TNIC: A Trusted NIC Architecture

A hardware-network substrate for building high-performance trustworthy distributed systems

Dimitra Giantsidi
The University of Edinburgh
Edinburgh, UK

Julian Pritzi
Technical University of Munich
Munich, Germany

Felix Gust
Technical University of Munich
Munich, Germany

Antonios Katsarakis*
Huawei Research
Edinburgh, UK

Atsushi Koshiba
Technical University of Munich
Munich, Germany

Pramod Bhatotia
Technical University of Munich
Munich, Germany

Abstract

We introduce TNIC, a trusted NIC architecture for building trustworthy distributed systems deployed in heterogeneous, untrusted (Byzantine) cloud environments. TNIC builds a minimal, formally verified, silicon root-of-trust at the network interface level. We strive for three primary design goals: (1) a host CPU-agnostic unified security architecture by providing trustworthy network-level isolation; (2) a minimalistic and verifiable TCB based on a silicon root-of-trust by providing two core properties of transferable authentication and non-equivocation; and (3) a hardware-accelerated trustworthy network stack leveraging SmartNICs. Based on the TNIC architecture and associated network stack, we present a generic set of programming APIs and a recipe for building high-performance, trustworthy, distributed systems for Byzantine settings. We formally verify the safety and security properties of our TNIC while demonstrating its use by building four trustworthy distributed systems. Our evaluation of TNIC shows up to 6× performance improvement compared to CPU-centric TEE systems.

CCS Concepts: • **Security and privacy → Trusted computing**.

Keywords: trusted computing, hardware-software co-design

ACM Reference Format:
Dimitra Giantsidi, Julian Pritzi, Felix Gust, Antonios Katsarakis*, Atsushi Koshiba, and Pramod Bhatotia. 2025. TNIC: A Trusted NIC Architecture. In *Proceedings of the 30th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2 (ASPLOS '25), March 30-April 3, 2025, Rotterdam, Netherlands.* ACM, New York, NY, USA, 20 pages. https://doi.org/10.1145/3676641.3716277

This work is licensed under a Creative Commons Attribution 4.0 International License.

ASPLOS '25, March 30-April 3, 2025, Rotterdam, Netherlands

© 2025 Copyright held by the owner/author(s).

ACM ISBN 979-8-4007-1079-7/2025/03.
https://doi.org/10.1145/3676641.3716277

1 Introduction

Distributed systems are integral to the third-party cloud infrastructure [4, 18, 29, 33]. While these systems manifest in diverse forms (e.g., storage systems [41, 47, 58, 62, 69, 73, 85], management services [60, 96], computing frameworks [8, 10, 19]) they all must be fast and remain correct upon failures.

Unfortunately, the widespread adoption of the cloud has drastically increased the surface area of attacks and faults [80, 91, 147] that are beyond the traditional fail-stop (or crash fault) model [75]. The modern (untrusted) third-party cloud infrastructure severely suffers from arbitrary (*Byzantine*) faults [109] that can range from malicious (network) attacks to configuration errors and bugs and are capable of irreversibly disrupting the correct execution of the system [64, 80, 91, 147].

A promising solution to build trustworthy distributed systems that can sustain Byzantine failures is based on the *silicon root of trust*—specifically, the Trusted Execution Environments (TEEs) [7, 26, 48, 71, 140]. While the TEEs offer a (single-node) isolated Trusted Computing Base (TCB), we have identified three core challenges (§ 3.3) that complicate their adoption for building trustworthy distributed systems spanning multiple nodes in Byzantine cloud environments.

First, TEEs in heterogeneous cloud environments introduce programmability and security challenges. A cloud environment offers diverse heterogeneous host-side CPUs with different TEEs (e.g., Intel SGX/TDX, AMD SEV-SNP, AWS Nitro Enclaves, Arm TrustZone/CCA, RISC-V Keystone). These heterogeneous host-side TEEs require different programming models and offer varying security properties. Therefore, they cannot (easily) provide a generic substrate for building trustworthy distributed systems. Our work overcomes this challenge by designing a *host CPU-agnostic silicon root of trust* at the network interface (NIC) level (§ 4). We provide a generic programming API (§ 6) and a *recipe* (§ 6.2) for building high-performance, trustworthy distributed systems (§ 7).

Secondly, TEEs with a large TCB are plagued with security vulnerabilities, rendering them non-verifiable. With hundreds of security bugs already uncovered [81], TEEs' large TCBs further increase their security vulnerabilities [107, 130], impeding a formal verification of their security. We overcome

*This work started when the author was at the University of Edinburgh.

this with a *minimalistic verifiable TCB* (§ 4.1). Our TCB resides at the NIC hardware and is equipped with *the lower bound of security primitives*; we provide only two key security properties of non-equivocation and transferable authentication for building trustworthy distributed systems (§ 2.1). Since we strive for a minimal trusted interface, we can (and we did) formally verify the security properties of our TCB (§ 4.4).

