# QProxy

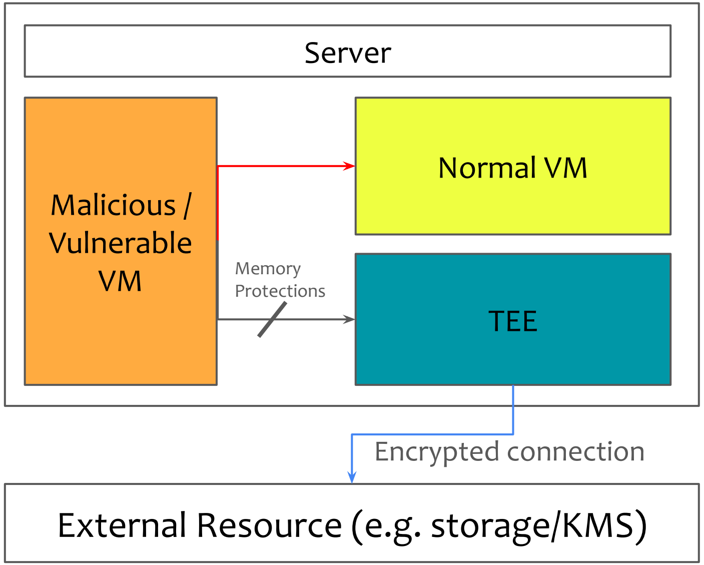
## Towards running legacy applications inside AWS Nitro Type Enclaves

Jalil David Salamé Messina
Advisor: Dr. Quoc Do Le
Chair of Computer Systems
https://dse.in.tum.de/

15.01.2024 – 15.04.2024

# Motivation: Research context

- Confidential applications need isolation
- Shared cloud environments don’t ensure it
- Trusted Execution Environments (TEEs) do
  - Application Level
    - Intel SGX
  - Virtual Machine (VM) Level:
    - Intel TDX
    - AMD SEV-SNP
    - AWS Nitro enclaves
    - Huawei Qingtian enclaves


Server
Malicious / Vulnerable VM
Normal VM
Memory Protections
TEE
Encrypted connection
External Resource (e.g. storage/KMS)

- AWS Nitro and Huawei Qingtian enclaves only allow VSock connections
- We need a TCP ↔ VSock translation layer

# State-of-the-art

- Nitriding Proxy[1]
- Enclaver (https://github.com/edgebitio/enclaver)
- Fortanix (Confidential Computing Manager) (https://support.fortanix.com/hc/en-us/categories/360003107511-Confidential-Computing-Manager)

[1] Winter, Philipp, et al. "A Framework for Building Secure, Scalable, Networked Enclaves." arXiv preprint arXiv:2206.04123 (2022).

# Research gap

- Limitations of State-of-the-art:
  - Focused only on AWS enclaves
  - Integrates with AWS KMS only
  - Automatic TLS Termination
  - Poor Performance
- Our work:
  - Generic Proxy
  - No direct integrations
  - No TLS termination
  - Acceptable performance

# Problem statement

- In this Thesis, our goal is to develop and implement a proxy system that:
  - Exposes enclave services outside the enclave
  - Supports both AWS Nitro and Huawei Qingtian enclaves
  - Without modifying the confidential application
  - With minimal overhead
  - Scale to high loads

