- Each container hosts a complete Linux user space: its own init system, package manager, user accounts, and long-running services.
- All system containers on a host share a single kernel, giving significantly lower overhead and startup latency than a full VM.
- Delivers VM-like isolation and administration at a fraction of the resource cost.

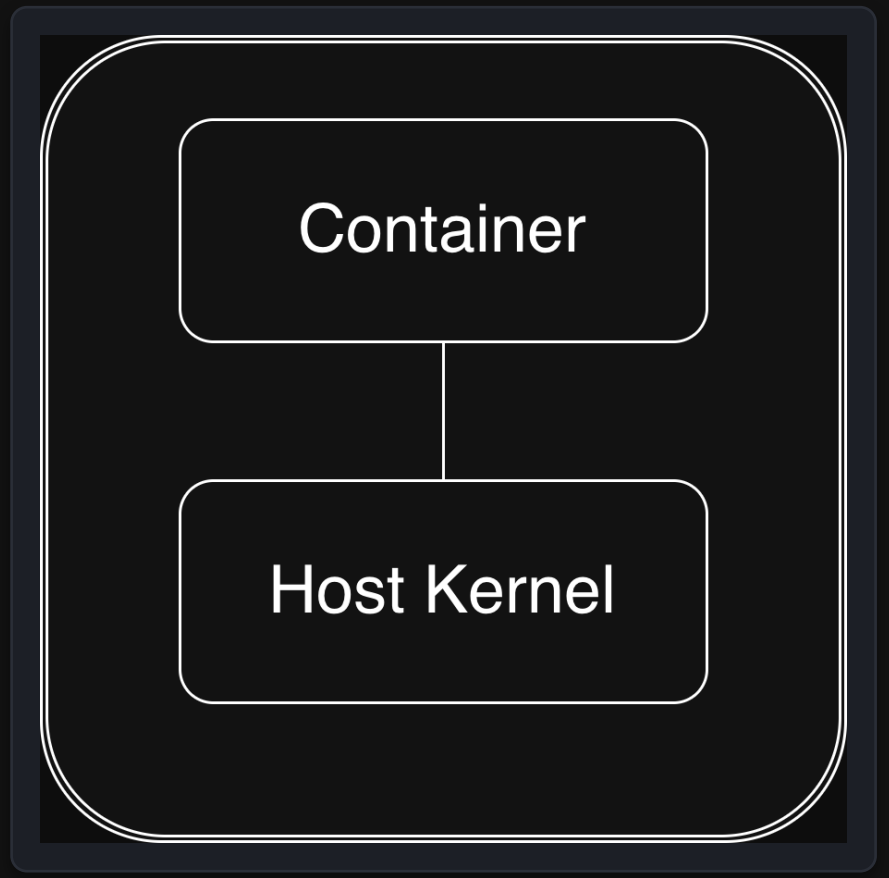


Figure 4.2 — Container

SECTION

Topology of Incus

Physical layout, control-plane structure, and project-level multi-tenancy

TOPOLOGY

Physical Topology

- Incus runs as a cluster spanning every participating node — a highly available, horizontally scalable private cloud.
- Control-plane state (dqlite / Raft consensus) is fully separated from workload execution on each node.
  - *3-node control plane: q(3) = ⌊3/2⌋ + 1 = 2 nodes required for quorum — tolerates the loss of 1 node.*
- Compute nodes are fully diskless, networked through an offloaded OVN fabric accelerated with NIC sub-functions.
- Storage (DRBD) enforces its own independent quorum — numerically aligned here, but mechanistically distinct from the compute plane.

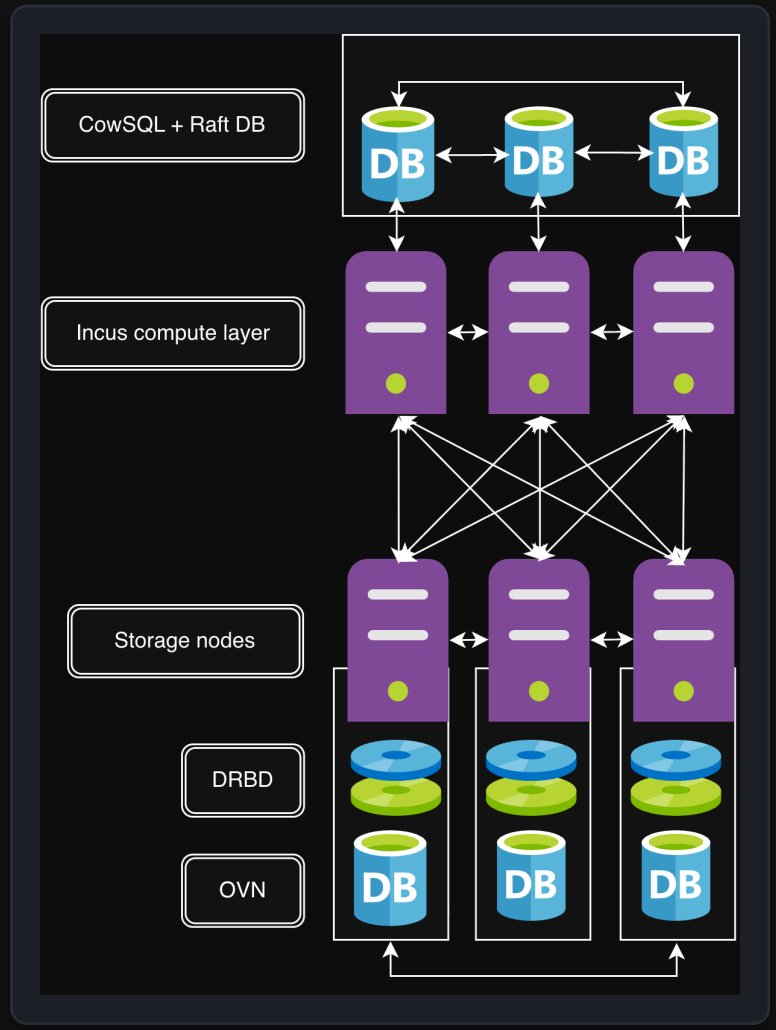


Figure 4.3 — Incus Virtualization Stack

TOPOLOGY

Project Structure & Multi-Tenancy

- Incus divides the cluster into projects — each project acts as an isolated tenant on shared hardware.
- In this deployment, each professor is a tenant, receiving their own software-defined network.
- Tenants share common profile templates and storage backends, but every tenant network is isolated from the others.
- Each network fully resolves hostnames for its own containers and VMs only — no cross-tenant visibility.

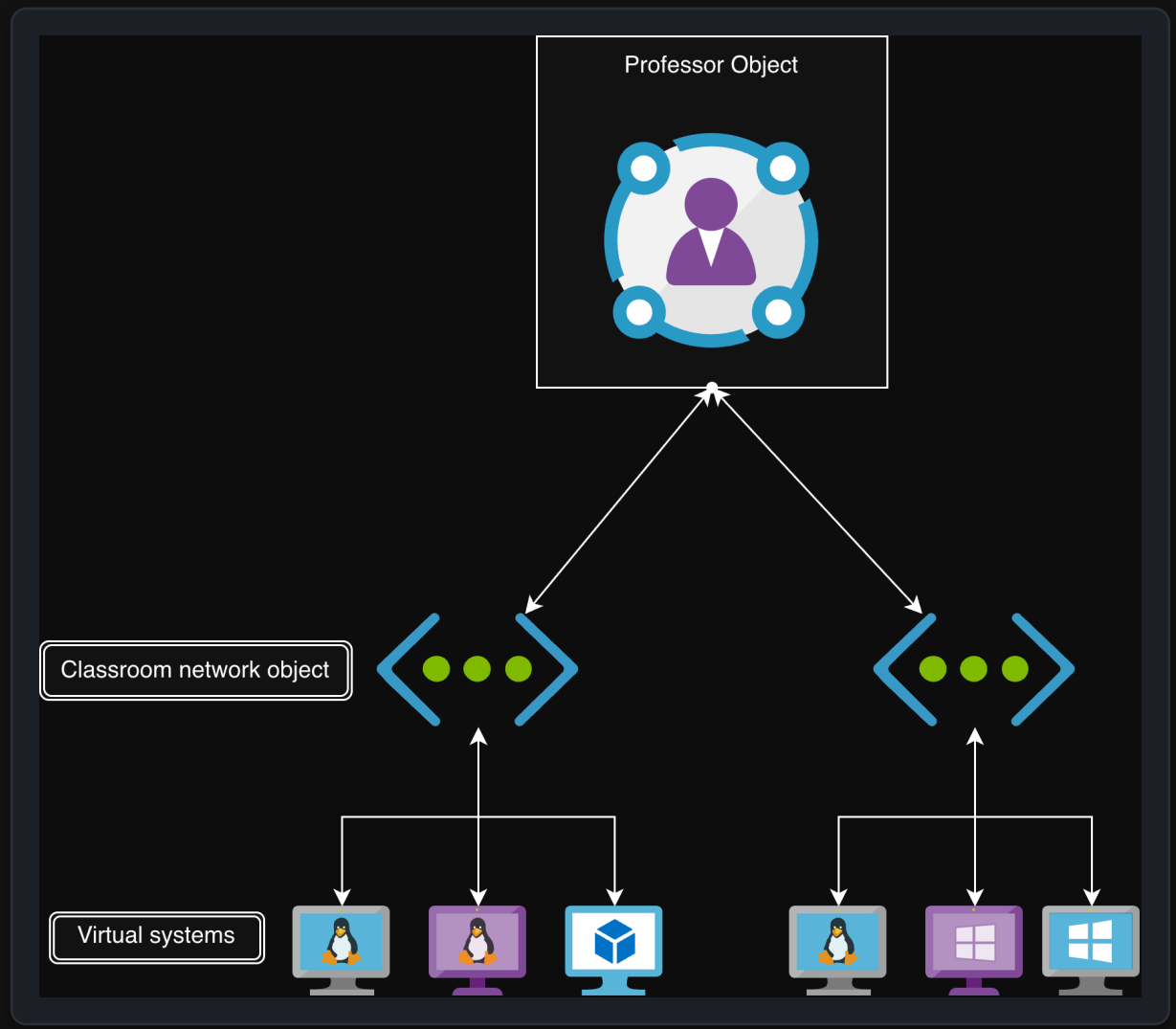


Figure 4.4 — Incus Tenant Stack

GPU Sharing Semantics for Containers

- The GPU stays attached to the host kernel — no full PCIe passthrough, so the host keeps ownership of the NVIDIA kernel module.
- Device nodes (/dev/nvidia0, /dev/nvidia-uvm) are exposed directly into each container's filesystem namespace.
- CUDA toolkit and runtime libraries are bind-mounted from one host installation instead of duplicated (~10 GiB) per container.
- Container processes queue for the GPU like ordinary host processes — the NVIDIA driver multiplexes execution across them.
- No MIG partitioning: trades hardware-level isolation for full-device access and operational simplicity.

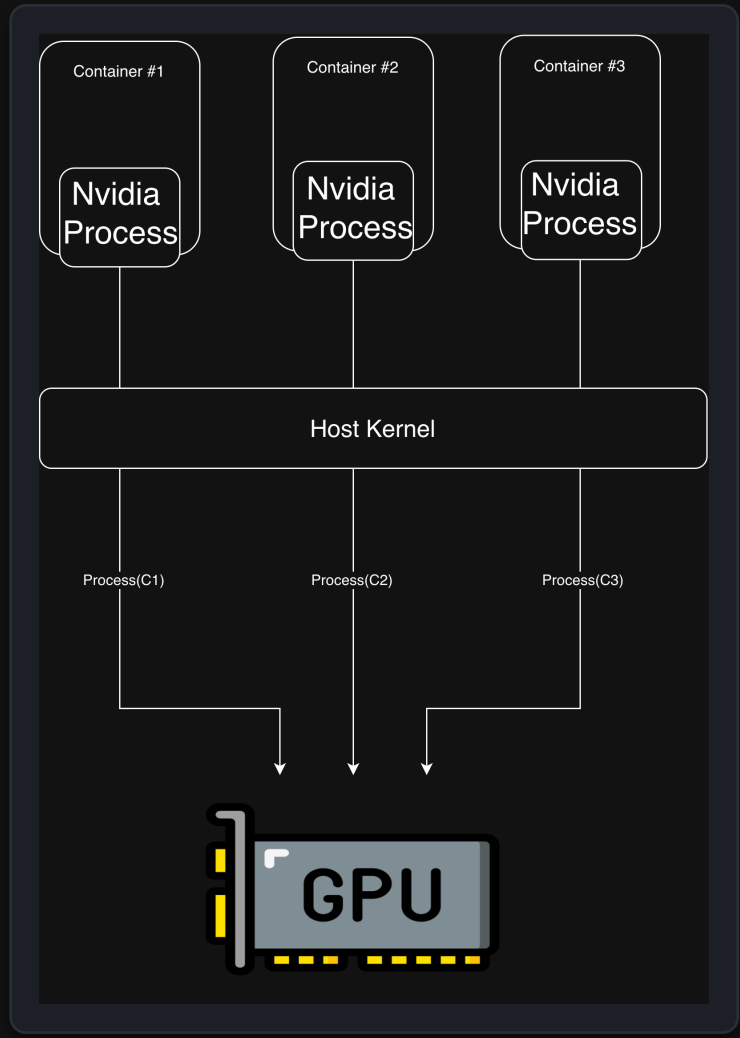


Figure 4.5 — GPU Passthrough

END-USER ACCESS

Apache Guacamole

A clientless remote-desktop gateway for the cluster

- Clientless, HTML5-based remote-desktop gateway maintained by the Apache Software Foundation.
- Lets a user open an RDP, VNC, or SSH session against any container or VM straight from a browser.
- Removes the need for local RDP/VNC/SSH clients or VPN tooling on the end user's machine — a deliberate design choice for this infrastructure.

guacamole (Web UI)

Browser-facing HTML5 component — renders the UI and manages users and connection configuration.

guacd (Guacamole daemon)

Backend daemon that speaks the actual protocols — translating each session into a live RDP, VNC, or SSH connection.

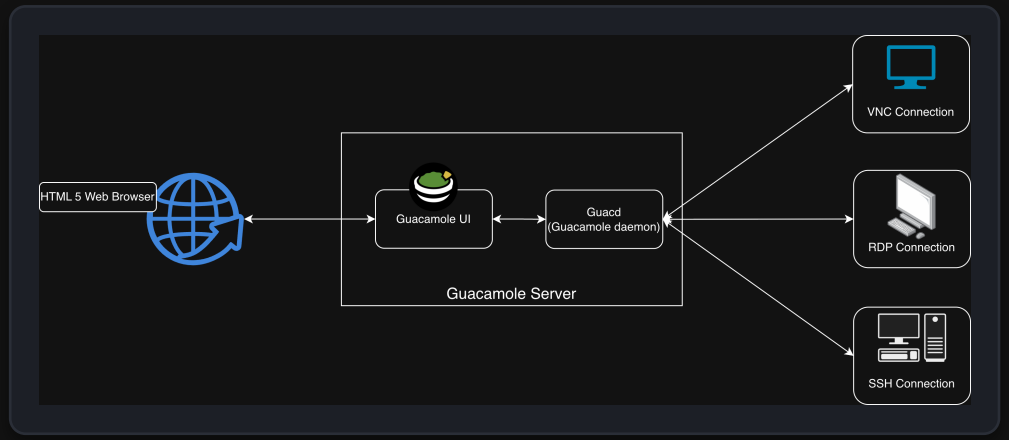


Figure 5.1 — Guacamole request flow

Containerized Deployment

OCI images, Docker Swarm, and a layered HAProxy frontend

- Both the Guacamole web application and the guacd daemon run as official OCI container images — independent Docker Swarm services that scale out separately as session load grows.
- A request layers through the stack: WAN → firewall → Nginx (TLS termination) → HAProxy load balancer → a Guacamole web replica → guacd → the target container or VM.
- HAProxy fronts both the web tier and the guacd tier, each sitting behind its own Keepalived VRRP virtual IP for high availability.