Thirdly, TEEs report significant performance bottlenecks. TEEs syscalls execution for (network) I/O is extremely costly [166], whereas even state-of-the-art network stacks showed a lower bound of 4× slowdown [55]. We attack this challenge based on two aspects. First, we build a scalable transformation with our minimal TCB's security properties (§ 6.2) to transform Byzantine faults ($3f$+1) to much cheaper crash faults ($2f$+1) for tolerating f (distributed) Byzantine nodes. Secondly, we design hardware-accelerated offload of the security computation at the NIC level by extending the scope of SmartNICs with *the lower bound of security primitives* (§ 4) while offering kernel-bypass networking (§ 5).

To overcome these challenges, we present TNIC, a trusted NIC architecture for building trustworthy distributed systems deployed in Byzantine cloud environments. TNIC realizes an abstraction of trustworthy network-level isolation by building a hardware-accelerated silicon root of trust at the NIC level. Overall, TNIC follows a layered design:

- **Trusted NIC hardware architecture (§ 4):** We materialize a minimalistic, verifiable, and host-CPU-agnostic TCB at the network interface level as the key component to design trusted distributed systems for Byzantine settings. Our TCB guarantees the security properties of non-equivocation and transferable authentication that suffice to implement an efficient transformation of systems for Byzantine settings. We build TNIC on top of FPGA-based SmartNICs [3]. We formally verify the safety and security guarantees of TNIC protocols using Tamarin Prover [125].
- **Network stack (§ 5) and library (§ 6):** Based on the TNIC architecture, we design a HW-accelerated network stack to access the hardware bypassing kernel for performance. On top of TNIC's network stack, we present a networking library that exposes a simplified programming model. We show *how to use* TNIC APIs to construct a generic transformation of a distributed system operating under the CFT model to target Byzantine settings.
- **Trusted distributed systems using TNIC (§ 7):** We build with TNIC the following (distributed) systems for Byzantine environments: Attested Append-only Memory [67], Byzantine Fault Tolerance [63], Chain Replication [161], and Peer-Review [90] —showing the generality of our approach.

We evaluate TNIC with a state-of-the-art software-based network stack, eRPC [101], on top of RDMA [126]/DPDK [24] with two different TEEs (Intel SGX [99] and AMD-sev [48]). Our evaluation shows that TNIC offers 3×—5× lower latency than the software-based approach with the CPU-based TEEs. For trusted distributed systems, TNIC improves throughput by up to 6× compared to their TEE-based implementations.

2 Motivation and Background

We first examine the design requirements for high-performance, trustworthy distributed systems for cloud environments.

2.1 Trustworthy Distributed Systems

Byzantine fault model. In the untrusted cloud infrastructure, arbitrary (Byzantine) faults are a frequent occurrence in the wild [80, 147, 171, 172]. To this end, system designers introduced Byzantine Fault Tolerant (BFT) systems that remain correct even under the presence of (a bounded number of) Byzantine failures [109]. Traditional BFT protocols need *at least* $3f+1$ nodes in order to provide consistent replication while tolerating up to f Byzantine failures. While BFT accurately captures the realistic security needs in the cloud [79], it is rarely adopted in practice [151] due to its complexity and limited performance [53, 149].

Crash fault model. The vast majority of cloud applications operate under the fail-stop (crash fault) model [14, 17, 21, 61, 69], optimistically *assuming* that the entire cloud infrastructure is trusted and only fails by crashing [75]. Compared to BFT replication, Crash Fault Tolerant (CFT) protocols [96, 108, 132, 133], require $2f+1$ replicas to tolerate f (yet non-Byzantine) failures. While CFT systems can offer performance and scalability [84], they are fundamentally incapable of ensuring safety in the presence of non-benign faults, hence, are ill-suited for the modern cloud.

Security properties for BFT. We seek to build BFT systems while reducing their programmability and performance overheads. Our approach, inspired by the theoretical findings of Clement et al. [68], *transforms* CFT systems into BFT systems by providing the *lower bound* of security properties, i.e., *transferable authentication* and *non-equivocation*.

We next explain the two security properties. First, *transferable authentication* allows a node to verify the original sender of a received message, even if it is forwarded by other than the original sender. Assuming that the sender p_i sends an authenticated message m to a recipient p_j, the authenticated message m is accompanied by an authentication token $\sigma(p_i)$ that allows p_j to verify that p_i generated the message, e.g., verify(m,$\sigma(p_i)$). Authentication tokens are unforgeable:

- if p_i is correct, then verify(m,$\sigma(p_i)$) is true if and only if p_i generated m.
- if p_i is faulty, verify(m,$\sigma(p_i)$) $\wedge$ verify(m',$\sigma(p_i)$) $\Rightarrow m=m'$. As such, a compromised p_i cannot produce two valid different messages that can be verified with the same token $\sigma(p_i)$.