# QProxy: A TCP to VSock proxy for enclaves

QProxy

**QProxy design goals:**

- Security
- Transparency
- Performance
- Scalability

# Outline

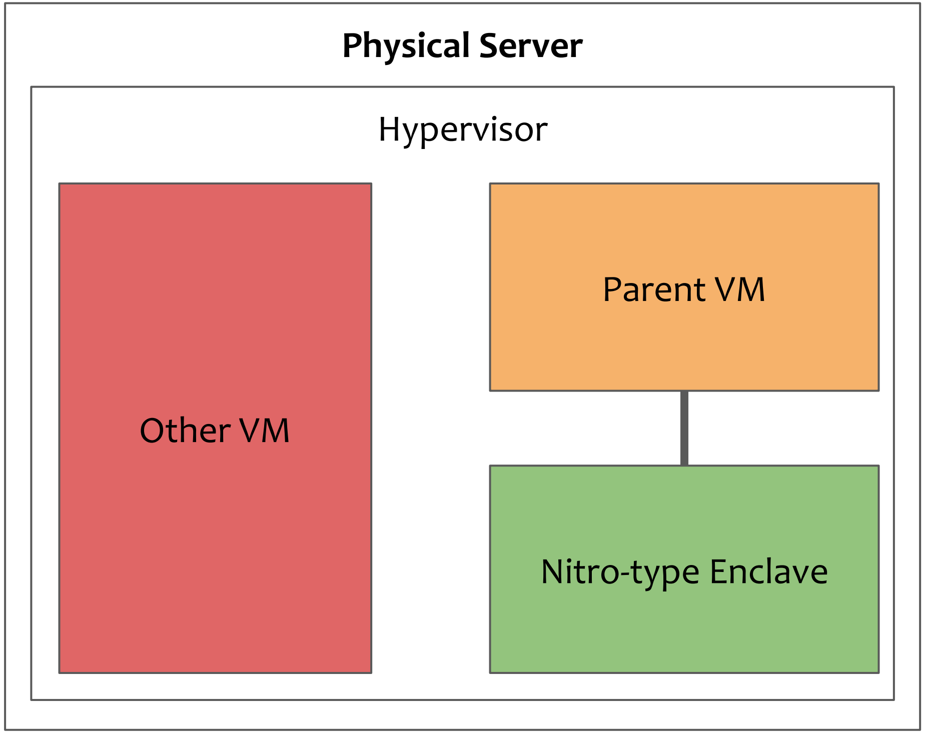
- ~~Motivation~~
- Background
  - Trusted Execution Environments (TEEs)
  - Nitro-type Enclaves
- Design
- Implementation
- Evaluation

# Trusted Execution Environments (TEEs)

- AMD SEV-SNP/Intel TDX
  - VM Level
  - Confidential (through memory encryption)
  - Integrity (through memory protections)
  - Remote Attestation
- Intel SGX
  - Application Level
- ARM TrustZone
  - VM Level (boot time)
- ARM CCA
  - VM Level (runtime)

# Nitro-type Enclaves (AWS Nitro and Huawei Qingtian)

- VM as a TEE
- Isolated (besides VSock)
  - Mitigates Spectre & Meltdown
- Signed Attestation Document