– *Packaging both components as containers was driven mainly by horizontal scalability — replicas can be added under load without interrupting existing sessions.*

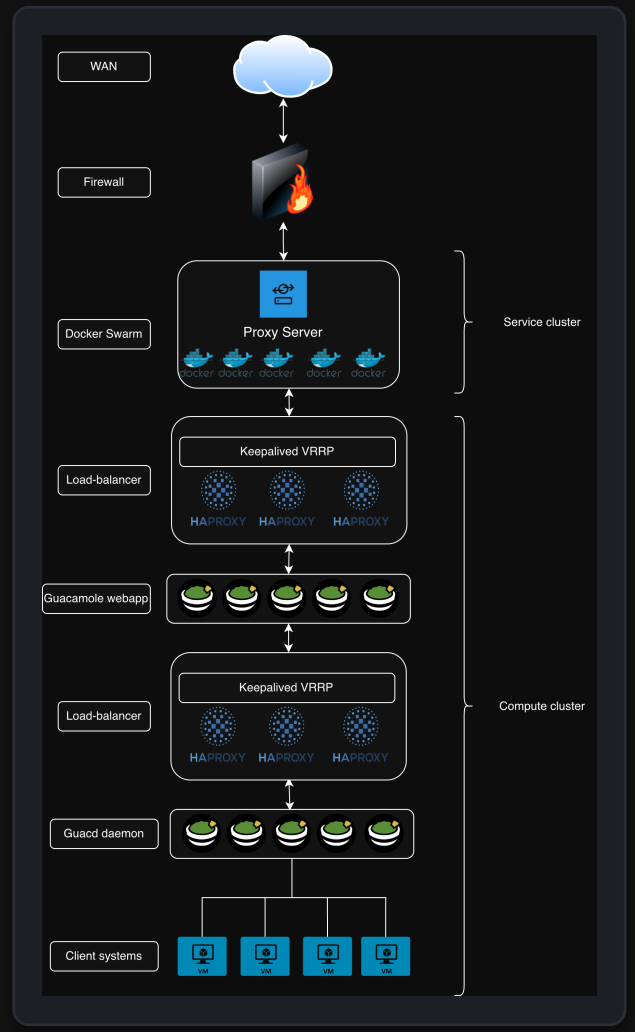


Figure 5.2 — Guacamole request flow

APACHE GUACAMOLE

Why Session Affinity Matters

A misrouted request means an instantly dropped remote-desktop session

- The Guacamole web app keeps an in-memory link between a user's HTTP session and their guacd tunnel — state that cannot be reconstructed from the database.
- A follow-up request that lands on a different replica finds no matching tunnel state, and the session disconnects immediately.

`balance source`

Hashes the client's source IP to a backend — the primary, deterministic mapping.

`1-hour stick-table`

Preserves that mapping across brief disturbances, like a backend failing and later rejoining its health check.

`cookie SERVERID`

An explicit backend cookie — the authoritative signal when many clients share one NAT'd source IP.

The guacd tier (TCP :4822) uses source-hashing alone, with no cookie — the tunnel speaks Guacamole's native protocol, not HTTP, so hashing is sufficient on its own.

APACHE GUACAMOLE

The HAProxy Configuration

Listing 5.1 — web tier · Listing 5.2 — guacd tier

```
backend guacamole_web_backends
    mode http
    balance source
    cookie SERVERID insert indirect nocache
    stick-table type ip size 200k expire 1h
    stick on src

    server guac-web1 10.143.227.2:8080 check cookie web1
    server guac-web2 10.228.0.3:8080   check cookie web2
```

```
backend guacd_backends
    mode tcp
    balance source
    option tcp-check
    stick-table type ip size 200k expire 1h
    stick on src

    server guacd1 10.112.127.2:4822 check
    server guacd2 10.144.160.3:4822 check
```

Operational payoff (§5.1.1.1): all durable state — users, connections, permissions, session history — lives in a centralized PostgreSQL database. Replicas can be rebuilt, replaced, or scaled with minimal downtime; losing one only affects the sessions actively bound to it.

Cross-Tenant Connectivity via Network Peering

Reaching into an isolated tenant network without breaking isolation

- The guacd daemons live in their own project on a dedicated management network — but the student workloads they must reach sit in separate, isolated per-tenant networks.
- Direct connectivity between these networks is prohibited by default. That's the intended multi-tenancy behavior — but it conflicts with guacd needing to open SSH, RDP, or VNC sessions against student instances on demand.

– *The fix: Incus network peers — an OVN abstraction that establishes a controlled routing relationship between two otherwise-isolated networks.*

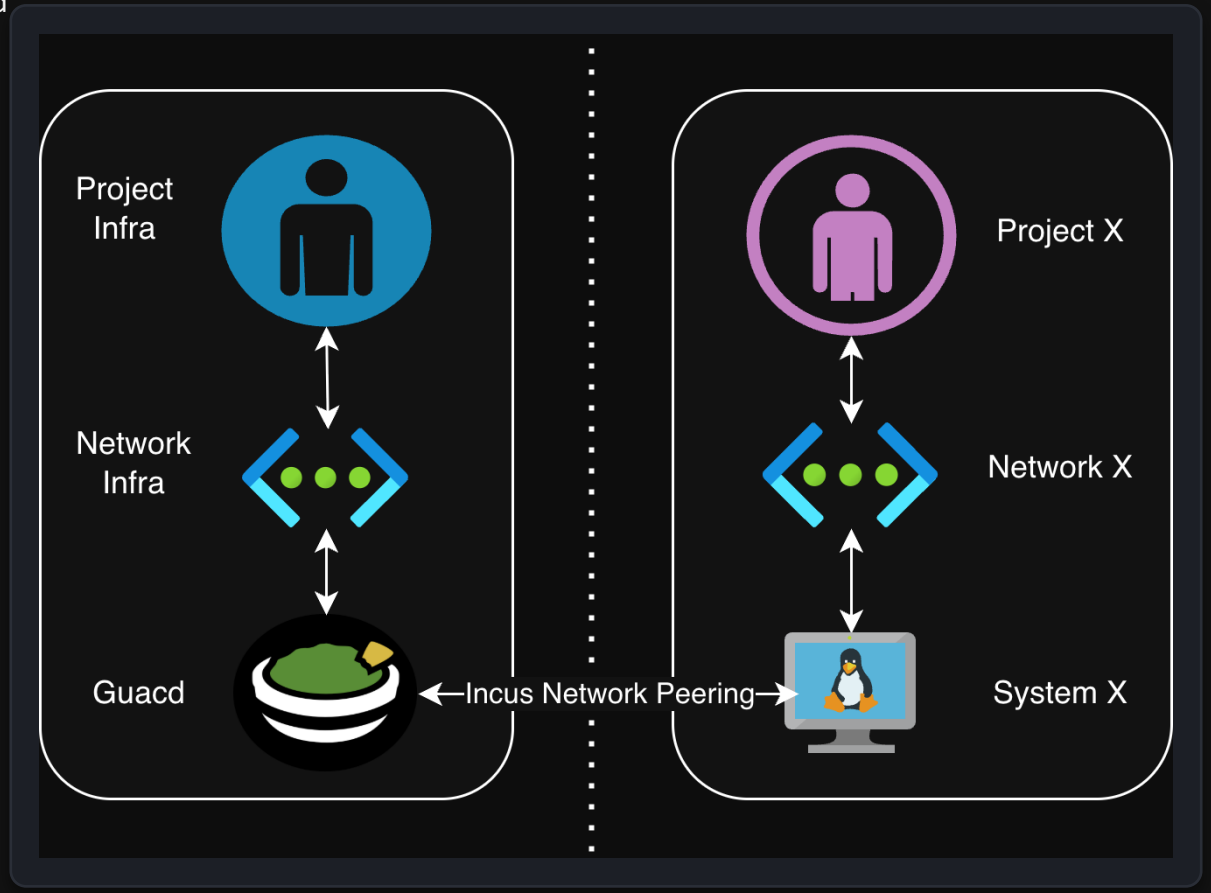


Figure 5.3 — Guacamole network peering

Peering Behavior & Security Posture

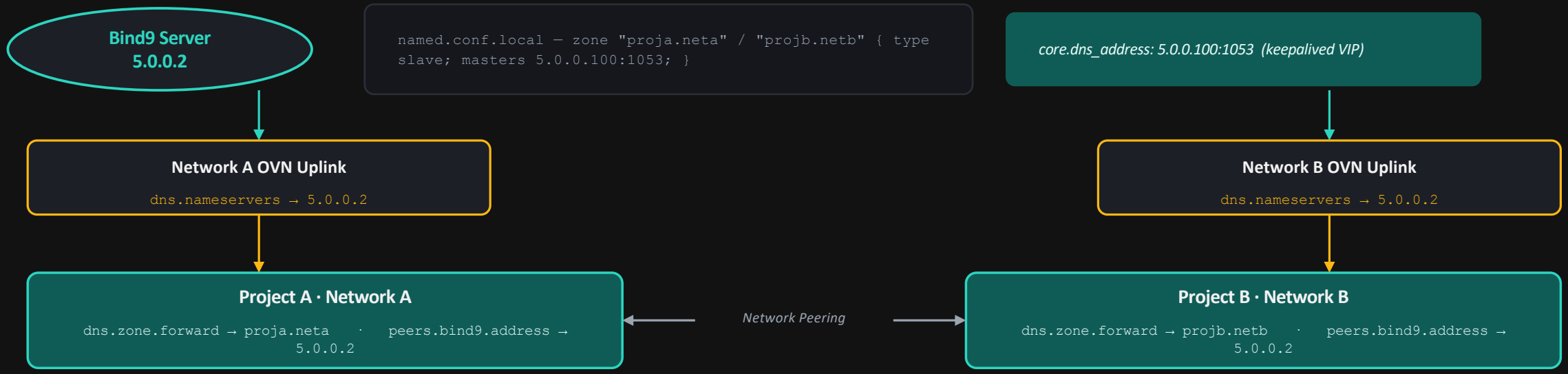
- A peer is created between the guacd management network and each tenant network that needs remote access — bidirectional, so guacd initiates sessions in and workloads can respond back.
- A peer routes traffic across the boundary, but it does not propagate DNS: each tenant network keeps its own DNS scope, so guacd never sees tenant hostnames.
  - *Connections are addressed by the instance's IP on the tenant network instead — resolved through the Incus API at session-initiation and handed to guacd as a connection parameter.*
- The peering relationship is the only traversable path in either direction: guacd is unreachable from any tenant network except through its own public HAProxy listener, and every tenant network stays isolated from every other tenant.

Resolving Guacamole Connections with DNS Zones

Why the cluster resolves instances by name — and never by a raw IP

- A DNS zone is a portion of the namespace one nameserver is authoritative for — here, one zone per tenant (proja.neta, projb.netb).
- Every Incus project network owns a forward zone (dns.zone.forward), so an instance's name stays valid even after OVN reassigns or renews its address.
- Each zone is transferred (AXFR) from Incus's built-in DNS — core.dns_address, fronted by the keepalived VIP — to a central Bind9 server, which is the resolver every instance actually uses (dns.nameservers).

– *Result: Guacamole connections reference desktop1.proja.neta, never an IP — nothing to update by hand as instances are created, destroyed, or migrated.*

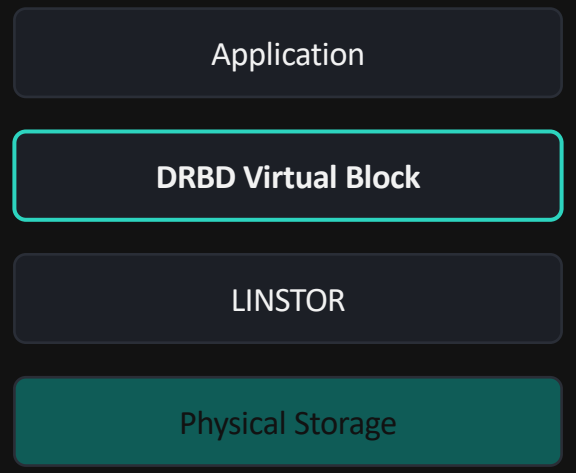


Source: cluster network-zones topology diagram

PART 2 OF 3 · COMPUTE · STORAGE · NETWORK

Storage

DRBD replication, quorum-based split-brain protection, and LINSTOR orchestration



What Is DRBD?

Distributed Replicated Block Device — synchronous replication in the kernel

- A Linux kernel module that performs synchronous or asynchronous replication of block-level writes across the network.
- Exposes a virtual block device on each node and intercepts writes, mirroring them to peer nodes before the application receives an acknowledgment.

– *RAID 1 / network RAID keep both replicas active under one application — this cluster instead runs DRBD asymmetrically.*

- One replica serves I/O as primary; the other stays a passive secondary, ready for promotion on failover.

On primary failure:

1. Failing resource demoted to secondary
2. Surviving secondary promoted to primary
3. Standby app instance started on new primary

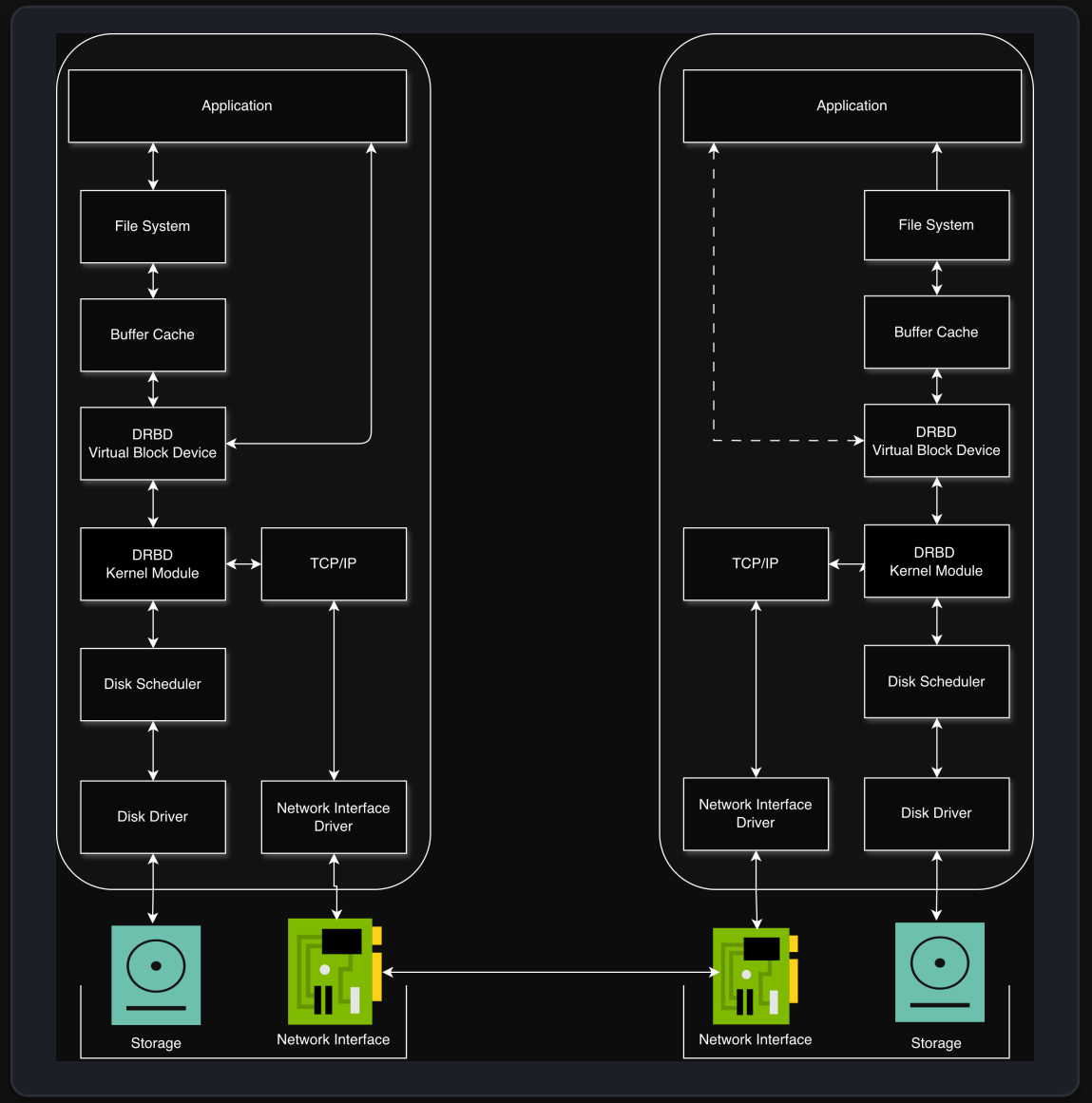


Figure 6.1 — DRBD I/O stack (primary ↔ secondary)

Quorum & Split-Brain

Why enforcing a quorum majority matters for a replicated store

- Split-brain: nodes lose communication and disconnected partitions keep operating independently.
- Uncoordinated writes on both sides of a partition produce divergent replica states that cannot be reconciled automatically.
- Quorum enforcement is the safeguard — a partition may only serve writes if it holds a strict majority of nodes.