As an authentication token is transferable, it allows another recipient p_k to evaluate verify(m,$\sigma(p_i)$) in the same way even when m and $\sigma(p_i)$ are forwarded from p_j.

Second, *non-equivocation* guarantees that a node cannot make conflicting statements to different nodes. Equivocation also manifests as network adversaries or replay attacks that send invalid messages or re-send valid but stale messages.

The seminal paper [68] proves that, given these two properties, a transformation from any CFT protocol to a BFT protocol is *always* possible without increasing the number of participating nodes; e.g., a reliable broadcast can be implemented to tolerate up to f Byzantine failures in an asynchronous system with $2f+1$ replicas, rather than the conventional $3f+1$.

2.2 High-Performance Distributed Systems

The aforementioned two security properties are sufficient to *correctly transform* (any) CFT distributed system to operate in the BFT model [68, 70]. However, a fundamental design trade-off exists between efficiency and robustness for practical deployments in the cloud. Our work aims to resolve this tension.

Trusted hardware for BFT. System designers established trusted hardware, TEEs, as the most effective way to eliminate a system's Byzantine counterparts [55, 57, 74, 162]. While TEEs can be used to offer BFT, prior research illustrated significant performance and architectural limitations in the context of networked systems [55, 57, 74, 162]. Based on performance and security studies [45, 46], TEEs' overheads in the heterogeneous cloud, in addition to their heterogeneity in programmability and security guarantees, are incapable of offering high-performant trusted networking under the BFT model.

SmartNICs for high-performance and BFT. We leverage the state-of-the-art hardware-level networking accelerators, i.e., SmartNICs [3, 9, 11, 28, 30–32, 40], to address the trade-off between performance and security, overcoming the limitations of TEEs. Our design choice of leveraging SmartNICs is not hypothetical; SmartNIC devices have already been launched by major cloud providers [9, 32, 40], presenting great opportunities for performance thanks to their integrated fully programmable hardware (e.g., ARM cores [11, 28, 31, 40], FPGAs [2, 3, 32]). Precisely, we rely on two promising directions: *(1)* security and network processing offloading at the NIC-level hardware and *(2)* an efficient transformation for BFT.

3 Overview

3.1 System Overview

We propose TNIC, a trusted NIC architecture for high-performance, trustworthy distributed systems, formally guaranteeing their secure and correct execution in the heterogeneous Byzantine cloud infrastructure. TNIC is comprised of three layers (shown in Figure 1): **(1) the TNIC hardware architecture** (green box) that implements trusted network operations on top of SmartNIC devices (§ 4), **(2) the TNIC network stack** (yellow box) that intermediates between the application layer and the TNIC hardware (§ 5), and **(3) the TNIC network library** (blue box) that exposes TNIC's programming APIs (§ 6).

Our TNIC hardware architecture implements the networking IB/RDMA protocol [1] on FPGA-based SmartNICs [3]. It extends the conventional protocol implementation with a minimal hardware module, the attestation kernel, that materializes the security properties of the non-equivocation and transferable authentication. The TNIC network stack configures the TNIC device on the control path while it offers the data path as kernel-bypass device access for low-latency operations. Lastly, the TNIC network library exposes programming APIs built on top of (reliable) one-sided RDMA primitives.

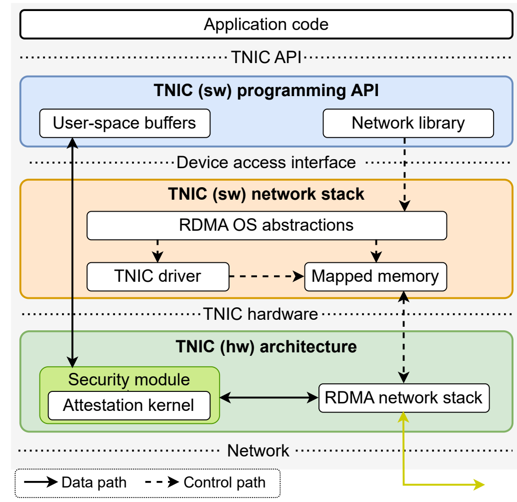

Figure 1. TNIC system overview.

3.2 Threat Model

We inherit the fault and threat model from the classical BFT [64] and trusted computing domains [99]. The cloud infrastructure (machines, network, etc.) can exhibit Byzantine behavior and also being subject to attackers that can control over the host CPU (e.g., the OS, VMM, etc.) and the SmartNICs (post-manufacturing). The adversary can attempt to re-program the SmartNIC, but they cannot compromise the cryptographic primitives [64, 112, 162]. The physical package, supply chain, and manufacturer of the SmartNICs are trusted [103, 174]. The TNIC implementation (bitstream) is synthesized by a trusted IP vendor with a trusted tool flow for covert channels resilience.

