# Using DPUs and attested TLS to build a trusted network

**Guilhem Bryant,** Veracruz Development Team, Arm
May 2024

# Introduction

- Great strides in the last few years towards making confidential computing more practical and useable
- But confidential computing is still (mostly) restricted to the CPU hence general-purpose computations...
- How could we do confidential ML training in the cloud?
  - ML company outsourcing training to the cloud must trust the whole system (CPUs, AI accelerators, network interfaces, storage devices etc.) processing their data
  - Additional complications: multi-tenancy and distrust between cloud provider and tenants
- => Let's make confidential computing even more useful by connecting CPU enclaves to other devices

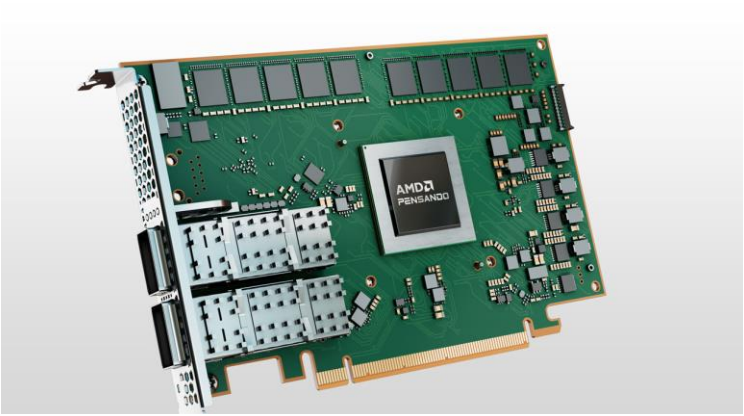
# What is a DPU?

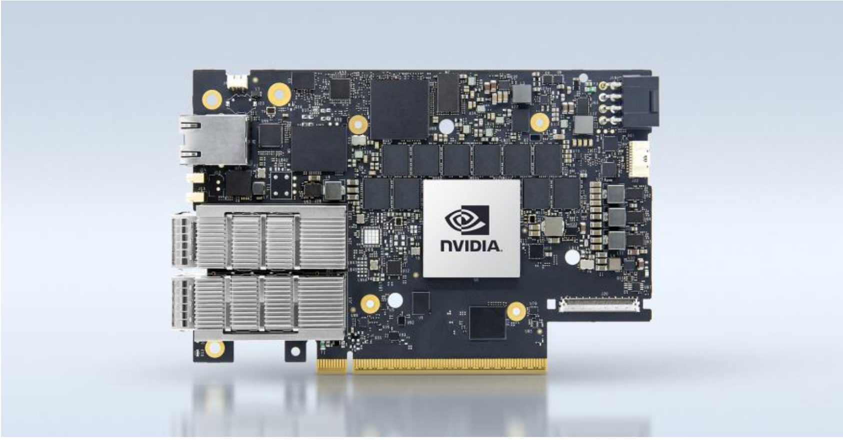
- DPUs specialize in infrastructure functions:
  - High-performance networking: TCP, packet processing acceleration
  - Security: cryptography, RNG, HW root of trust
  - Storage: NVMe over network (typically RDMA as transport layer), decompression
- Widely used by hyperscalers & cloud providers (AWS, Azure, Alibaba, IBM, Oracle)
- Some figures for Nvidia’s Bluefield 2 DPU [1]
  - ~ 24% energy savings in telecoms datacenters compared to traditional CPU-centric servers
  - Similar figures for IPsec offload in the datacenter
  - 54x throughput increase when offloading Red Hat OpenShift's control and data planes to DPUs