✓ HEALTHY

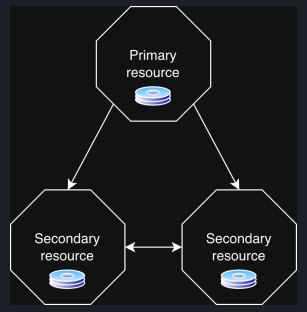


Figure 6.2 — Healthy Single Primary

One primary + two secondaries reachable — quorum satisfied, writes flow, replicas stay in sync.

✕ UNHEALTHY

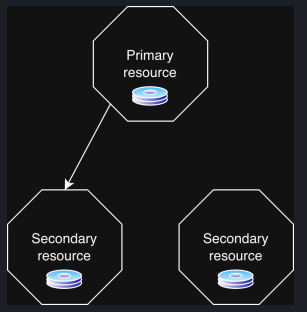


Figure 6.3 — Un-healthy Single Primary

A secondary loses the majority link — without enforcement it could self-promote, producing split-brain.

DRBD · IMPLEMENTATION (§6.0.3)

Enforcing Majority Quorum

$$q(n) = \lfloor n/2 \rfloor + 1$$

majority-quorum threshold

$$q(3) = \lfloor 3/2 \rfloor + 1 = 2$$

for this 3-node, fully-diskful cluster

- By default, a DRBD resource that loses quorum returns I/O errors to the caller — a poor fit for workloads with no defined recovery path.
- This cluster instead suspends I/O on quorum loss, holding operations until quorum is re-established — consistency is prioritized over availability.

Applied to the 3-node cluster:

3 / 3

All nodes up

Quorum satisfied — writes proceed normally.

2 / 3

Single-node failure

Majority retained — writes continue, synchronously replicated.

1 / 3

Two-node failure

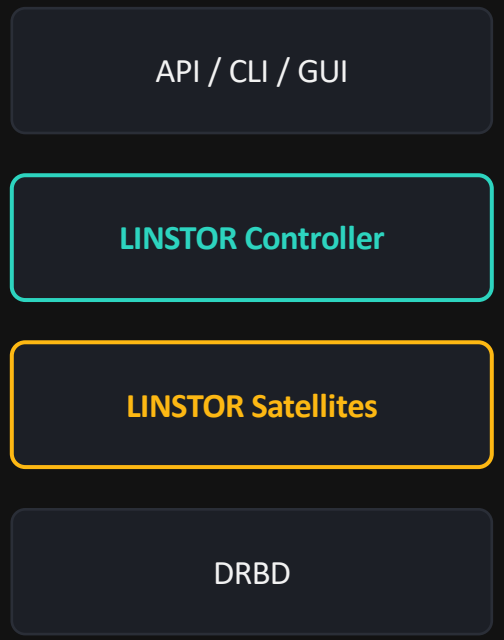
Below threshold — I/O suspended until a peer reconnects.

LINSTOR

What Is LINSTOR?

The software-defined orchestration layer above DRBD

- LINSTOR is the software-defined storage orchestration layer responsible for provisioning and lifecycle management of replicated block storage in the cluster.
- It coordinates resource groups across participating nodes and delegates the actual data replication to DRBD.
- The system follows a client–server architecture with two principal components: the Controller and the Satellite.
- Controller and Satellite communicate over a dedicated TCP/IP control plane, logically separate from the DRBD data plane — a transient controller loss does not interrupt ongoing replication or data access.



Controller & Satellite

LINSTOR Satellite

- Runs as a local agent on every node that contributes or consumes managed storage.
- Interfaces directly with the host's storage subsystem (e.g. LVM) to provision and remove backing devices.
- Configures and maintains DRBD replication relationships with peer satellites.

– *Deliberately stateless — persists no configuration, acts only on controller directives.*

LINSTOR Controller

- The authoritative, stateful component of the system.
- Maintains the cluster-wide database: node state, resource definitions, storage-pool configuration.
- Owns placement decisions, replication topology, and orchestration calls dispatched to satellites.
- Exposes the administrative CLI and API for the cluster.

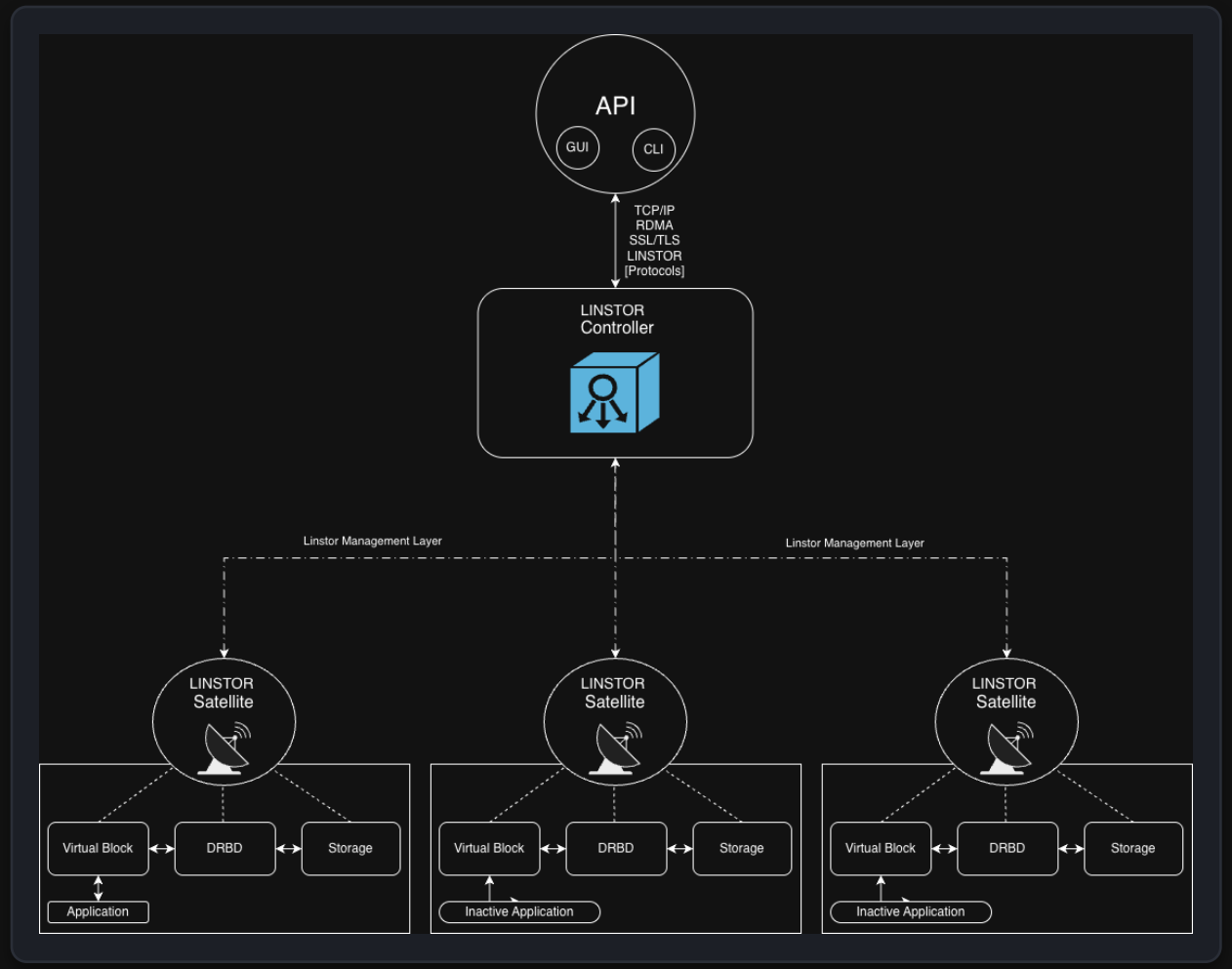


Figure 6.4 — LINSTOR client–server architecture

Highly Available Controllers

- In production, LINSTOR sits beneath Incus and the service-group workloads: DRBD performs block-level replication, LINSTOR manages resource lifecycle, placement, and satellite coordination.
- The Controller is made highly available; Three controller instances sit behind Keepalived VRRP, with only one active at a time.
- This split keeps replicated block devices available to higher-level consumers while replication, quorum behavior, and placement stay centrally managed.

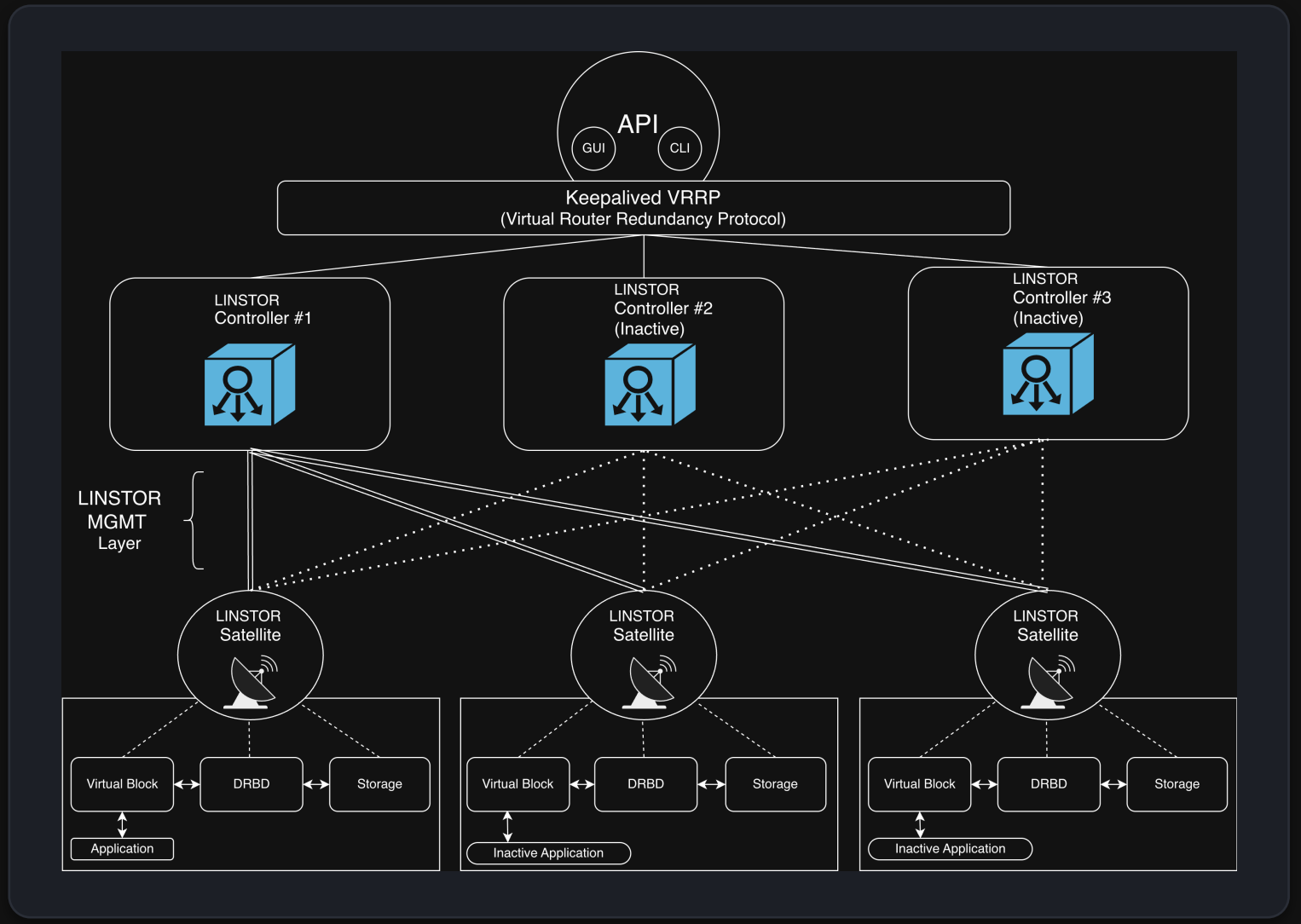


Figure 6.5 — NGLab LINSTOR deployment (VRRP-fronted controllers)

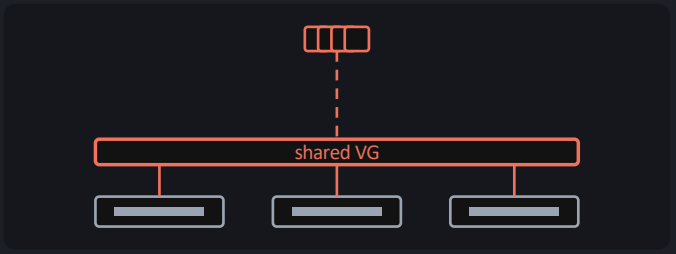
STORAGE POOL DESIGN

One Storage Pool per NVMe

Why a single volume group — and striping — fall short for VM/container I/O

Single large thin VG

(linear, per node)



- All volumes share one thin-pool's metadata — contention under concurrent I/O.
- Every workload funnels through one device queue depth — noisy neighbors across unrelated VMs.
- One exhausted or corrupted pool takes down every volume on the node.

Striping across NVMes

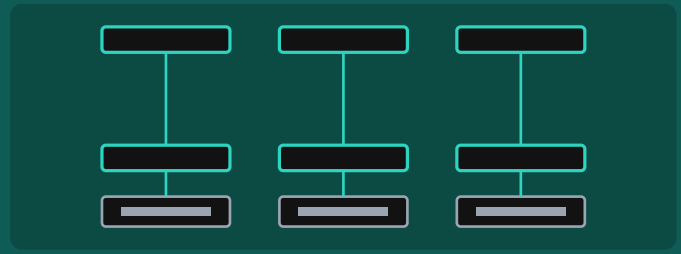
(RAID0-style)



- Ties every volume's fault domain to all drives in the stripe — one dead NVMe breaks everything on it.
- Widens DRBD's blast radius: a stripe failure invalidates many resources at once, not one.
- LINSTOR loses the ability to place a resource on one specific, isolated device.

One pool per NVMe

(adopted)



- Each physical NVMe is its own LINSTOR storage pool — a drive failure only affects volumes placed there.
- Many independent I/O paths in parallel → higher aggregate IOPS/throughput for VM and container workloads.
- Matches real workload shape: many independent volumes, not one shared pool — no cross-tenant IO interference.