Since TNIC does not rely on CPU-based TEEs and its network stack and library run on the unprotected CPU, both software can be compromised by a potentially Byzantine actor on the machine. As such, TNIC does not distinguish between different types of untrusted software components. Whether the network library, the network stack, or the application code is compromised, the node is considered faulty (Byzantine) and must conform to the BFT application system model, which should specify its tolerance to Byzantine failures.

3.3 Design Challenges and Key Ideas

While designing TNIC, we overcome the following challenges:

#1: Heterogeneous hardware. CPU-based TEEs in the cloud infrastructure are heterogeneous with different programmability [51, 56, 145, 158, 159, 167] and security properties [123, 128, 135] that complicate their adoption and the system's correctness [145]. Prior systems [57, 74, 119, 162] *could* not address this heterogeneity challenge as they require *homogeneous* x86 machines with SGX extensions of a specific version. This is rather unrealistic in modern heterogeneous distributed systems where system designers are compelled to *stitch heterogeneous TEEs together*. TEE's heterogeneity in programmability and security semantics hampers their adoption and adds complexity to ensuring the system's overall correctness.

Key idea: A host CPU-agnostic unified security architecture based on trustworthy network-level isolation. Our TNIC offers a unified and host-agnostic network-interface level isolation that guarantees the specific yet well-defined security properties of the non-equivocation and transferable authentication. TNIC shifts the security properties from CPU-hosted TEEs to NIC hardware, thereby addressing the heterogeneity and programmability issues associated with CPU-based TEEs. TNIC also offers generic programming APIs (§ 6.1) that are used to *correctly* transform a wide variety of distributed systems for Byzantine settings. We demonstrate the power of TNIC with a generic transformation *recipe* (§ 6.2) and its usage to transform prominent distributed systems (§ 7).

#2: Large TCB in the TEE-based silicon root-of-trust. TEEs based on a *silicon root of trust* are promising for building trustworthy systems [55, 57, 74, 162]. Unfortunately, the state-of-the-art TEEs integrate a *large* TCB; for example, we calculate the TCB size of the state-of-the-art Intel TDX [26]. The TEE ports within the trusted hardware the entire Linux kernel (specifically, v5.19 [38]) and "hardens" at least 2000K lines of usable code, leading to a final TCB of 19MB. Such large TCBs have been accused of increasing the area of faults and attacks [107, 130] of commercial TEEs that are already under fire for their security vulnerabilities [25, 27, 42, 52, 138]. Importantly, TEE's large TCBs complicate their security analysis and verification, rendering their security properties *incoherent*.

Key idea: A minimal and formally verifiable silicon root-of-trust with low TCB. In our work, we advocate that a *minimalistic silicon root of trust* (TCB) at the NIC level hardware is the foundation for building verifiable, trustworthy distributed systems. In fact, TNIC builds a minimalistic and verifiable attestation kernel (§ 4.1) that guarantees the TNIC security properties at the SmartNIC hardware. Moreover, we have formally verified the TNIC secure hardware protocols (§ 4.4).

#3: Performance. TEE's overheads are significant in the context of networked systems [55, 74, 86, 162]. Prior research [55] reported 4×–8× performance degradation with even a sophisticated network stack. Others [57, 74, 162] limit performance due to the communication costs between their untrusted and TEE-based counterparts [103]. The actual performance overheads in heterogeneous distributed systems are expected to be more exacerbated [45, 46]. As such, TEEs cannot *naturally* offer high-performant, trusted networking.

Key idea: Hardware-accelerated trustworthy network stack. Our TNIC bridges the gap between performance and security with two design insights. First, TNIC attestation kernel offers the foundations to transform CFT distributed systems to BFT systems without affecting the number of participating nodes, significantly improving scalability. Second, TNIC user-space network stack (§ 5) bypasses the OS and offloads security and network processing to the NIC-level hardware.

4 Trusted NIC Hardware

Figure 2 shows our TNIC hardware architecture that implements trusted network operations on a SmartNIC device. TNIC introduces two key components: *(i)* the attestation kernel that guarantees the non-equivocation and transferable authentication properties over the untrusted network (§ 4.1) and *(ii)* the RoCE protocol kernel that implements the RDMA protocol including transport and network layers (§ 4.2). We also introduce a bootstrapping and a remote attestation protocol for TNIC (§ 4.3) and formally verify them (§ 4.4).

4.1 NIC Attestation Kernel

The attestation kernel *shields* network messages and materializes the properties of non-equivocation and transferable authentication by generating *attestations* for transmitted messages. As shown in Figure 2, the attestation kernel resides in the data pipeline between the RoCE protocol kernel that transmits/receives network messages and the PCIe DMA that transfers data from/to the host memory. The kernel processes the messages as they *flow* from the memory to the network and vice versa to optimize throughput.

Hardware design. The attestation kernel is comprised of three components that represent its state and functionality: the HMAC component that generates the message authentication codes (MAC), the Keystore that stores the keys used by the HMAC module, and the Counters store that keeps the message's latest sent and received timestamp.