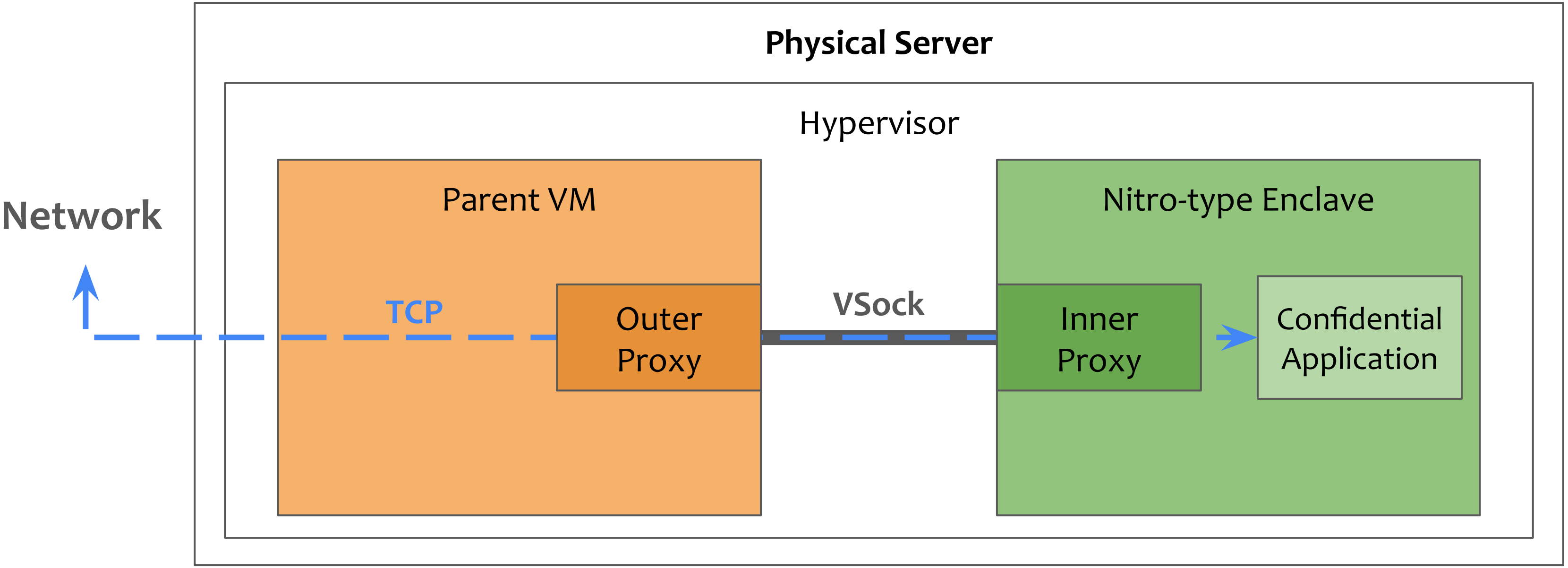
# Outline

- ~~Motivation~~
- ~~Background~~
- Design
  - System design
- Implementation
- Evaluation

# System overview

- Outside (host) Proxy
  - Network to and from Enclave
  - Rudimentary access control
- Inside (enclave) Proxy
  - Enclave to and from applications(s)

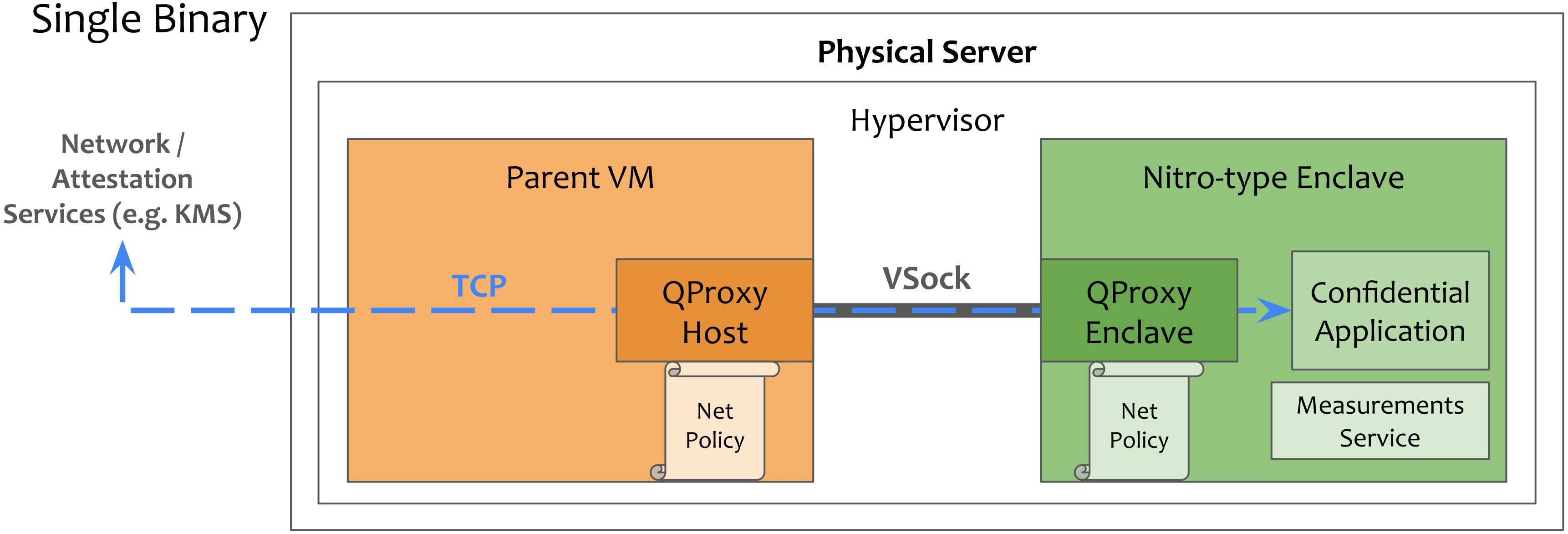
# Design overview



QProxy transparently establishes connections to the confidential application

# Design details

- Layer 4 Proxy (Over TCP)
- No Tunnelling
- Rust with Tokio.rs
- Single Binary



# Outline

- ~~Motivation~~
- ~~Background~~
- ~~Design~~
- Implementation
- Evaluation

# Implementation

QProxy is built with Rust[1] and Tokio.rs[2]