PART 3 OF 3 · COMPUTE · STORAGE · NETWORK

Network

Software-defined networking with OVN, hardware-offloaded OVS, and RDMA over Ethernet

- OVN / OVS
- ASAP² Hardware Offload
- Scalable Functions
- RDMA / RoCEv2

SOFTWARE-DEFINED NETWORKING

What Is OVN?

Open Virtual Network — the logical-network layer above Open vSwitch

- Open vSwitch (OVS) is a programmable, OpenFlow-controllable software switch that runs on every host and forwards traffic according to installed flows.
- OVN (Open Virtual Network) is the logical-network management layer that sits above OVS — it lets you describe a network's desired shape instead of programming flows by hand.
- OVN is split into two databases: the Northbound and the Southbound.

Northbound

- Defines the desired logical network: switches, routers, ACLs, NAT rules, subnets.
- This is where intent is expressed — what the tenant's network should look like.

Southbound

- Translates that intent into low-level flows, tunnels, and forwarding behavior each host understands.
- Talks directly to every OVN/OVS instance so data actually moves correctly between containers, VMs, and the physical network.

Geneve: the overlay encapsulation OVN uses to carry tenant traffic between hosts — tunneling logical-network packets across the shared physical fabric.

SOFTWARE-DEFINED NETWORKING

OVN in the Incus Cluster

Benefits, and how it drives per-tenant, multi-project networking

- Incus uses OVN as its default backend for managed software-defined networks.
- Each project (faculty tenant) receives one or more OVN logical networks — each supplying its own logical-switch, logical-router, and ACL primitives.

– *This is what enforces per-tenant isolation and connectivity without hand-built VLANs per tenant.*

- Whenever Incus provisions a network, attaches an instance, or creates a network peer, it submits the change to OVN — which reprograms every OVS instance in the cluster to match.
- OVS sits on the data path of essentially every cross-host network operation: VM east-west traffic and the Geneve overlay traffic that carries it.

Declarative isolation

Per-tenant switches, routers & ACLs — no manual VLAN sprawl.

Automatic propagation

One API call reprograms OVS across the whole cluster.

Consistent multi-tenancy

Same primitives scale from one project to many.

The Kernel Data Path

- Packets traverse a fast-path cache implemented as a Linux kernel module (green path in the figure).
- A cache miss is upcalled to ovs-vswitchd in userspace, which computes and installs the corresponding kernel flow so future matching packets stay in-kernel.

Benefits

- Requires no special hardware — runs on any NIC.
- Fast-path cache means most packets never leave the kernel.

Downsides

- Still incurs the per-packet cost of the kernel network stack — measurable at 100 GbE line rates.
- Every matched packet still consumes host CPU cycles that GPU/VM workloads also need.

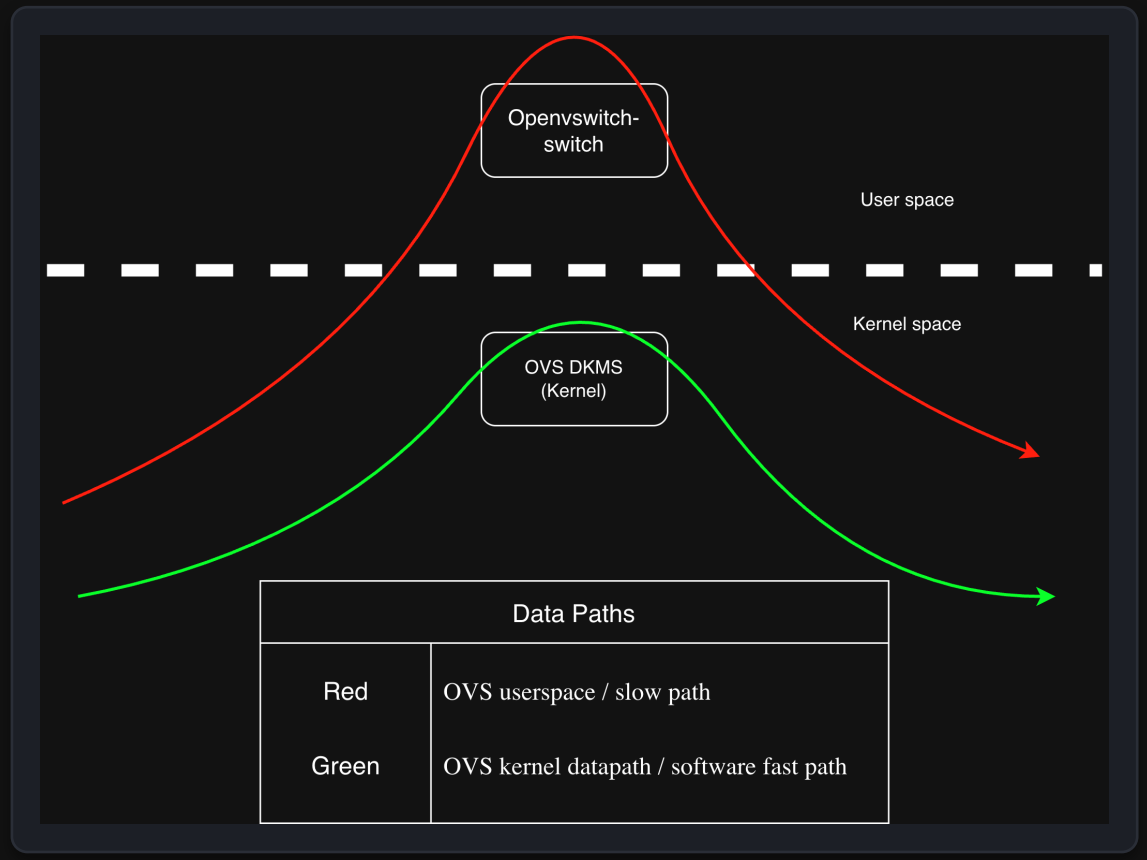


Figure 7.1 — green: OVS kernel datapath (fast path)

The Userspace Data Path

- Packets traverse a software slow-path implemented in the ovs-vswitchd module in userspace (red path in the figure).
- This is the central design problem for the cluster: the same host CPUs are also expected to serve VMs, system containers, and GPU-bound workloads.

Benefits

- Most flexible path — full software logic, easiest to extend and debug.
- The blueprint NVIDIA's ASAP² (Accelerated Switching and Packet Processing) offload later relocates into hardware, without changing the OVS control plane.

Downsides

- Consumes host CPU cycles directly — any cycle spent forwarding is one not available to workloads.
- Scaling it further (e.g. OVS-DPDK) means permanently dedicating CPU cores to polling.

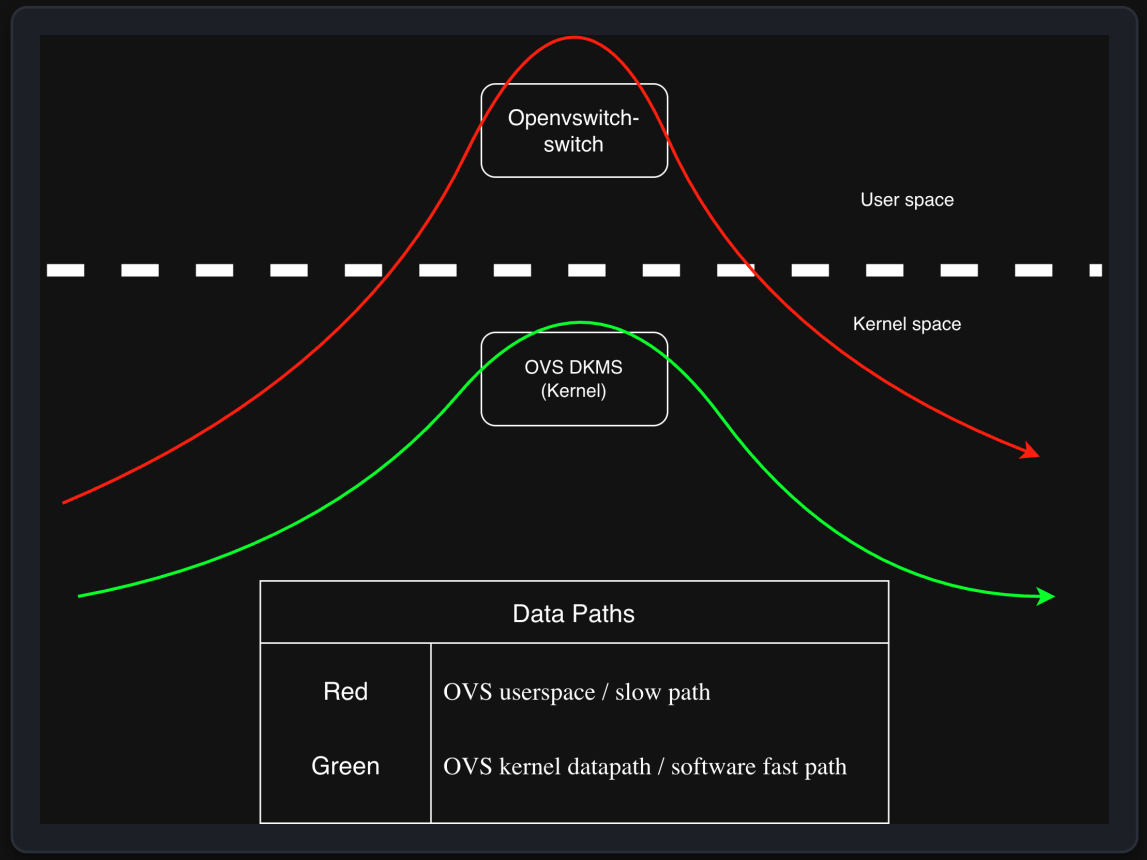


Figure 7.1 — red: OVS userspace datapath (slow path)

The Hardware Offloaded Data Path

- Moves the forwarding decision out of host software entirely — matched traffic is forwarded by NIC silicon (blue path), with zero host CPU involvement.
- Linux exposes this through switchdev mode: the eSwitch is programmed via standard kernel interfaces, and each NIC-backed endpoint gets a host-visible representor port.
- The OVS control plane is unchanged — flows are still computed the same way. Only the data plane silently relocates to the NIC.

What is the e-switch?

The embedded switch inside the NIC's silicon — a hardware forwarding table OVS can program directly. It's the piece that lets a "SmartNIC" host an actual switch, not just a wire.

Why use it: it's the only place matched traffic can be forwarded without ever waking the host CPU — giving full line-rate throughput back to VM, container, and GPU workloads.

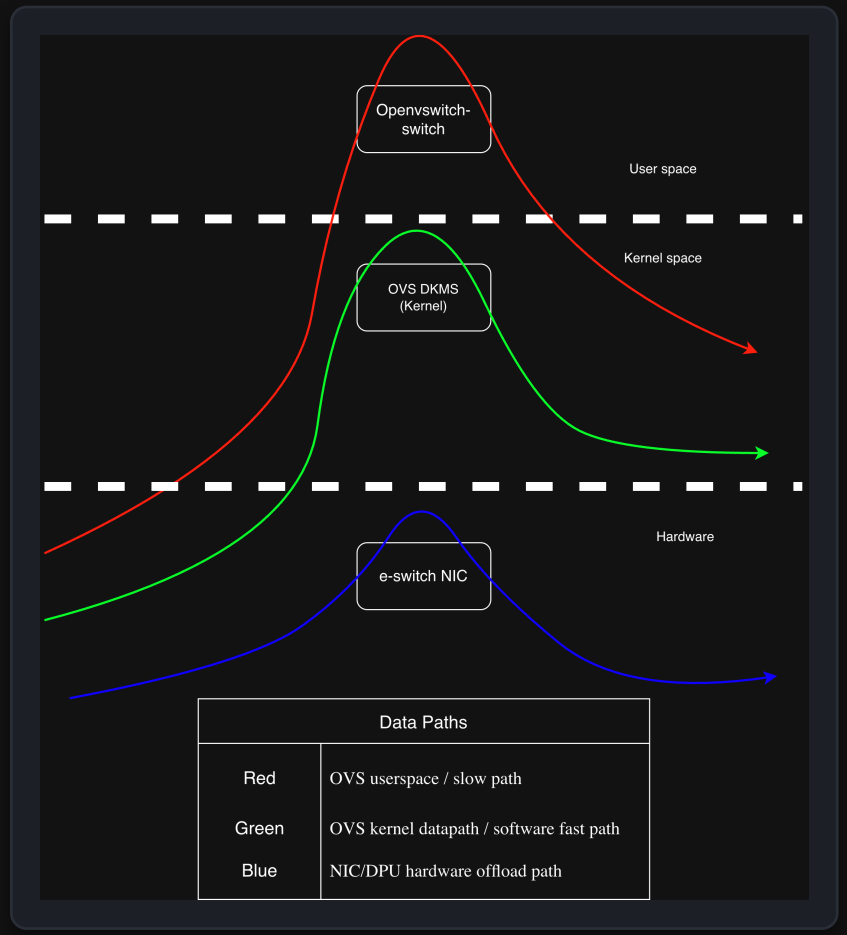


Figure 7.2 — blue: NIC/eSwitch hardware offload path

SR-IOV Virtual Functions

The classical way to expose multiple hardware-backed endpoints from one NIC

- Single Root I/O Virtualization: the NIC presents one physical function (PF) and a fixed number of virtual functions (VFs), each appearing as its own independent PCIe device.
- In switchdev mode, each VF is paired with a representor that OVS programs hardware flows against.
- Mature and strongly isolating — but bounded and inflexible for this deployment.

✕ Downsides for this cluster

- VF count is bounded at adapter initialization, and further constrained by host BIOS, IOMMU, and PCIe-resource limits.
- Enabling SR-IOV needs firmware-level configuration that doesn't accommodate a multi-tenant cluster's changing topology.

– *Biggest issue: no live migration. VFs are tied to a physical I/O path, so only cold migration is possible — hurting failover (§7.4.4.1).*

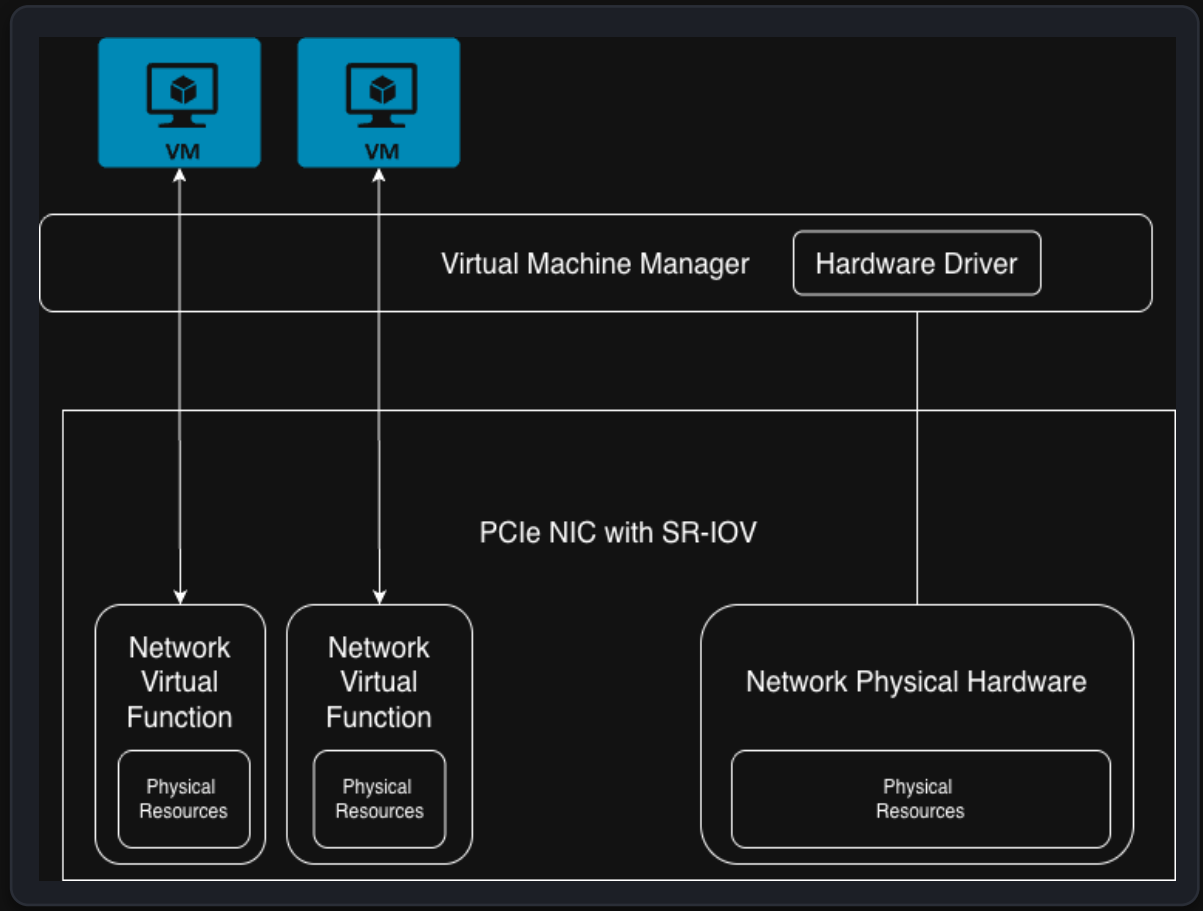


Figure 7.3 — SR-IOV Virtual Functions

Scalable Functions (SFs)

The lightweight successor to SR-IOV VFs adopted in this deployment

- Sits conceptually between a full physical NIC and an SR-IOV VF — its own network identity plus dedicated TX/RX queues, without needing a distinct PCIe function or SR-IOV enabled at all.
- Created and destroyed dynamically through the Linux devlink interface — the per-port count is far larger than the VF limit.
- Fully integrated with the mlx5 eSwitch and switchdev offload model: creating an SF spins up a representor on the host that OVS observes and governs, wiring the SF straight into the accelerated data path.