The system designer initializes each TNIC device during bootstrapping with a unique identifier (ID) and a shared secret key—ideally, one shared key for each session—stored in static memory (Keystore). The keys are shared and, hence, unknown to the untrusted parties.

TNIC holds two counters per session in the Counters store: `send_cnts`, which holds sending messages, and `recv_cnts`, which holds the latest seen counter value for each session. The counters represent the messages' timestamp and are increased monotonically and deterministically after every send and receive operation to ensure that unique messages are assigned to unique counters for non-equivocation. Consequently, no messages can be lost, re-ordered, or doubly executed.

Algorithm. Algorithm 1 shows the functionality of the attestation kernel. The module implements two core functions: `Attest()`, which generates a unique and verifiable attestation for a message, and `Verify()`, which verifies the attestation

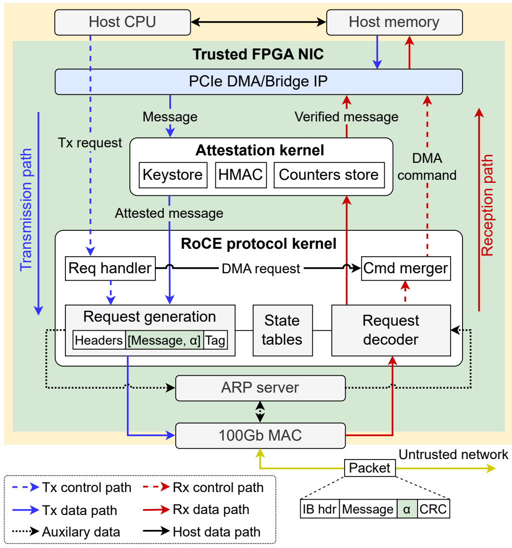

Figure 2. TNIC hardware architecture.

of a received message. The message transmission invokes `Attest()`, and likewise, the reception invokes `Verify()`.

Upon transmission, as shown in Figure 2, the message is first forwarded to the attestation kernel. The attestation kernel executes `Attest()` and generates an *attested* message comprised of the message data and its attestation certificate α. The function calculates α as the HMAC of the message concatenated with the counter `send_cnt` and the device ID (for transferable authentication) with the `key` for that connection (Algo 1: L4). It also increases the next available counter for subsequent future messages (Algo 1: L2). The function forwards the message with its α to the RoCE protocol kernel (Algo 1: L4).

Upon reception, the received message passes through the attestation kernel for verification before it is delivered to the application. Specifically, `Verify()` checks the authenticity and the integrity of the received message by re-calculating the *expected* attestation α' based on the message payload and comparing it with the received attestation α of the message (Algo 1: L7—8). The verification also ensures that the received counter matches the expected counter for that connection to ensure *continuity* (Algo 1: L8).

Algorithm 1: `Attest()` and `Verify()` functions.

```
1  function Attest(c_id, msg) {
2      cnt ← send_cnts[c_id]++;
3      α ← hmac(keys[c_id], msg||ID||cnt);
4      return α||msg||ID||cnt;
5  }
6  function Verify(c_id, α||msg||ID||cnt) {
7      α' ← hmac(keys[c_id], msg||ID||cnt);
8      if (α' = α && cnt = recv_cnts[c_id]++ )
9          return (α||msg||cnt);
10     assert(False);
11 }
```

4.2 RoCE Protocol Kernel

The RoCE protocol kernel implements a reliable transport service on top of the IB Transport Protocol with UDP/IPv4 layers (RoCE v2) [23] (transport and network layers). As shown in Figure 2, to transmit data, the `Req handler` module in the RoCE kernel receives the request opcode (`metadata`) issued by the host. The message is fetched through the PCIe DMA engine and passes through the attestation kernel. The request opcode and the attested message are forwarded to the `Request generation` module that constructs a network packet.

Upon receiving a message from the network, the RoCE kernel parses the packet header and updates the protocol state (stored in the State tables). The attested message is then forwarded to the attestation kernel. The message is delivered to the application's (host) memory upon successful verification.

Hardware design. The RoCE protocol kernel is also connected to a 100Gb MAC IP and an ARP server IP.
100Gb MAC. The 100Gb MAC kernel implements the link layer connecting TNIC to the network fabric over a 100G Ethernet Subsystem [39]. The kernel also exposes two interfaces for transmitting (Tx) and receiving (Rx) network packets.
ARP server. The ARP server has a lookup table containing MAC and IP address correspondences. Right before the transmission, the RDMA packets at the `Request generation` first pass through a MAC and IP encoding phase, where the `Request generation` module extracts the remote MAC address from the lookup table in the ARP server.
IB protocol. The RoCE protocol kernel implements the reliable version of the IB protocol. Similarly to its original specification [1], the kernel implements State tables to store protocol queues (e.g., receive/send/completion queues) as well as important metadata, i.e., packet sequence numbers (PSNs), message sequence numbers (MSNs), and a Retransmission Timer.