- Memory-safety at compile-time
- No unsafe Rust
- Leveraging well-tested OSS libraries
- Highly concurrent through Tokio.rs
- Currently running in production

[1] Rust Programming Language: https://www.rust-lang.org/

[2] Tokio an asynchronous runtime for Rust: https://tokio.rs/

# Outline

- ~~Motivation~~
- ~~Background~~
- ~~Design~~
- ~~Implementation~~
- Evaluation

# Evaluation

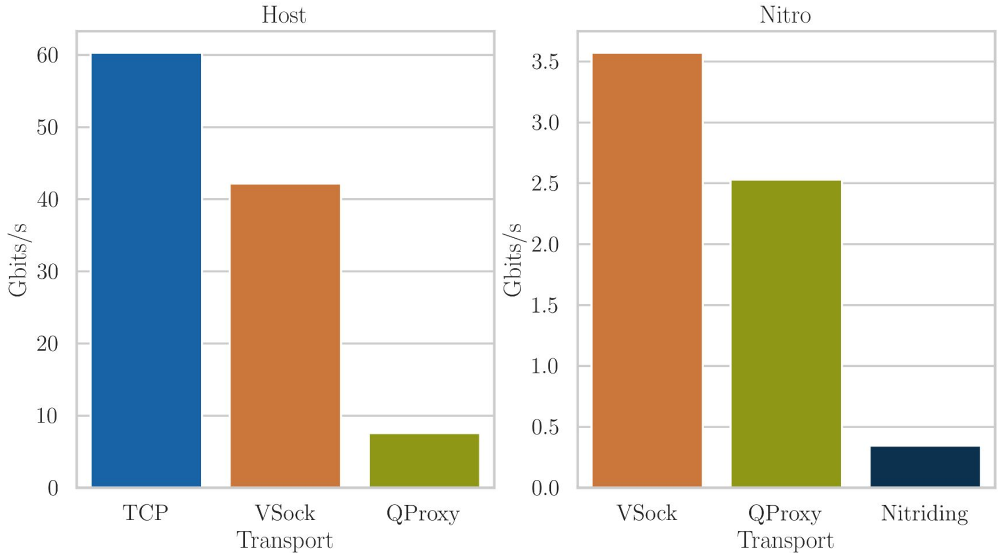
- What is the performance overhead of QProxy?
  - Latency
  - Throughput
- Is it transparent?
  - Run Redis through QProxy
- How does it compare with the state-of-the-art?
  - Compare against Nitriding
- How does it scale with the available resources?
  - Increasing the available resources

# Evaluation

- Experimental setup (Nitro):
    - c6.8xlarge AWS Nitro EC2
    - Intel® Xeon® Platinum 8375C CPU @ 2.90GHz
    - 32 vCPUs (2 allocated to the enclave)
    - 64 GiB DRAM (1.5 GiB allocated to the enclave)
- Experimental setup (Qingtian):
    - c7.8xlarge.4 Huawei EC2
    - Intel® Xeon® Platinum 8378A CPU @ 3.00GHz
    - 32 vCPUs (2 allocated to the enclave)
    - 128 GiB DRAM (1 GiB allocated to the enclave)
- Baseline:
    - Using a VSock service (no proxy overhead)
    - Using Nitriding (state-of-the-art) (only on Nitro)