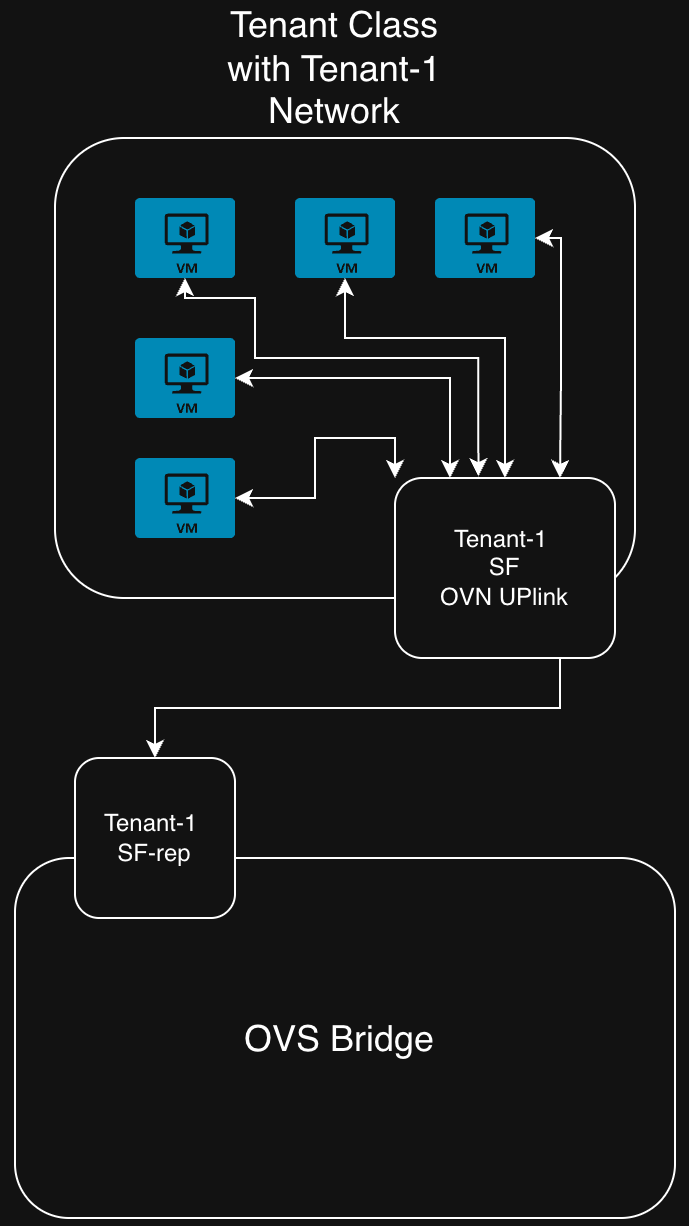


Figure 7.4 — a tenant's SF acting as its OVN uplink

SCALABLE FUNCTIONS · SETUP

Wiring an SF into ASAP²

Attaching hardware offload isn't automatic — it's a specific 4-step sequence

1

Switchdev mode

Put the parent PCIe function into switchdev mode. The mlx5 driver exposes the NIC's eSwitch and begins emitting representors for subordinate functions.

2

Instantiate the SF

Create the SF against the parent function via devlink. This produces the SF itself plus a paired representor netdev — the pairing can't be reassigned later.

3

Enable hw-offload

Set hw-offload=true on the OVS bridge and restart ovs-vswitchd so the TC-flower offload backend initializes. Without this, the eSwitch stays unused.

4

Attach to the bridge

Add the representor to the OVS bridge. OVS computes flows and — because offload is already on — pushes them straight into the eSwitch.

Order matters: attaching a representor before offload is enabled locks in software-only flows — the bridge must be restarted (or the port re-attached) to promote them to hardware.

NETWORK VIRTUALIZATION · DESIGN ARGUMENT

Why Scalable Functions

The per-tenant uplink mechanism has to satisfy two constraints at once

SR-IOV VF

doesn't scale to tenant density

- Each faculty project needs its own dedicated, hardware-backed uplink — uplink count tracks tenant and workload count.
- A dense multi-project deployment would exceed the practical VF budget the adapters and BIOS/PCIe limits can support.

OVS-DPDK

burns cores the cluster needs

- Resolves per-packet overhead, but only by permanently dedicating CPU cores to polling forwarding threads.
- Those cores are the same ones GPU and VM workloads depend on — a direct trade against the cluster's purpose.

Scalable Functions

(adopted)

- Created dynamically via devlink — not constrained by SR-IOV-specific BIOS/PCIe limits, so uplink density scales natively.
- ASAP² hardware offload gets OVS-DPDK's throughput without paying its CPU cost.

Result: Incus/OVN stay the orchestration layer, OVS stays the policy layer, and ASAP² + SFs deliver the actual line-rate forwarding — cleanly separated.

What Is RDMA?

Remote Direct Memory Access — bypassing the host on the data path

- A network protocol that lets one host access memory on another host without the remote OS or CPU participating in the data movement.
- In traditional TCP/IP, a transfer pays a latency tax: data passes through the full networking stack, gets copied between user buffers, and costs CPU cycles on both ends.
- RDMA reduces CPU overhead, lowers latency, and moves data near the physical hardware limit — critical for storage replication, distributed databases, and GPU cluster traffic.

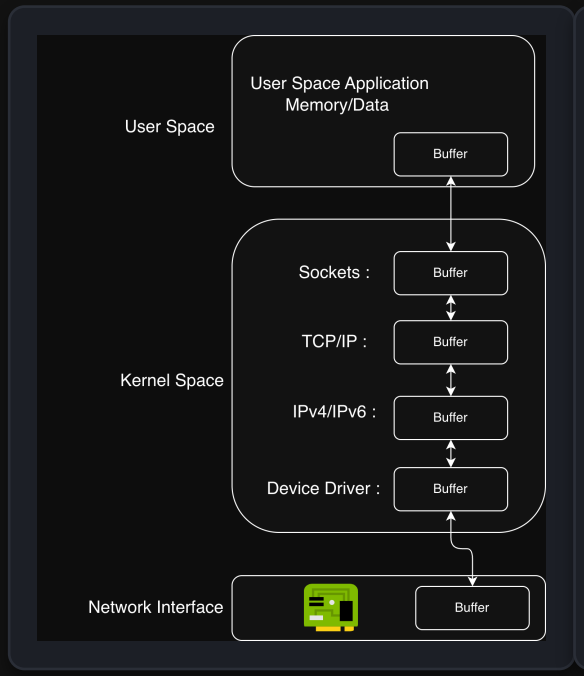


Figure 8.1 — TCP

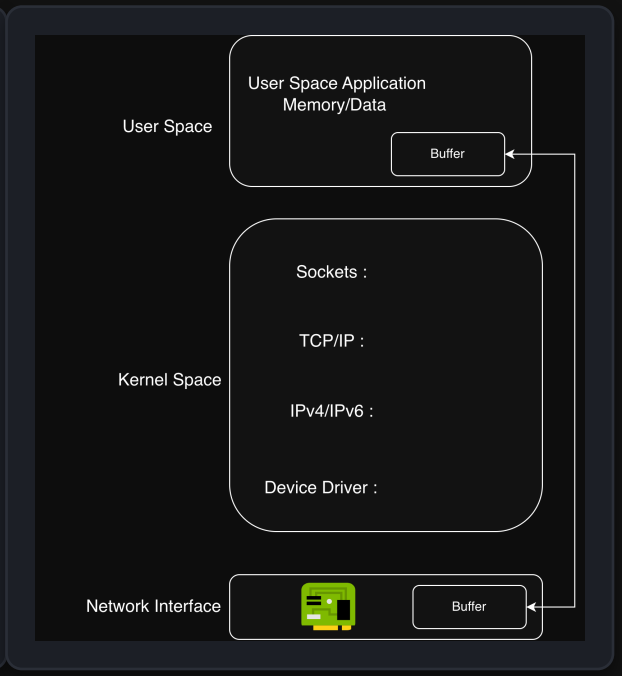


Figure 8.2 — RDMA

TCP copies data up through sockets, TCP/IP, and the device driver on both hosts. RDMA writes straight from the NIC's buffer into application memory — the kernel stack is never touched.

Properties of an RDMA Packet

1

Self-describing

The transport header carries an opcode (RDMA WRITE, RDMA READ, SEND, or an atomic op), a remote memory key, and — for one-sided operations — a remote virtual address. The receiving adapter has everything it needs to complete the operation without ever consulting the host.

2

Transport state lives on the adapter, not the host

Sequence numbers, acknowledgment windows, and connection state sit in hardware queues on the NIC, not in kernel data structures. Protocol processing runs at the adapter's clock rate rather than at host-interrupt speed.

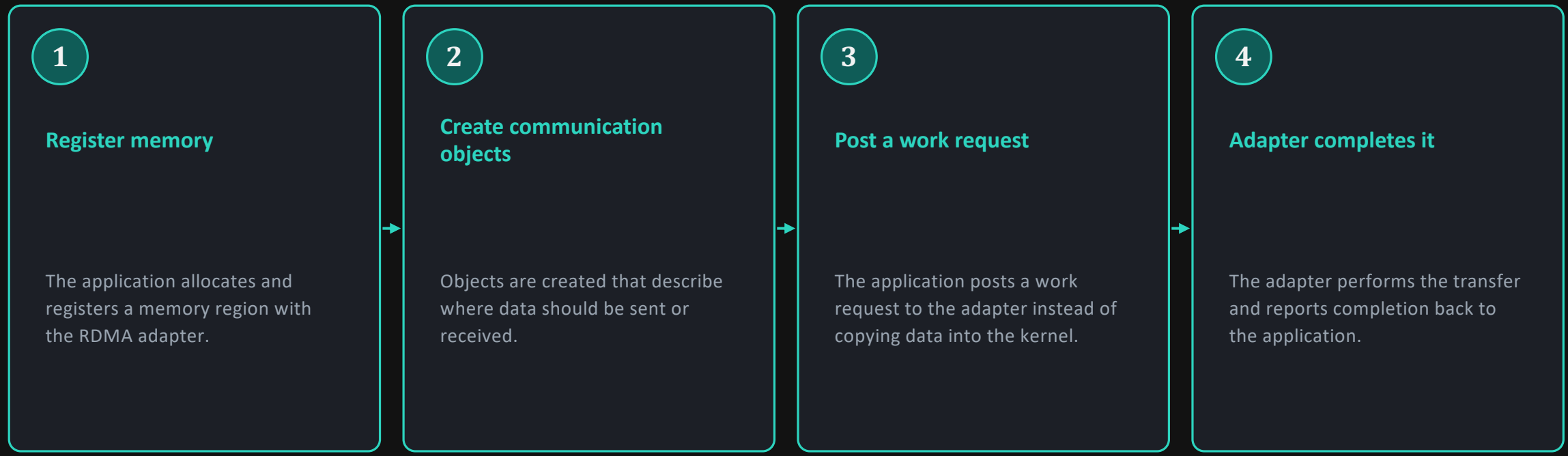
3

Connected to memory, not a socket

The destination address is encoded in the packet itself, so the receiver's role shrinks to a hardware address translation plus a DMA write. No system call, no scheduling event, no copy stands between the wire and the application's address space.

How RDMA Works

Four steps move the work from the application to the adapter



This design moves nearly all the communication work off the CPU and into dedicated network hardware.

Queue Pairs (QP)

The core communication abstraction in RDMA

- A QP represents a communication endpoint between two RDMA-capable systems — it's the connection itself, not just a channel riding on top of one.
- Each QP holds two independent queues: a Send Queue for outgoing work requests, and a Receive Queue that holds buffers or requests for incoming messages.

– *In simple terms: a QP is a software object that posts requests and keeps the connection alive.*

- An application places work on a QP through verbs calls (ibv_post_send / ibv_post_recv) — the adapter drains the queue and executes the operation in hardware.
- Each QP is paired with exactly one QP on the remote NIC — that pairing is what makes it a connection, not just a queue.

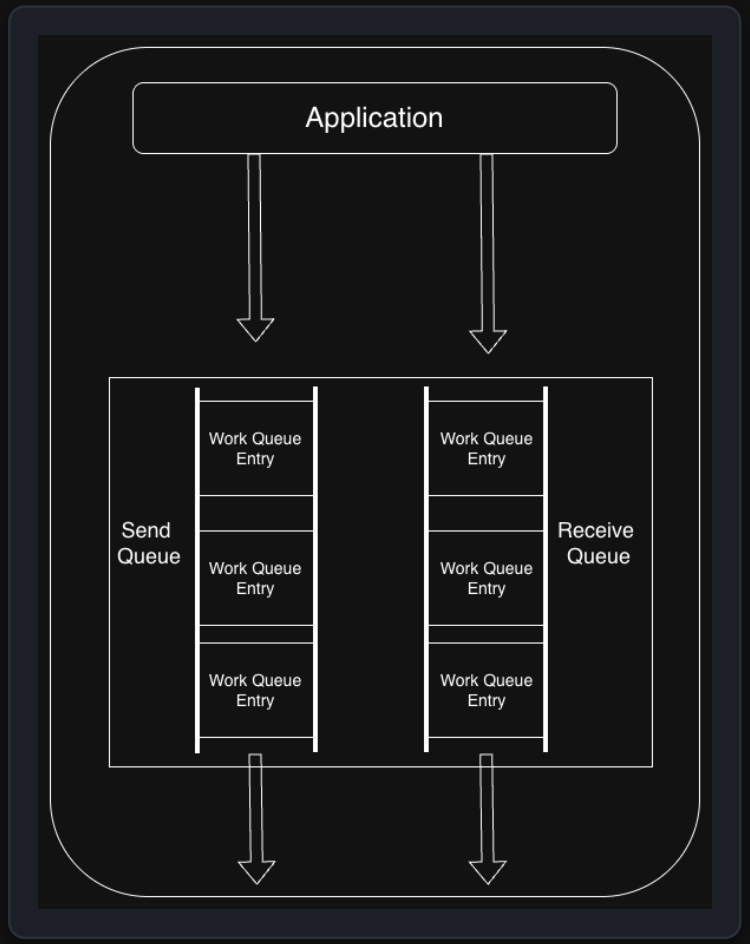


Figure 8.3 — Queue Pair: Send Queue + Receive Queue

Completion Queues (CQ)

How the adapter reports that a work request actually finished

- A CQ is the RDMA NIC's way of telling the application that a posted work request has completed — the closing half of the four-step RDMA flow.
- Applications learn about completions one of two ways:
  - *Polling — the app repeatedly checks the CQ itself. Lowest latency, but spends CPU cycles even when idle.*
  - *Interrupt-driven — the app blocks and the NIC raises an interrupt when a completion lands. Saves CPU at the cost of a little latency.*
- Each completion queue entry (CQE) reports which work request finished, its status (success or error), and how much data moved.
- A single CQ can service completions from multiple QPs — one place to drain instead of polling every connection separately.