Dataflow. The transmission path is shown with the blue-colored axes in Figure 2. The `Req handler` receives requests issued by the host and initializes a DMA command to fetch the payload data from the host memory to the attestation kernel. Once the attestation kernel forwards the attested message to the `Req handler`, the module passes the message from several states to append the appropriate headers `IB hdr` along with metadata (e.g., RDMA op-code, PSN, QP number). The last part of the processing generates and appends UDP/IP headers (e.g., IP address, UDP port, and packet length). The message is then forwarded to the 100Gb MAC module.

In the reception path (red-colored axes in Figure 2), the `Request decoder` extracts the headers, metadata, and attested message. The message is forwarded to the attestation kernel for verification and finally copied to the host memory.

The message format in TNIC follows the original RDMA specification [1]; only the difference between our TNIC and the original RDMA messages is that the attestation kernel *extends* the payload by appending a 64B attestation α and the metadata. The metadata includes a 4B id for the session id of the

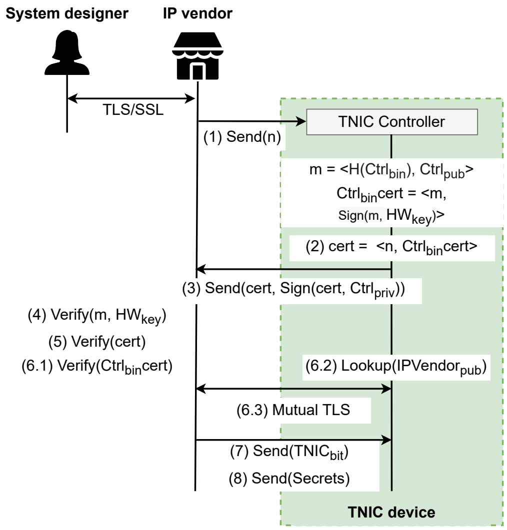

Figure 3. TNIC remote attestation protocol.

sender, a 4B ID for the device id (unique per device), and the appropriate `send_cnt`. This payload extension is negligible and does not harm the network throughput.

4.3 TNIC Attestation Protocol

We design a remote attestation protocol to ensure that the TNIC device is genuine and the TNIC bitstream and secrets are flashed securely in the device.

Boostrapping. The TNIC hardware is securely bootstrapped in an untrusted third-party cloud by the Manufacturer, System designer, and IP vendor, who trust each other. At the device construction, the Manufacturer burns $\mathtt{HW}_{key}$, a secret key unique to the device. It is possible with commercial FPGA cards that have access to an AES key and the hash of a public encrypted key embedded in secure, on-chip, non-volatile storage (Intel [34], AMD [16]). The System designer shares the configuration with the IP vendor and instructs the cloud provider to load the (encrypted) FPGA firmware which is then decrypted with the $\mathtt{HW}_{key}$. The firmware loads the controller binary $\mathtt{Ctrl}_{bin}$, generates a key pair $\mathtt{Ctrl}_{pub,priv}$ for the specific device and binary, and signs the measurement of the $\mathtt{Ctrl}_{bin}$ and the $\mathtt{Ctrl}_{pub}$ with the $\mathtt{HW}_{key}$ ($\mathtt{Ctrl}_{bin}\mathtt{cert}$).

Remote attestation. Figure 3 shows TNIC remote attestation. The IP vendor sends a random nonce `n` for freshness to the Controller. The IP vendors public key $\mathtt{IPVendor}_{pub}$ is embedded into the $\mathtt{Ctrl}_{bin}$. The Controller generates a certificate `cert` over the $\mathtt{Ctrl}_{bin}\mathtt{cert}$ and `n` (2) which signs with $\mathtt{Ctrl}_{pub}$ and sends it to the IP vendor (3).

The IP vendor verifies the authenticity of the `cert` (4)—(5) and establishes a TLS connection with the Controller. First, the vendor verifies the authenticity of `m` with the $\mathtt{HW}_{key}$, ensuring that a genuine $\mathtt{Ctrl}_{bin}$ and a genuine device has signed `m` (4). As such, the vendor ensures that the $\mathtt{Ctrl}_{bin}$ runs in a genuine TNIC device by verifying that the measurement of the $\mathtt{Ctrl}_{bin}$ has been signed with an appropriate device key installed by the Manufacturer. Lastly, the vendor verifies the nonce `n` and `cert` to ensure freshness (with the $\mathtt{Ctrl}_{pub}$ included in `m`).

Now, a mutually authenticated TLS connection is established; the IP vendor verifies authenticity by checking for the desired $\mathtt{Ctrl}_{pub}$ and the Controller checks for it's embedded $\mathtt{IPVendor}_{pub}$ (6.1)—(6.3). Once the TLS connection is established the IP Vendor sends the Controller the secrets and the TNIC bitstream, $\mathtt{TNIC}_{bit}$.