- [1] https://blogs.nvidia.com/blog/bluefield-dpus-energy-efficiency/

# What do DPUs look like?

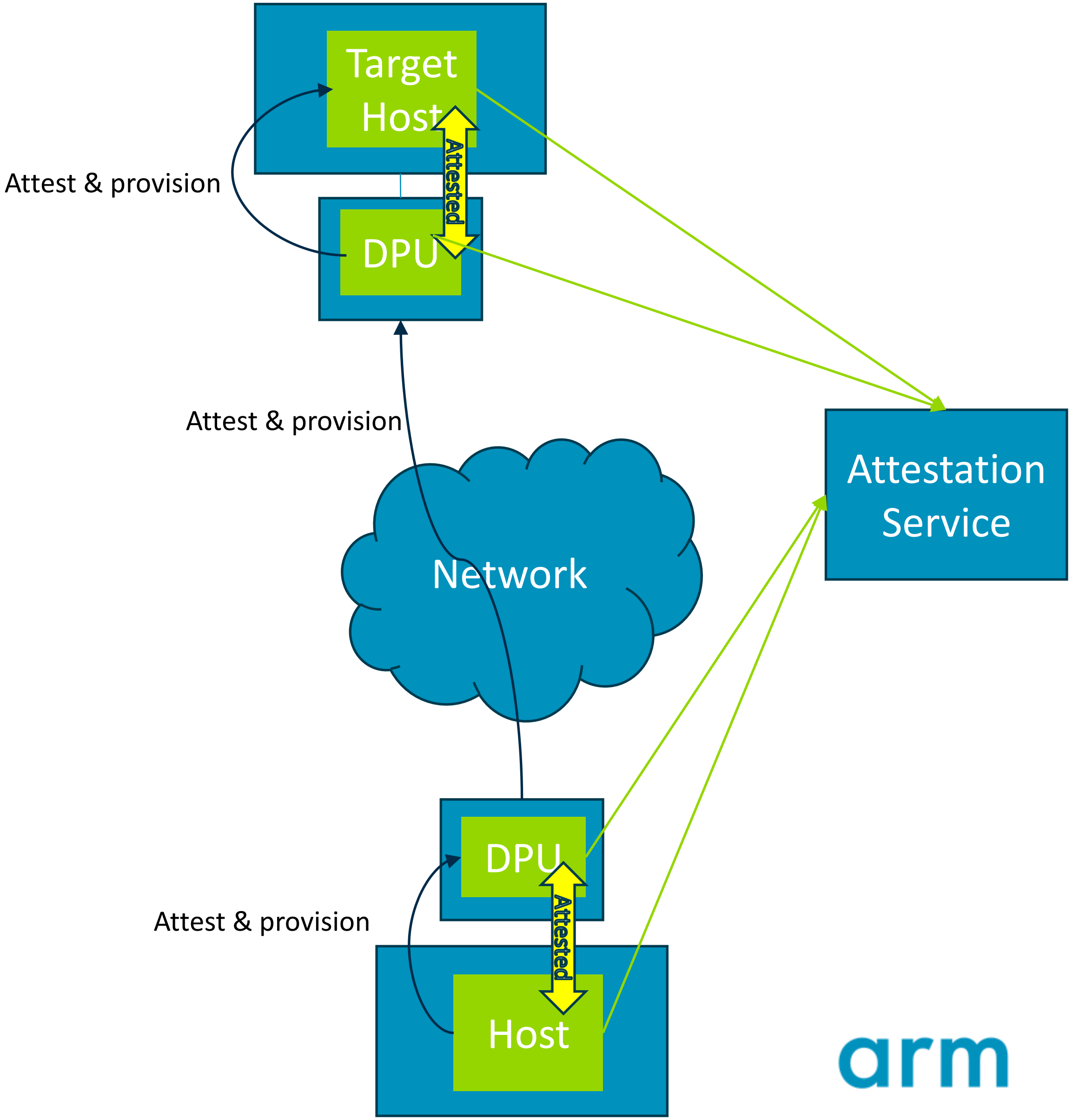
- Tradeoff between performance and programmability
  - Great variety of DPUs (CPUs, FPGAs, ASICs) with even greater variety of software stacks...
  - Linux Foundation's OPI tries to reduce this fragmentation
- Typically consists in a SoC with:
  - CPU cores for general-purpose tasks and management of the overall system
  - High-performance Ethernet/InfiniBand NICs (400Gb/s on latest models)
  - Hardware accelerators for crypto, packet processing, storage, etc.
  - PCIe switch to interconnect NICs, CPU subsystem and other devices (e.g. host CPU, storage, GPU)