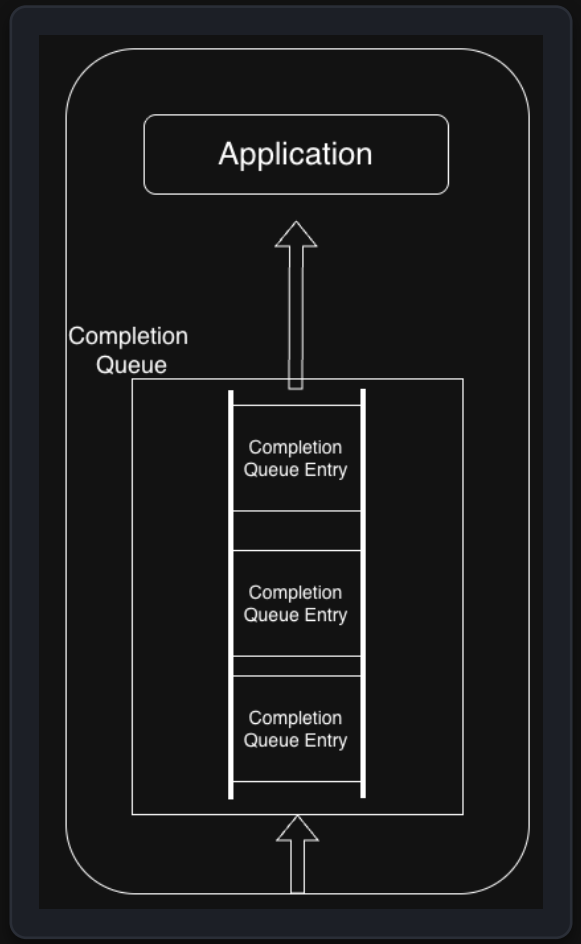


Figure 8.4 — Completion Queue

Memory Region (MR)

The pinned memory the adapter is allowed to touch directly

- An MR is a designated segment of host memory the RDMA NIC can read and write directly — size varies with scope of use.
- Once registered, the region is pinned to physical memory: the OS is barred from paging it out mid-transfer, so the NIC's physical address mapping stays valid for the life of the operation.
- Registration also produces a start/end key identifying the region — the NIC translates between that key and the region's virtual/physical address to run the DMA with zero CPU involvement.
- Every MR carries two keys:

– *L-Key — used when the RNIC receives the request from the host side.*

– *R-Key — used when the RNIC receives the request from the wire (a remote peer, for one-sided operations).*

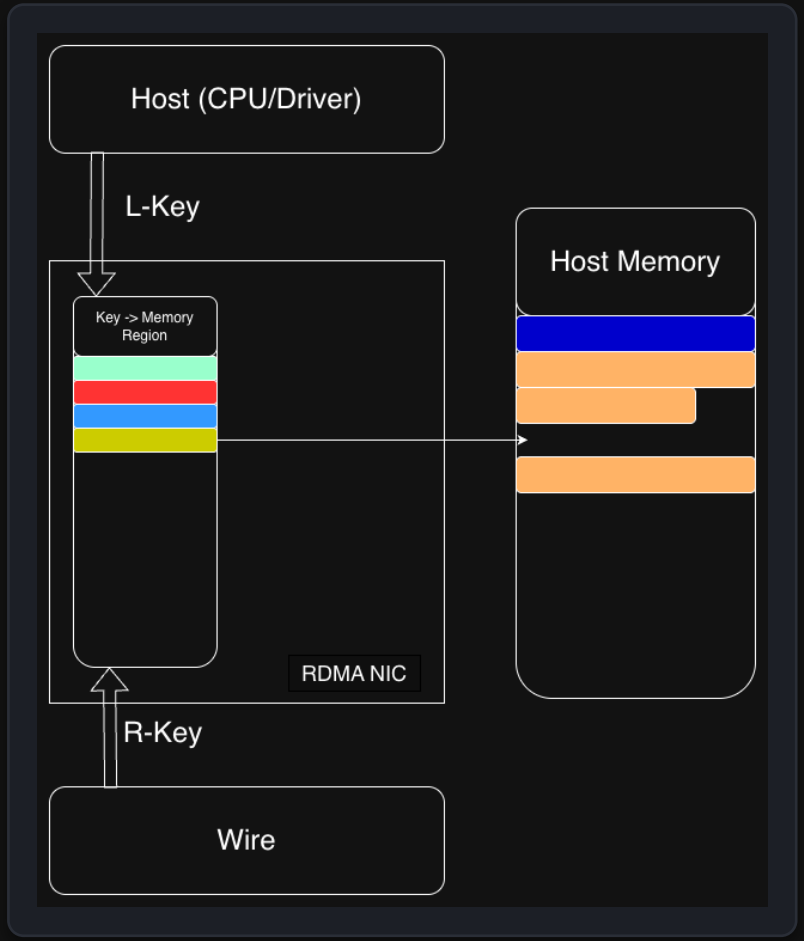


Figure 8.5 — Memory Region: L-Key / R-Key

RDMA & ROCEV2

RoCE v1 vs. RoCE v2

RDMA over Converged Ethernet — bringing RDMA onto the fabric we actually have

- RDMA was originally tied to specialized fabrics like InfiniBand. RoCE extends the same low-latency, low-CPU model onto Ethernet — the fabric deployed in this infrastructure.

RoCE v1

- RDMA carried directly over Layer 2 Ethernet.
- Not routable — confined to a single Ethernet broadcast domain.

RoCE v2 (used here)

- Extends the model across routable UDP/IP networks.
- Supports scalable, routed, and isolated infrastructure while preserving RDMA's performance advantages — why it's the relevant protocol for this cluster.

Challenges Compared to Traditional Ethernet

RoCE needs a network engineered more carefully than plain TCP/IP Ethernet

Lossless behavior

RoCE leans on Priority Flow Control (PFC) to pause a priority before it drops packets — but pausing can spread congestion across the fabric.

Congestion control

Without ECN marking to slow senders down, buffers overflow and packets are lost anyway.

Operational complexity

MTU, trust mode, DSCP, PCP, PFC, and ECN must all agree across servers, NICs, and switches — a mismatch looks "connected" but silently breaks RDMA.

Lossy vs. Lossless vs. Semi-Lossless

Lossy

Ordinary best-effort Ethernet. Buffers overflow, packets drop, TCP retransmits. Fine for normal traffic — fatal for RDMA, which has no equivalent software retry path.

Semi-lossless

ECN marking only: packets are tagged to ask senders to slow down before buffers fill. Reduces drop probability but doesn't guarantee zero loss on its own.

Lossless

PFC pauses a priority class entirely before its buffer overflows — a hard guarantee of zero drops for that class. This is what RoCE actually requires end-to-end.

Making the Fabric Lossless, End to End

Every hop — host and switch — has to agree on priority, MTU, and PFC

Host side (Debian)

- Install rdma-core, ibverbs-utils, perftest — the verbs libraries applications need to open RDMA devices.
- Bring up a dedicated RDMA VLAN/interface with 9000-byte jumbo frames.
- Force egress RDMA traffic onto priority 3 (PCP/DSCP 26) — the cluster's lossless RoCE class.
- Increase socket, backlog, and NIC ring buffers so bursts don't get dropped before RDMA even engages.

Switch side (Mellanox SN2700)

- Dedicated VLAN (10) for RDMA — isolates replication traffic from management/tenant traffic.
- Trunk + LACP port-channels carry the RDMA VLAN with redundancy and aggregate bandwidth.
- Global PFC enabled and forced on priority 3 — the same lossless class the host uses.
- Per-port: trust L3 and map DSCP 26 → switch-priority 3, so the switch honors the host's marking.

PFC storm watchdog: if a stalled host or bad port holds a pause on priority 3, back-pressure can propagate hop-by-hop into a PFC storm. The watchdog detects sustained pauses and locally drops traffic on that queue instead of letting it spread.

CHAPTER 9 · EVALUATION

Results

Verifying RDMA, hardware offload, and lossless network performance across the cluster

- iperf3 — Network
- fio — Storage
- Prometheus — Metrics
- RDMA vs. TCP

EVALUATION OBJECTIVE

What We're Verifying, and Why

1

RDMA is correctly configured

That the RoCEv2 stack — PFC, DSCP mapping, trust mode, QoS on every Scalable Function — actually behaves the way it’s designed to.

2

The data path is offloaded to hardware

That traffic is really moving through the NIC's embedded switch (e-switch), not silently falling back to the kernel or userspace path.

3

The network stays lossless under load

That the fabric holds a stable, lossless connection under sustained load — not just in a short burst test.

Why it matters: Chapters 6–8 made a design argument for Scalable Functions, ASAP² offload, and RoCEv2 over the simpler alternatives. That argument is only worth anything if it's measured, not assumed — this chapter is where the design either earns its complexity or doesn't.

EVALUATION OBJECTIVE

Experimental Setup

Three identically-configured nodes, isolated interfaces per traffic class

Supermicro

Diskless — compute plane

3 dedicated, subnet-isolated interfaces

r730xd-1

Dell PowerEdge — storage / OVN

3 dedicated, subnet-isolated interfaces

r730xd-2

Dell PowerEdge — storage / OVN

3 dedicated, subnet-isolated interfaces

Table 9.1 — Per-node interface assignment and MTU

Interface role	Network layer	MTU (bytes)
LINSTOR management	Physical uplink	9216
Incus management	Physical uplink	9216
DRBD data layer (RDMA)	Physical uplink	9216
Incus virtualised network	Geneve (OVN)	9116

Why this arrangement

The DRBD resource is created on LINSTOR and manually attached to the diskless Supermicro node, which acts as a tie-breaker. Two data replicas stay synchronized across the storage nodes — letting the physical replication path and the container-facing block device be tested independently.

MEASUREMENT STRATEGY

Two Variations, Two Vantage Points

- Two test variations were defined (baseline vs. fully offloaded + RDMA), each capturing two sets of six key metrics — letting performance be observed on the local and remote node simultaneously.

Physical infrastructure set

- Collected directly from the physical infrastructure.
- Exposes the true behavior of RDMA and hardware offloads at the system and hardware level as data actually moves across the fabric.

Container set

- Collected from two containers, thesis-1 and thesis-2.
  - *thesis-1 sits on the Supermicro node; thesis-2 sits on a separate R620 — forcing traffic across the physical network instead of a local loopback.*

Six metrics per set (detailed next): network transmit/receive bytes, disk read/write bytes, and disk read/write operations completed — mirrored across every layer measured.

MEASUREMENT TOOLS · NETWORK

Throughput & Latency: iperf3

A controlled TCP stream between server and client, sustained long enough to trust

```
# Server
iperf3 -s

# Client
iperf3 -c <Server IP> -P 128 -Z -t 600
```

`-P 128` — 128 parallel streams — needed to fully saturate the link and expose any per-flow bottleneck.

`-Z` — Zero-copy mode — data moves straight from the buffer without an intermediate userspace copy, mirroring the offloaded data path under test.

`-t 600` — 600-second sustained window — confirms throughput holds steady and the connection stays lossless, rather than just peaking momentarily.

What this tests

- One node listens in server mode; the other connects as a client and drives the load.
- 128 parallel streams saturate a 100 GbE fabric far better than a single flow ever could.
- The 600 s window is long enough to reveal a PFC storm, a throughput collapse, or any instability a short burst test would hide.
- The same test is run once per variation, and once per vantage point (physical host, then container) — so the numbers are directly comparable.

MEASUREMENT TOOLS · NETWORK

Throughput & Latency: ib_write_bw & ib_read_bw

Tool used to test RDMA bypass capabilities and amount of data we can flood through pipeline at once

```
# Server
ib_write_bw -d <rdma-interface> -x 3 -a -F

# Client
ib_write_bw -d <rdma-interface> -x 3 -a -F <Server IP>
```

- Tests the full PCIe-to-network saturation by pushing raw chunks directly between the memory layers.

```
# Server
ib_read_bw -d <rdma-interface> -x 3 -a -F

# Client
ib_read_bw -d <rdma-interface> -x 3 -a -F <Server IP>
```

- Tests the full PCIe-to-network saturation by pulling raw chunks directly between the memory layers.

-d <rdma-int>	Interface being used for RDMA
-x	Global identified Index that is found from “show_gids | grep –E RoCE v2|RoCEv2 ”. For our setup this is RoCE v2 (IPv4) using standard UDP encapsulation to route cleanly.
-a	Tells tool to run automated sweep across multiple packet sizes to see where fabric hits maximum efficiency.
-F	Forces bidirectional send/receive simultaneously.

MEASUREMENT TOOLS · NETWORK

Throughput & Latency: ib_write_lat & ib_read_lat

Tool used to test RDMA bypass capabilities and turnaround speed for data

```
# Server
ib_write_lat -d <rdma-interface> -x 3 -C

# Client
ib_write_lat -d <rdma-interface> -x 3 -C  <Server IP>
```

```
# Server
ib_read_lat -d <rdma-interface> -x 3 -C

# Client
ib_read_lat -d <rdma-interface> -x 3 -C  <Server IP>
```

- Tests the time it takes for initiating interface to post a small packet directly into the remote target memory.

- Tests the two-way turnaround time and chip delay . Since RAM seek times are included from target internal PCIe bus.

-d <rdma-int>	Interface being used for RDMA
-x	Global identified Index that is found from “show_gids | grep –E RoCE v2|RoCEv2 ”. For our setup this is RoCE v2 (IPv4) using standard UDP encapsulation to route cleanly.
-C	Prints out highly detailed, histogram table showing percentile distribution of network latencies.

Storage Performance: fio

Small-block random I/O at deep queue depth — the workload NVMe fears most

- fio issues synthetic I/O against a target device and reports throughput, IOPS, and latency.
- 4 KiB random reads/writes at deep queue depth were chosen because they're one of the most demanding patterns for NVMe media — and the pattern that most clearly exposes the limits of the replication path itself.

– *Physical target: /dev/drbd1133 · Container target: /dev/sdb*

```
fio --name=randread-4k --filename=/dev/drbd1133 \
    --ioengine=libaio --direct=1 --rw=randread \
    --bs=4k --numjobs=20 --iodepth=512 \
    --runtime=300s --time_based --group_reporting
```

`--direct=1`

Bypasses the page cache — measures true device/replication performance, not cached results.

`--iodepth=512 × --numjobs=20`

Deep, parallel demand — enough in-flight requests to actually saturate the NVMe drives.

`--group_reporting`

Aggregates all twenty workers into one consolidated report.

Baseline for comparison: the Micron 7300 MAX 800 GB — rated 220K random-read / 60K random-write IOPS at 4 KiB, QD512. Every result in this chapter is measured against that ceiling.

TEST DESIGN

Three Progressive Variations

Each variation isolates the contribution of one acceleration mechanism

#	Configuration	HW Offload (ASAP²)	RDMA
1	Baseline (software path)	No	No
2	Fully offloaded	Yes	No
3	Fully offloaded + RDMA	Yes	Yes

Table 9.2 — Progressive test variations isolating each acceleration mechanism

Variation 1 — Baseline

All traffic follows the conventional path through userspace and the kernel networking stack. This vanilla configuration is the reference point for how the network performs on the default software data path.