4.4 Formal Verification of TNIC Protocols

We formally verify the safety and security properties of TNIC hardware using Tamarin [125]. Our verification consists of a model for bootstrapping, remote attestation, message transmission, and reception, according to Figure 3. This model is augmented with custom *action facts*, which mark the occurrence of defined events in the execution trace. These include:

1. $\mathrm{D}_e(x)$, which marks the end of the attestation phase for endpoint e, with associated connection information x.
2. $\mathrm{S}_e(m)$ and $\mathrm{A}_e(m)$ marking the sending and accepting of a message m, following Algorithm 1, respectively.

The intended temporal relationship of these action facts is expressed using lemmas, which Tamarin then proves using automated deduction and equational reasoning. The relation $a @ t_i$ expresses that action fact a occurred at time t_i. Using this relation, we can express our desired security properties as follows:

Remote attestation. We define the main attestation lemma for any TNIC device *tnic* and associated IP Vendor *ipv*. The lemma holds if, after the last step of the remote attestation protocol, the TNIC device is in a valid, expected state:

$$\forall\, ipv, tnic, c, t_i.\ \mathrm{D}_{ipv}(c)\ @\ t_i \Longrightarrow \exists\, t_j.\ t_j < t_i \wedge \mathrm{D}_{tnic}(c)\ @\ t_j \quad (1)$$

Transferable authentication. We define the lemma, which states that any accepted message was sent by an authentic TNIC device in a valid configuration:

$$\forall\, e1, m, t_i.\ \mathrm{A}_{e1}(m)\ @\ t_i \Longrightarrow \exists\, e2, t_j.\ t_j < t_i \wedge \mathrm{S}_{e2}(m)\ @\ t_j \quad (2)$$

Non-equivocation. We further extend the model by three lemmas that help to reason about non-equivocation. For any message that is accepted, it holds that *(i)* there is no message that was sent before but not accepted:

$$\begin{aligned} &\forall\, e1, e2, m_j, t_i, t_j.\ \mathrm{A}_{e1}(m_j)\ @\ t_i \wedge \mathrm{S}_{e2}(m_j)\ @\ t_j \\ &\quad \Longrightarrow (\forall m_k, t_k.\ t_k < t_j \wedge \mathrm{S}_{e2}(m_k)\ @\ t_k \\ &\quad\quad \Longrightarrow \exists\, t_l.\ t_l < t_i \wedge \mathrm{A}_{e1}(m_k)\ @\ t_l) \end{aligned} \quad (3)$$

(ii) there is no message that was sent after, but accepted before:

$$\begin{aligned} &\forall\, e1, m_i, m_j, t_i, t_j.\ t_i < t_j \wedge \mathrm{A}_{e1}(m_i)\ @\ t_i \wedge \mathrm{A}_{e1}(m_j)\ @\ t_j \\ &\quad \Longrightarrow \exists\, e2, t_k, t_l.\ t_k < t_l \wedge \mathrm{S}_{e2}(m_k)\ @\ t_k \wedge \mathrm{S}_{e2}(m_l)\ @\ t_l \end{aligned} \quad (4)$$

(iii) this message has not been accepted before:

$$\forall\, e1, m, t_i, t_j.\ \mathrm{A}_{e1}(m)\ @\ t_i \wedge \mathrm{A}_{e1}(m)\ @\ t_j \Longrightarrow t_i = t_j \quad (5)$$

Our complete verification includes additional action facts and lemmas to verify properties like the secrecy of private information and the implications of out-of-band key compromises.

To sum up, Tamarin successfully shows that there is no sequence of transitions that leads to any state where our lemmas

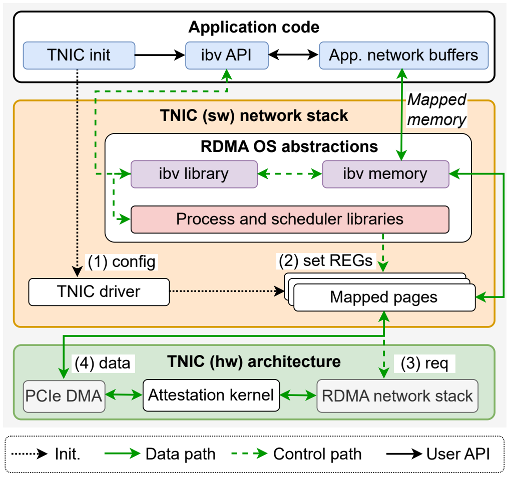

Figure 4. TNIC network system stack.

are violated. Thus, the attestation and transferable authentication lemmas hold for our model, and the counters behave as expected for non-equivocation.

5 TNIC Network Stack

We build a software TNIC system network stack that operates as the *middle layer* between the TNIC programming APIs (see § 6.1) and the hardware implementation of TNIC. Figure 4 shows an overview of the network stack design that is comprised of two core components: *(1)* the TNIC driver and mapped REGs pages that are responsible for initializing the device and configuring host—device communication and *(2)* the RDMA OS abstractions that execute networking operations.