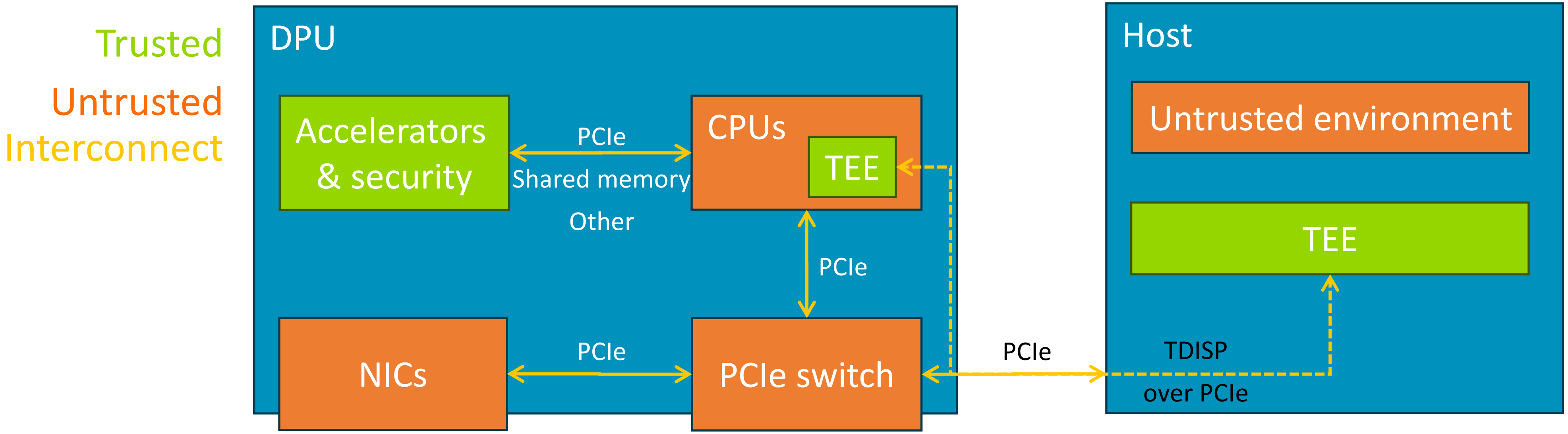



# Confidential computing and DPUs: What can we do?

- Vicarious attestation:
  - Use enclaves and transitivity of trust to allow nodes to attest each other and indirectly trust other nodes already attested
  - Create a trusted network of nodes (VPN or Virtual Private Cloud), each node hiding behind a DPU attesting other nodes on its behalf
- Confidential offload:
  - Host offloads TLS (handshake and record protection, maybe even bits of TCP) to DPU enclave after attesting it
  - Plaintext available in the enclave for processing (threat detection, data leak mitigation, policy enforcement)



# Confidential computing and DPUs: What threat model?




- TDISP (TEE Device Interface Security Protocol)
  - Establishes a TLS-like secure channel between a confidential VM and a VF on a supported PCIe device
- Caveats:
  - Devices supporting TDISP are not widely available yet
  - DPUs are not shipped with CPUs supporting enclaves yet
  - DPU architecture can vary widely across DPUs and so does the threat model

# Ongoing work

- Developed a framework to explore the two use cases
  - https://github.com/veracruz-project/dpu/tree/tls
  - Written (mostly) in Rust
  - 🥁🥁🥁Uses attested TLS to establish trust between nodes then for control & provisioning

# Why are we using attested TLS?

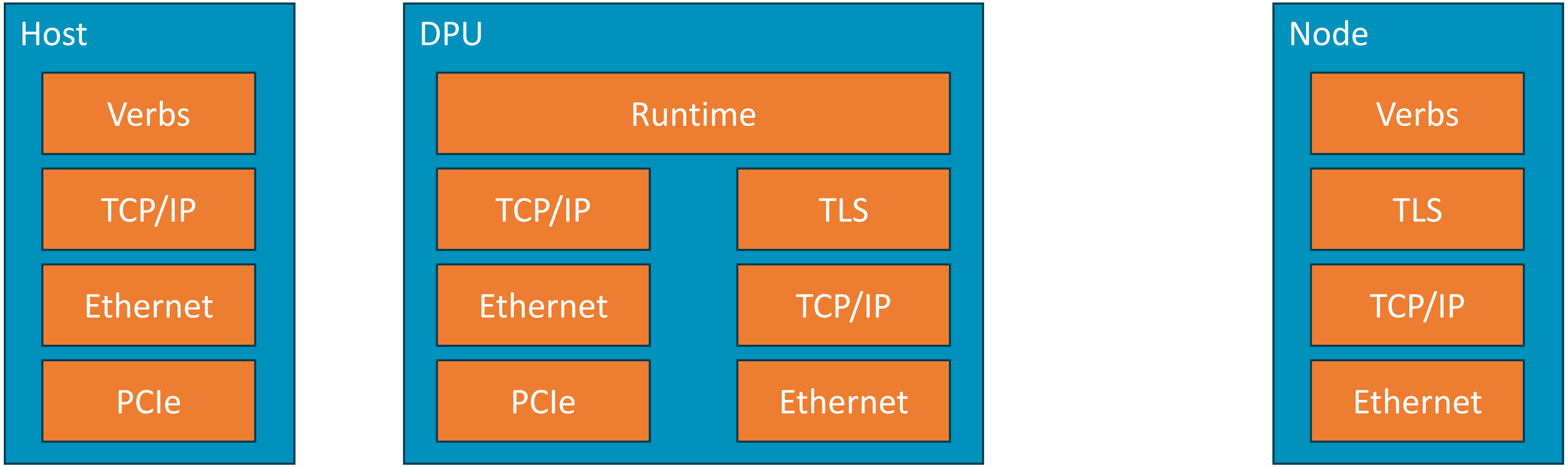
- “Attestation + TLS” vs TLS
  - “Attestation + TLS” provides stronger guarantees than TLS
- Attested TLS vs “Attestation + TLS”
  - Combining attestation and TLS into one standard protocol makes it less prone to failures (e.g. TLS with no attestation)
  - => all-or-nothing outcome: connection is either (attested AND confidential) OR (insecure)
- More scalable than certificate-based solutions?