# Performance overhead (throughput)

**iperf3-vsock:** single-core, throughput stress-test (5 minute average)

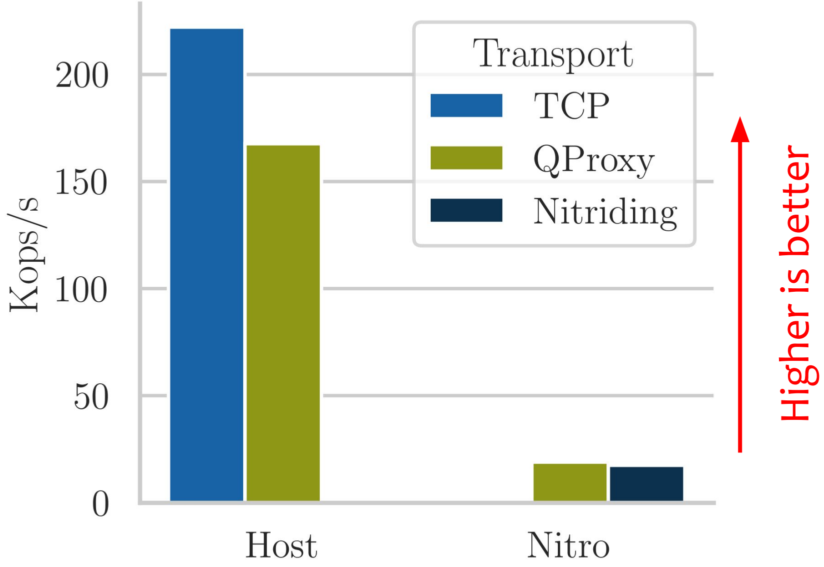


Higher is better

QProxy is between 2.5 and 5 times better than Nitriding

# Performance overhead (throughput)

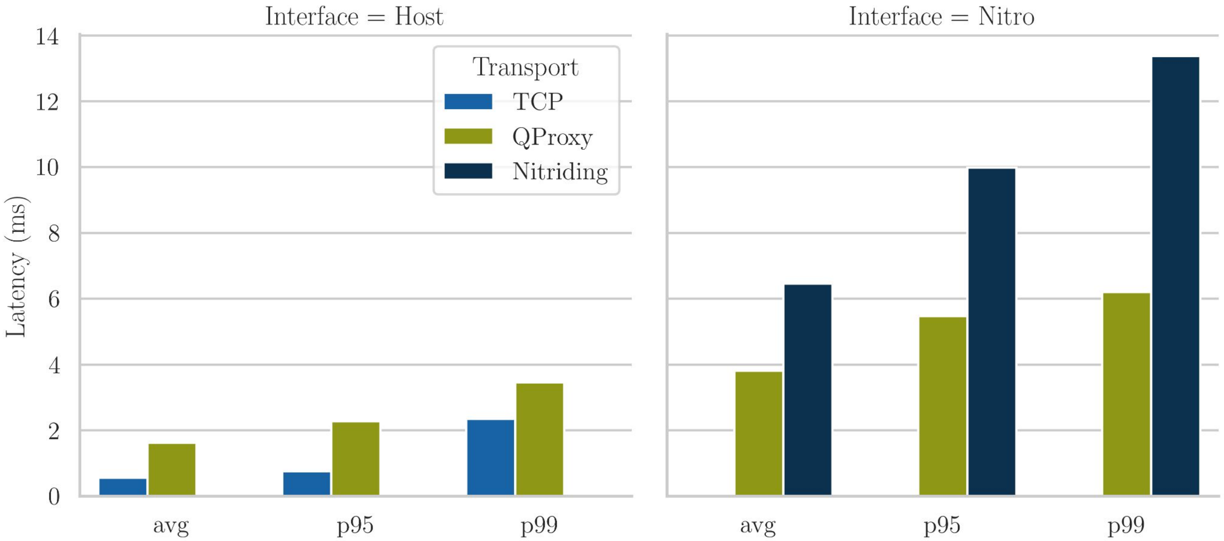
**memtier_benchamrk:** redis load-test (average over 5 minutes)