Variation 2 — Fully Offloaded

The data path moves off the software route and onto hardware to the greatest extent possible. Expected to deliver the highest throughput and lowest latency — and forms half of the central result of this evaluation.

Variation 3 — Fully Offloaded + RDMA

The data path moves off the software route and onto hardware to the greatest extent possible; RDMA additionally lets one node write directly into a remote node's memory without involving the remote OS or CPU. Expected to deliver the highest throughput and lowest latency — and forms the full central result of this evaluation.

MEASUREMENT TOOLS · METRICS

Metric Collection

Why RDMA traffic needs its own exporter

- The standard metrics exporter watches traffic through the Linux kernel network stack. RDMA bypasses most of that path, so those metrics alone don't represent RDMA traffic — a dedicated Prometheus RDMA exporter was required.

RDMA (Prometheus RDMA exporter)

- rdma_port_rcv_data_total — total bytes received by the RDMA port.
- rdma_port_xmit_data_total — total bytes transmitted by the RDMA port.
- rdma_port_rcv_packets_total — total packets received by the RDMA port.
- rdma_port_xmit_packets_total — total packets transmitted by the RDMA port.

Non-RDMA (Prometheus Node Exporter)

- node_network_transmit_bytes_total / node_network_receive_bytes_total
- node_disk_read_bytes_total / node_disk_written_bytes_total
- node_disk_reads_completed_total / node_disk_writes_completed_total

In the virtualized environment, the same categories (network, disk, resource) are collected via the Incus Prometheus exporter — so virtual-layer results stay comparable to the host-level numbers.

VARIATION 1 · BASELINE · NETWORK

Network — Physical Layer

- The system is bottlenecked by a network path that cannot offload — every packet is classified, switched, and copied by the host CPU.
- Measured throughput held around 8–9 Gb/s on a NIC rated for 100 Gb/s.
- Reaching just 8–9% link utilization already cost the host almost half its compute resources.

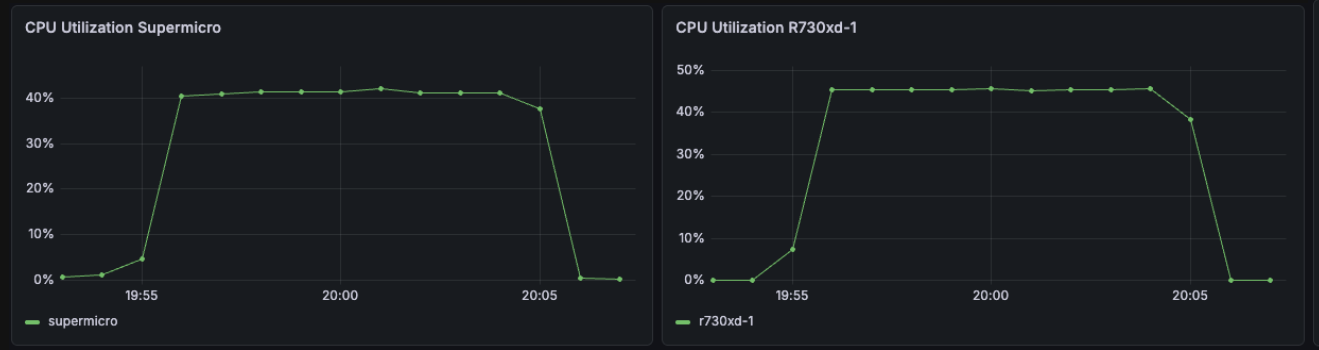


Figure 9.2 — CPU Utilization, 128 Parallel Streams

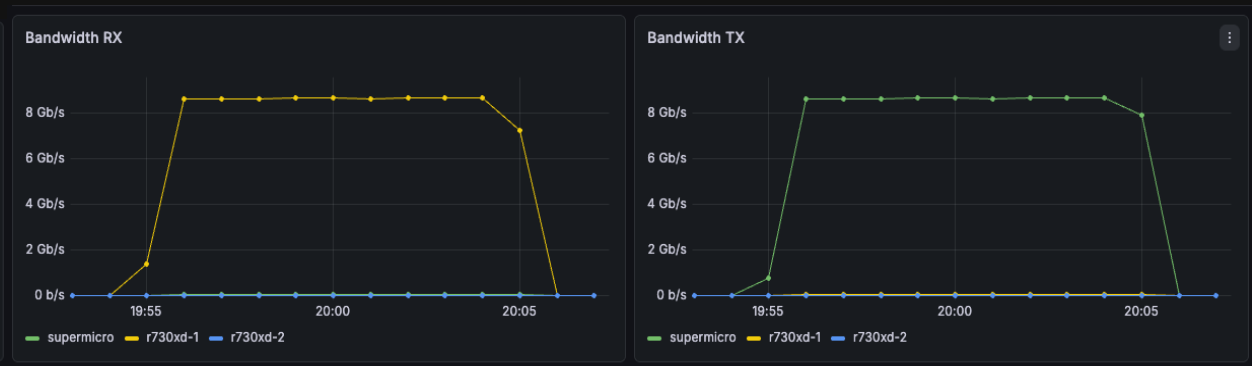


Figure 9.3 — Bandwidth Speed Test

~45%
avg CPU across cores

8–9 Gb/s
of 100 Gb/s line rate

8–9%
link utilization achieved

VARIATION 1 · BASELINE · NETWORK

Network — Container Layer

- Same bottleneck pattern, compounded: traffic now also crosses the virtual networking layer used by the container environment.
- Between the two test containers (thesis-1, thesis-2), measured throughput barely reached 1.5 Gb/s — roughly 10% of what the host itself could push.
- The containerized configuration adds real, measurable overhead on top of the existing host-level bottleneck rather than absorbing it.

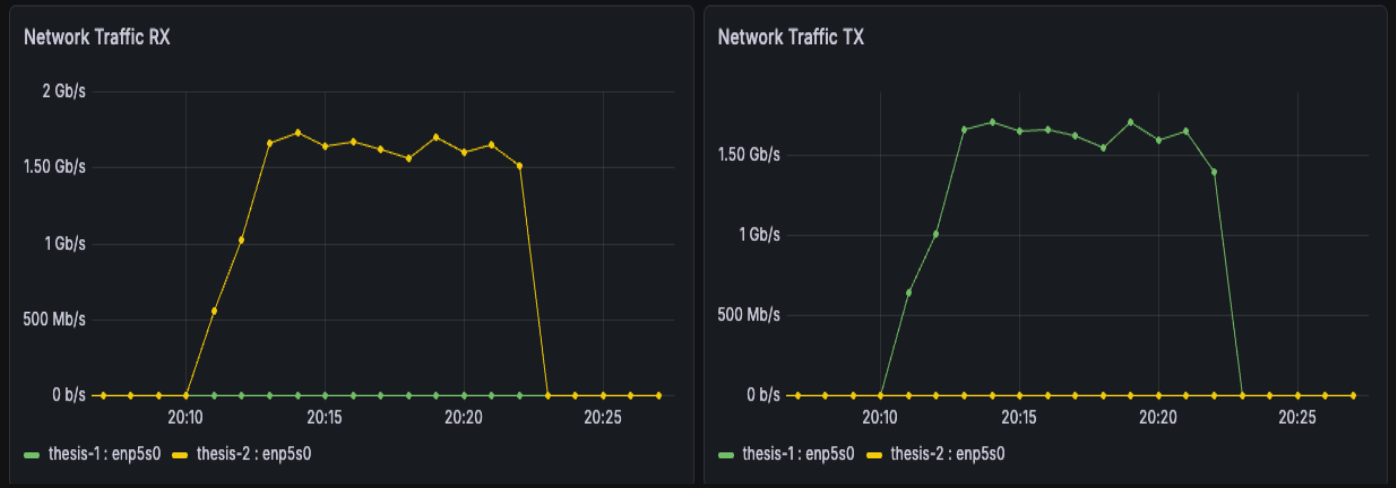


Figure 9.4 — Bandwidth Speed Test (Container Layer)

~1.5 Gb/s
thesis-1 ↔ thesis-2 throughput

~10%
of host-level throughput

2 layers
host stack + virtual net layer

Storage — Physical Layer

- 4K random read/write against the DRBD device (/dev/drbd1133) reaches only ~22K read IOPS and ~9K write IOPS.
- Against the Micron 7300 MAX's rated 220K read / 60K write IOPS (same 4 KiB block, QD512) that's only ~10% of rated reads and ~15% of rated writes.
- The extra CPU load visible alongside these runs is the DRBD device pushing replication traffic across the same congested network path.

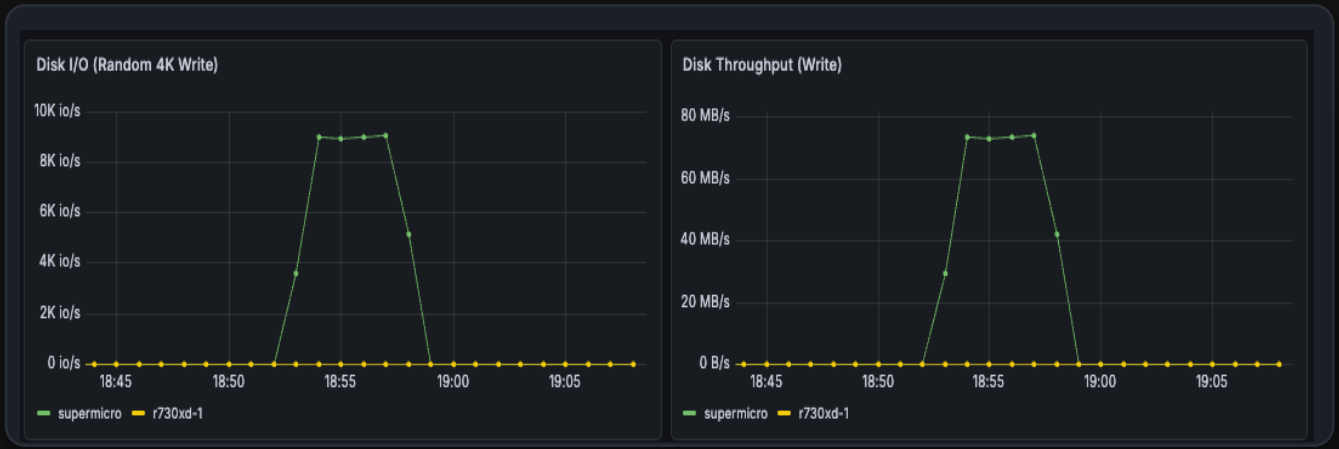


Figure 9.5 — Disk I/O and Write Speed



Figure 9.6 — Disk I/O and Read Speed

~22K / ~9K
read / write IOPS

~10% / ~15%
of rated read / write IOPS

220K / 60K
Micron 7300 MAX rated IOPS

Storage — Container Layer

- The same 4K random workload, now issued against the container-facing block device (/dev/sdb) from inside thesis-1.
- Results land around ~19K read IOPS and ~14K write IOPS — roughly 8% of rated reads and 23% of rated writes.
- Same block size, same queue depth as the physical test — the only difference is the virtualization abstraction sitting between the host and the container's block device.



Figure 9.7 — Disk I/O and Read/Write Speed (Container Layer)

~19K / ~14K
read / write IOPS

~8% / ~23%
of rated read / write IOPS

1 more layer
virtualization buffer vs. physical

VARIATION 1 · BASELINE · SUMMARY

What the Baseline Tells Us

The expected shape of an unaccelerated result — and the reference point everything else is measured against

- A consistent, CPU-bound bottleneck pattern — network and storage both land far below the hardware's actual ceiling, and both get worse (not better) once containerized.
- The bottleneck is the same root cause everywhere: every packet and every replicated write is classified, copied, and processed by the host CPU instead of the NIC.
- Figures 9.4–9.7 are the evidence trail for that pattern:
  - *Figure 9.4 — container network throughput collapses to ~1.5 Gb/s, worse than the already-limited host figure.*
  - *Figures 9.5 & 9.6 — physical storage reaches only ~10–15% of the drive's rated IOPS on writes and reads.*
  - *Figure 9.7 — the container-layer storage result stays in the same single-digit-to-low-double-digit percentage range.*
- This is the number Variation 2 has to beat — and by how much is the real measure of what offloading and RDMA are worth.

Network — Physical Layer

- With ASAP² offload the network datapath is correctly relocated to the NIC.
- CPU utilization for network processing is essentially non-existent — idling under 6% on both nodes.
- Bandwidth reaches near line-rate: ~96 Gb/s average RX/TX, the clearest possible evidence the offload transport is forwarding correctly.

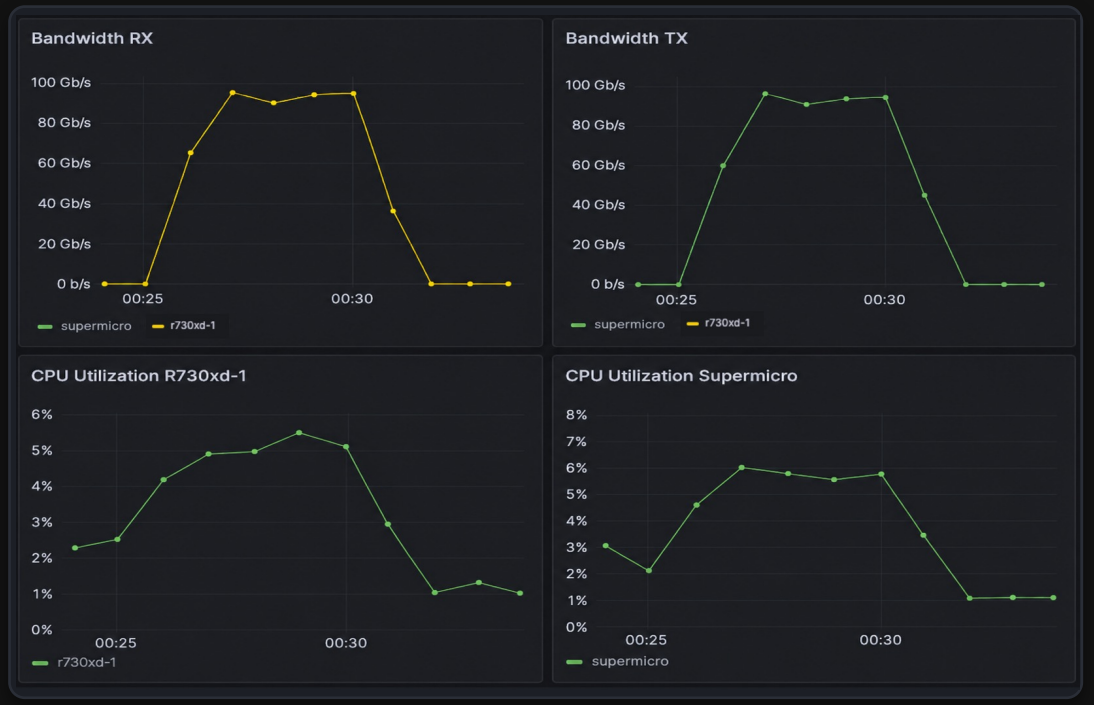


Figure 9.8 — Bandwidth Speed Test (Physical Layer)

VARIATION 2 · HW OFFLOAD · NETWORK

Network — Container Layer

- A similar offloading pattern holds inside the container environment — traffic moves through the OVS/ASAP²-accelerated data path but still traverses the host's software stack.
- Between thesis-1 and thesis-2, throughput reaches near line-rate utilization at ~96 Gb/s — the virtualization layer does see a significant increase in CPU load before the datapath is hardware-offloaded.
- This is the direct counterpoint to the baseline container result: the same two containers, the same test, an entirely different outcome once the fabric is set up correctly.