# Software architecture

- Main dependency https://github.com/gbryant-arm/rust-mbedtls/tree/attested-tls
  - Rust wrapper around Mbed TLS
  - Combines into one single branch:
    - Attester code (https://github.com/ionut-arm/mbedtls/tree/parsec-attestation)
    - Relying party code (https://github.com/paulhowardarm/mbedtls/tree/ph-tls-attestation)
- TLS client and server code rewritten in Rust
  - https://github.com/veracruz-project/dpu/blob/tls/common/transport/src/tls_client.rs
  - https://github.com/veracruz-project/dpu/blob/tls/common/transport/src/tls_server.rs
  - Mind the unsafe block! …
- Dockerfile: DPU-specific things + attested TLS PoC’s Dockerfiles (includes vTPM implementation, Parsec, etc.)

# Network stack

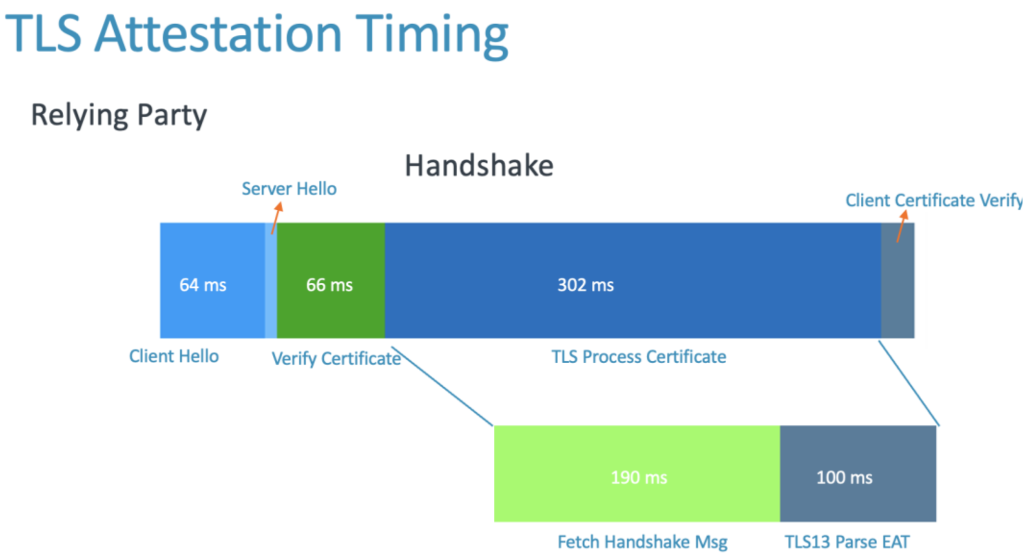
- Initiator/responder design




- Caveat: The prototype assumes the attester is the initiator/client… but in our case the attester is the responder/server (DPU) and the relying party is the initiator/client (host) => the attester piggybacks TLS on the TCP connection initiated by the relying party

# Early results (take a pinch/bucket of salt)

- Experiment:
  - Attester and attestation service running on DPU (Nvidia Bluefield 2 with Arm Cortex A72 cores)
  - Relying party running on host CPU (Arm N1)
  - Measuring handshake latency





- We need a proper baseline:
  - Traditional TLS?
  - Traditional TLS with attestation evidence added to certificate?

# Future work

- Benchmarking in progress...
- Improve performance
  - Better utilize Bluefield's acceleration capabilities
  - Improve network performance with RDMA and/or DPDK
- Explore other use cases:
  - DPU-orchestrated VM provisioning ala Nitro
  - Virtual Private Cloud & remote attestation at scale
  - Confidential middleboxes

Thank You
Danke
Gracias
谢谢
ありがとう
Asante
Merci
감사합니다
धन्यवाद
Kiitos
شكرًا
ধন্যবাদ
תודה