QProxy achieves slightly better throughput than Nitriding

# Performance overhead (latency)

**memtier_benchamark:** redis load-test, rate-limited to 10kops/s (5 minutes)

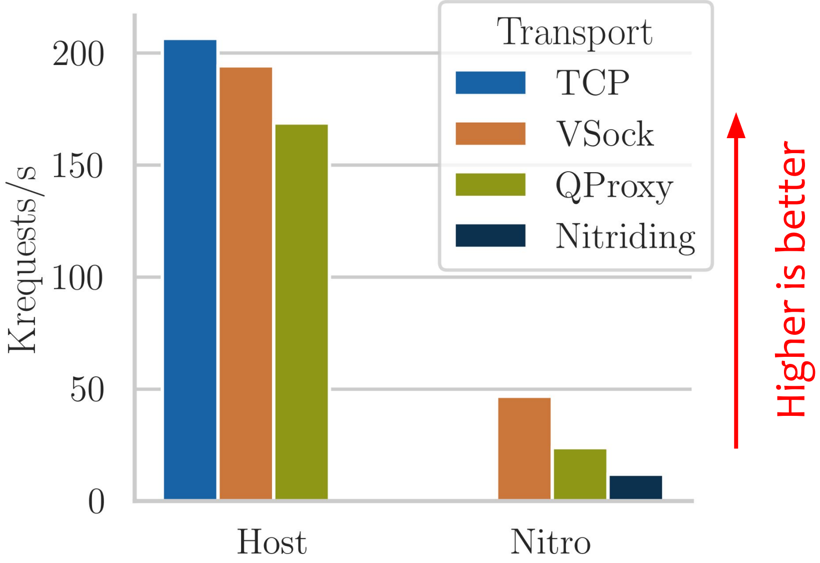


Lower is better

QProxy achieves circa 1.5 times better latency than Nitriding

# Performance overhead (throughput)

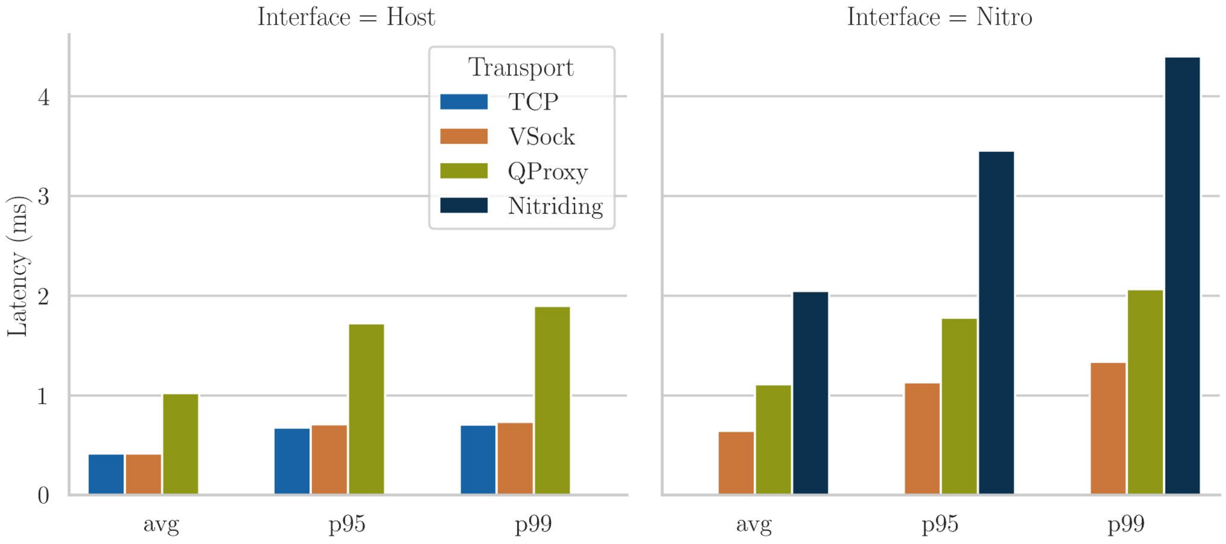
**oha:** HTTP load-test, 50 connections over 30 threads (5 minute average)