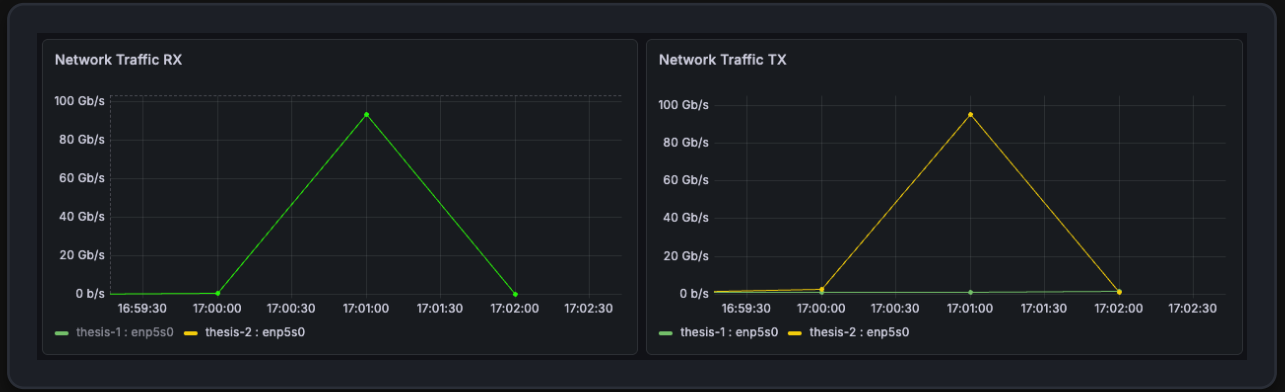


Figure 9.9 — Bandwidth Speed Test (Container Layer)

~96 Gb/s
thesis-1 ↔ thesis-2 throughput

1.5 → 96 Gb/s
baseline vs. offloaded container

Virtualization cost
High CPU cost because of VM traversal into kernel

Storage — Physical Layer

- 4K random read/write against the DRBD device (/dev/drbd1133) reaches only ~22K read IOPS and ~17K write IOPS.
- Against the Micron 7300 MAX's rated 220K read / 60K write IOPS (same 4 KiB block, QD512) that's only ~10% of rated reads and ~15% of rated writes.
- The extra CPU load visible alongside these runs is the DRBD device pushing replication traffic through the kernel before reaching the offloaded path.

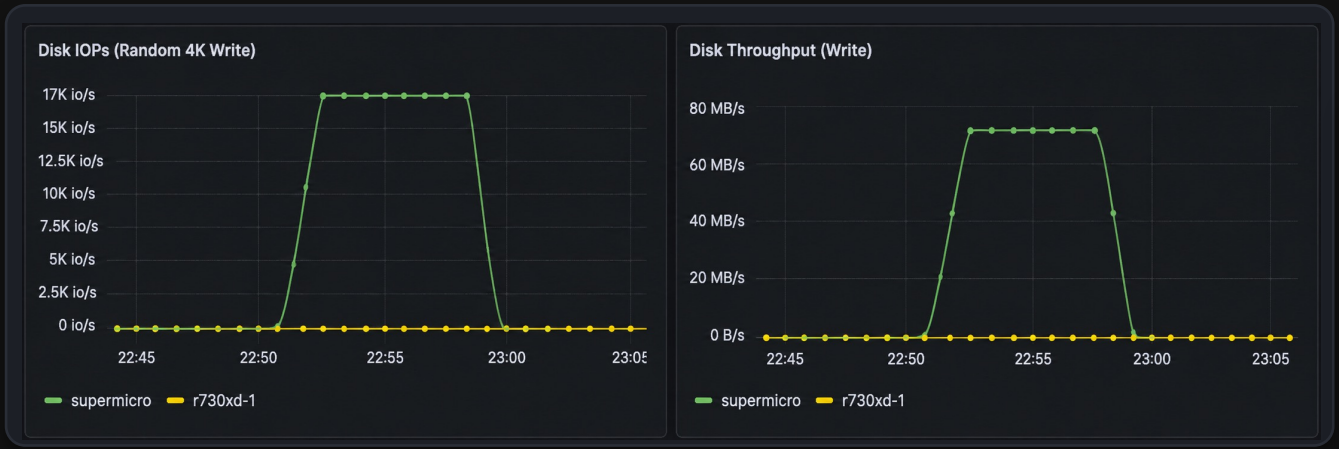


Figure 9.5 — Disk I/O and Write Speed



Figure 9.6 — Disk I/O and Read Speed

~22.5K / ~17K
read / write IOPS

~11% / ~28%
of rated read / write IOPS

220K / 60K
Micron 7300 MAX rated IOPS

Storage — Container Layer

- The same 4K random workload, now issued against the container-facing block device (/dev/sdb) from inside thesis-1.
- Results land around ~21.5K read IOPS and ~16K write IOPS — roughly 10% of rated reads and 35% of rated writes.
- Same block size, same queue depth as the physical test — the only difference is the virtualization abstraction sitting between the host and the container's block device.



Figure 9.7 — Disk I/O and Read/Write Speed (Container Layer)

~22K / ~16K
read / write IOPS

~10% / ~26%
of rated read / write IOPS

1 more layer
virtualization buffer vs. physical

VARIATION 2 · ANALYSIS

The Volume of Work Removed From the Host

Why the accelerated numbers are close to what the metal can actually do on the network but not the storage layer.

- Saturating the 100 GbE fabric at a 9000-byte MTU requires processing on the order of 1.4 million packets/sec/direction — 8.2 million at a standard 1500-byte MTU.
- Under the baseline, every one of those packets is classified, switched, and copied by the host CPU, and storage replication adds a second kernel copy on top — driving average CPU utilization to ~45% across all cores.
- Hardware offloading relocates that per-packet classification and switching into the NIC's embedded switch. It does not remove the remaining replication overhead which causes a significant CPU load to acumulate.

Baseline

8–9 Gb/s at ~45% host CPU

Nearly half the machine spent moving data instead of running workloads.

Fully Offloaded

~96 Gb/s with CPU near-idle for network and ~28% for storage

Close to an order of magnitude more data, with the CPU capacity handed back to tenants for network only transactions.

Network — Physical Layer

- With ASAP² offload and RDMA both active, the network datapath is correctly relocated to the NIC.
- CPU utilization for network processing is essentially non-existent — idling under 4% on both nodes.
- Bandwidth reaches near line-rate: ~96 Gb/s average RX/TX, the clearest possible evidence the offload and RDMA transport are both working.

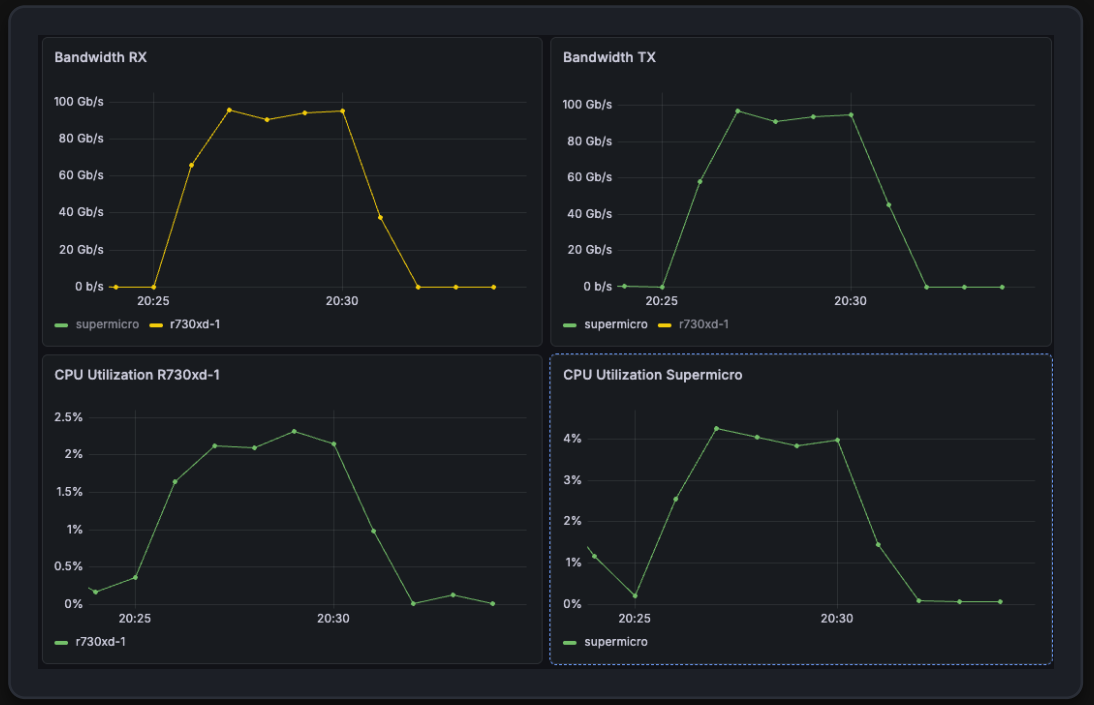


Figure 9.8 — Bandwidth Speed Test (Physical Layer)

VARIATION 3 · HW OFFLOAD + RDMA · NETWORK

Network — Container Layer

- The same offloading pattern holds inside the container environment — traffic moves through the OVS/ASAP²-accelerated data path rather than the host's software stack.
- Between thesis-1 and thesis-2, throughput again reaches near line-rate utilization at ~96 Gb/s — the virtualization layer costs nothing once the datapath is hardware-offloaded.
- This is the direct counterpoint to the baseline container result: the same two containers, the same test, an entirely different outcome once the fabric is set up correctly.

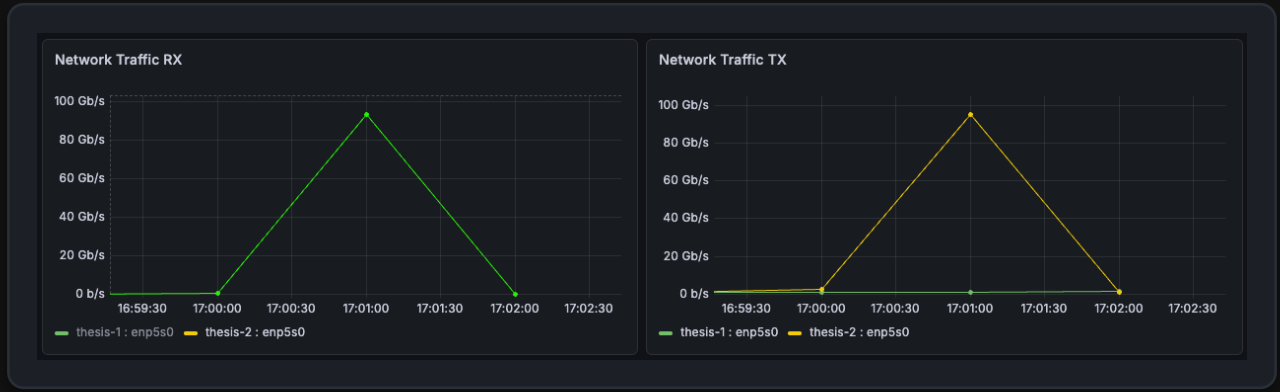


Figure 9.9 — Bandwidth Speed Test (Container Layer)

~96 Gb/s
thesis-1 ↔ thesis-2 throughput

1.5 → 96 Gb/s
baseline vs. offloaded container

Virtualization cost
effectively zero once offloaded

Storage — Physical Layer

- The DRBD block device now hovers around 210K read IOPS and ~59K write IOPS — about 95% and 98% of the Micron 7300 MAX's rated 220K/60K.
- Disk throughput reaches the drive's rated line speeds directly: ~2.4 GB/s reads, ~650 MB/s writes.
- The cluster is now taking full advantage of the available hardware offloads and the RDMA transport, rather than leaving the NVMe media mostly idle.



Figure 9.10 — Disk I/O and Read/Write Speed (Physical Layer)

VARIATION 3 · HW OFFLOAD + RDMA · STORAGE

Storage — Container Layer

- The same 4K random workload from inside thesis-1, against /dev/sdb, holds at the same ~210K read / ~59K write IOPS — ~95% and ~98% of rated.
- Disk throughput again reaches ~2.4 GB/s reads and ~650 MB/s writes — matching the physical-layer result almost exactly.
- The container abstraction that cost 8–23% of rated IOPS in the baseline now costs nothing measurable — the offloaded path carries straight through the virtualization layer.



Figure 9.11 — Disk I/O and Read/Write Speed (Container Layer)

~210K / ~59K
read / write IOPS

~95% / ~98%
of rated read / write IOPS

2.4 GB/s / 650 MB/s
read / write throughput

VARIATION 3 · ANALYSIS

The Volume of Work Removed From the Host

Why the accelerated numbers are close to what the metal can actually do

- Saturating the 100 GbE fabric at a 9000-byte MTU requires processing on the order of 1.4 million packets/sec/direction — 8.2 million at a standard 1500-byte MTU.
- Under the baseline, every one of those packets is classified, switched, and copied by the host CPU, and storage replication adds a second kernel copy on top — driving average CPU utilization to ~45% across all cores.
- Hardware offloading relocates that per-packet classification and switching into the NIC's embedded switch. RDMA removes the remaining replication overhead by writing directly into remote memory — no kernel stack, no interrupts, no extra copy.

Baseline

8–9 Gb/s at ~45% host CPU

Nearly half the machine spent moving data instead of running workloads.

Fully Offloaded + RDMA

~96 Gb/s at near-idle CPU

Close to an order of magnitude more data, with the CPU capacity handed back to tenants.

RDMA VS. TCP

Where RDMA & Offloading Win

On the high-bandwidth data path, the accelerated transport is decisively better

8–9 → 96 Gb/s
network throughput

~10% → ~95%
of rated read IOPS

~15% → ~98%
of rated write IOPS

- Beyond the raw throughput numbers, kernel bypass removes the system-call, data-copy, and interrupt costs the software path imposes on every transfer — lowering latency in addition to raising throughput.
- For storage replication specifically — large, sustained transfers that are sensitive to both bandwidth and host load — these are exactly the characteristics the workload demands.

RDMA VS. TCP

Where TCP Remains Preferable

The advantages of RDMA come at a cost that isn't justified everywhere

Configuration complexity

TCP runs correctly on any Ethernet network with no special setup. RoCEv2 required PFC, DSCP-to-priority mapping, DCB, trust mode, and QoS binding on every Scalable Function, jumbo frames end-to-end, and a PFC storm watchdog — a surface that must be correct on the switch, host, and function simultaneously.

Fragility of the lossless fabric

PFC guarantees losslessness but risks a PFC storm — back-pressure from one stalled device propagating hop-by-hop and stalling an entire traffic class. TCP was built to tolerate loss and recovers gracefully. For traffic that isn't bandwidth-critical, that robustness is worth more than the throughput RDMA adds.

Small, latency-tolerant traffic gains little

RDMA and jumbo frames pay off on large, sustained transfers. Management/control-plane traffic — LINSTOR, OVN, Incus coordination — is small messages where latency matters more than throughput. This is exactly why the cluster restricts RDMA to the DRBD data layer alone.

Hardware dependence & observability cost

RDMA needs SmartNIC-class adapters, a capable switch, and switchdev/SF driver support; TCP runs on commodity hardware. RDMA is also harder to see — it bypasses the kernel stack, so a dedicated RDMA exporter was required just to measure it, an ongoing maintenance cost TCP doesn't carry.

RDMA VS. TCP · SUMMARY

A Deliberate Division of Traffic

Not one transport for the whole cluster — the right transport for each job

RDMA + Hardware Offload

Storage replication & the high-bandwidth data path

Where throughput, storage, and CPU-efficiency gains are decisive — and the workload justifies the configuration and fragility costs.

Conventional TCP

Management & control-plane traffic

Where simplicity, robustness to loss, and freedom from specialized hardware and instrumentation make it the more appropriate transport.