QProxy achieves circa 2 times the throughput of Nitriding

# Performance overhead (latency)

**oha:** HTTP load-test, 50 connections over 30 threads, limited to 10krps (5 minutes)

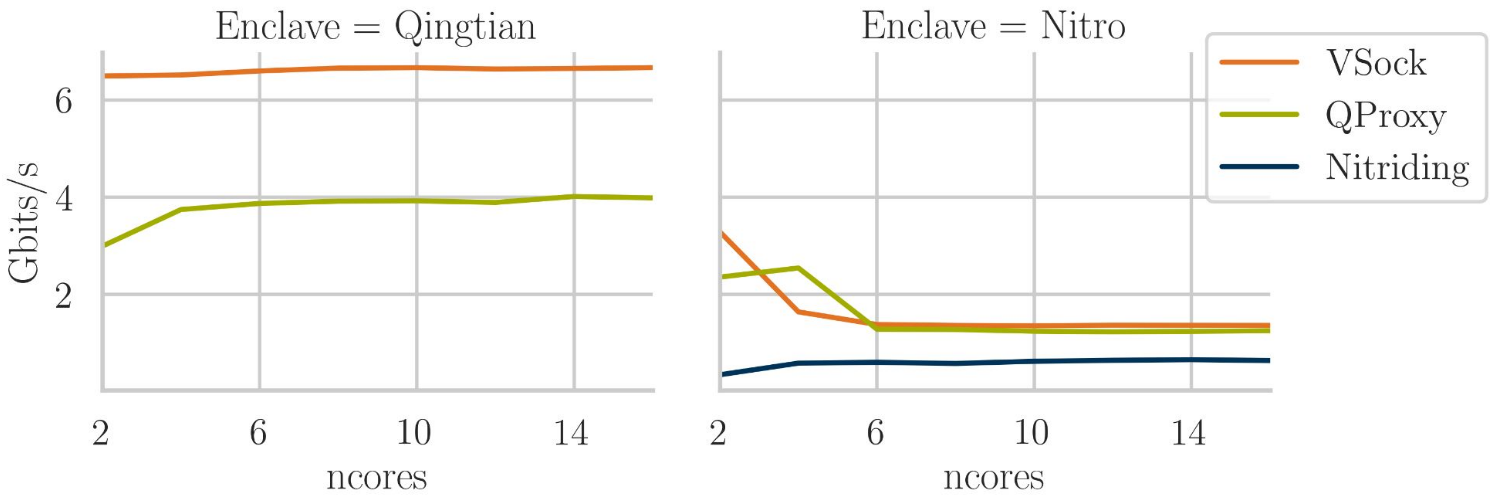


Lower is better

QProxy achieves circa 2 times better latency than Nitriding

# Scaling (throughput)

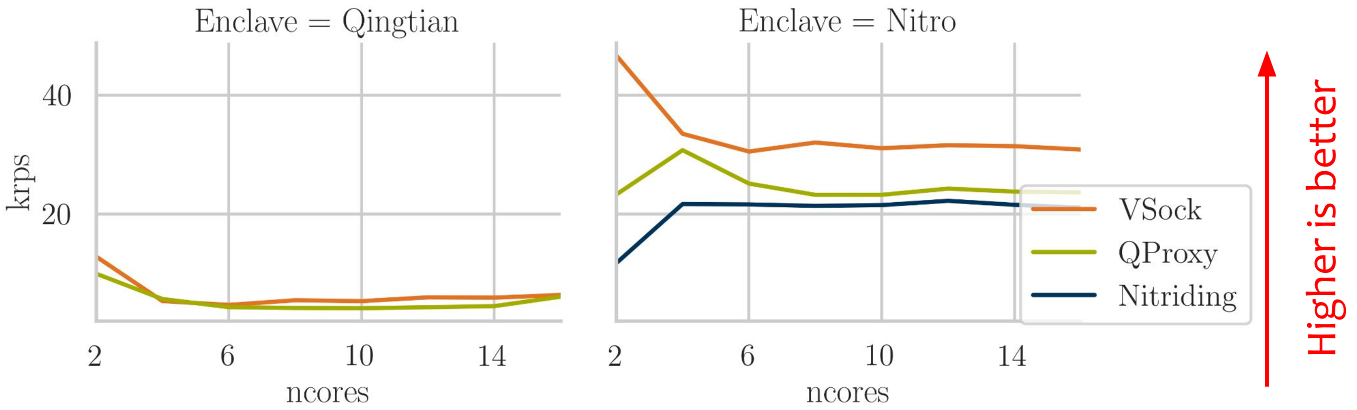
**iperf3-vsock:** single-core, throughput stress-test (5 minute average)



AWS Nitro Enclaves have decreased baseline performance with more cores

# Scaling (throughput)

**oha:** HTTP load-test, 50 connections over 30 threads (5 minute average)



This problem is also present in Qingtian enclaves

# Summary

**Current applications are not designed for Nitro-type enclaves**

- VSock transport
- confidentiality & integrity
- remote attestation

**QProxy:**

- transparent VSock proxy
- no source code modifications
- Circa 2x performance of Nitriding

**Future Work:**

- Integrate with configfs-tsm (kernel remote attestation ABI)
- Integrate with KMS services

# Questions?