5.1 TNIC Driver and Mapped REGs Pages

The TNIC driver is invoked at the device initialization, before the remote attestation protocol (§ 4.3), to configure the hardware with its static configuration (the device MAC address, the device QSFP port, and the network IP used by the application).

The driver enables kernel-bypass networking—similar to the original (user-space) RDMA protocol—by mapping the TNIC device to a user-space addresses range, the Mapped REGs pages. TNIC reserves one page at the page granularity of our system for each connected device that is represented as pseudo-devices in `/dev/fpga<ID>`. Read and write access to the pseudo-device is equal to accessing the control and status registers of the FPGA. Applications directly interact with the control path of the TNIC hardware bypassing the host OS.

5.2 RDMA OS Abstractions

The RDMA OS abstractions are a user-space library that enables the networking operations in the TNIC hardware, bypassing the OS kernel for performance. As shown in Figure 4, the RDMA OS library is comprised of two parts: *the network (RDMA) library* (colored in purple) that implements the software part of the RDMA protocol and *the OS library* (colored in red) that schedules the TNIC requests.

Initialization APIs	
`ibv_qp_conn()`	Creates an ibv struct
`alloc_mem()`	Allocates host ibv memory
`init_lqueue()`	Registers local memory to TNIC
`ibv_sync()`	Exchanges ibv memory addresses
Network APIs	
`local_send/verify()`	Generates/verifies attested messages
`auth_send()`	Transmits an attested message
`poll()`	Polls for incoming messages
`rem_read/write()`	Fetches/writes remote memory

Table 1. TNIC programming APIs.

Network (RDMA) library. The network (RDMA) library includes all the logic and data (e.g., Tx/Rx queues per connection, local and remote memory addresses, RDMA keys that denote memory access permissions) required to implement the RDMA protocol. It executes the application's networking operations by posting the requests to the hardware. More specifically, it creates an internal representation of the request and the associated data and metadata (i.e., request opcode, remote IP, source/destination addresses, data length, etc.) and writes them into specific offsets in the REGs pages to update the control registers of the TNIC hardware.

As shown in Figure 4, the transmission and reception of requests and responses mandate the allocation of application network buffers. We adopt memory management similar to that in widely used user-space networking libraries [24, 101, 126]. Importantly, the network buffers need to be mapped to a specific TNIC-memory, called the ibv memory. The ibv memory area is allocated at the connection creation in the huge page area by the application through the ibv library. It resides within the application's address space with full read/write permissions and is eligible for DMA transfers.

OS library. The TNIC-OS library is responsible for scheduling the requests and ensuring isolated access to the mapped REG pages. For performance, the TNIC data path eliminates unnecessary data copies throughout the network stack; the data to be sent is directly fetched by the hardware through DMA transfers. The OS library creates a TNIC-process object to represent each TNIC device. This TNIC-process in TNIC is not a separate scheduling entity (i.e., a thread as in classical OSes). In contrast, it is an object handle, exposed to the ibv library but managed by the TNIC-OS library that acquires locks on the respective REG pages to ensure isolated access to the TNIC hardware.

6 TNIC Network Library

We present TNIC's programming APIs (§ 6.1), and a generic recipe to transform existing distributed systems (§ 6.2).

6.1 Programming APIs

TNIC implements a programming API (Table 1) that resembles the traditional RDMA programming API [101] used in modern distributed systems[78, 84, 88, 102, 105, 122, 131]. We further extend the security semantics, offering the properties of non-equivocation and transferable authentication (§ 2.1).

Initialization APIs. The TNIC application first needs to configure the TNIC system to establish peer-to-peer RDMA connections. The application creates one ibv struct for each connection with `ibv_qp_conn()`, which sets up and stores the queue pair, the local and remote IP addresses, device UDP ports, and others. The application also invokes `alloc_mem()` to allocate the ibv memory and then register the ibv memory to the TNIC hardware. Lastly, the application synchronizes with the remote machine using `ibv_sync()` to exchange necessary data (e.g., ibv memory address, queue pair numbers).

Network APIs. TNIC executes trusted one-sided, reliable RDMA with the same reliability guarantees as the classical one-sided RDMA over Reliable Connection (RC), i.e., a FIFO ordering (per connection), similar to TCP/IP networking.

TNIC offers `auth_send()` to send an attested message with RDMA reliable writes. We support classical RDMA operations for reads and writes: `rem_read()` and `rem_write()`. The remote side runs `poll()` to fetch the number of completed operations; `poll()` is updated only when the message verification succeeds at the TNIC hardware. We offer local operations for generating and verifying attested messages within a single-node setup: `local_send()` and `local_verify()`.

TNIC does not support a hardware-assisted multicast, but it can offer equivocation-free multicast uni-casting the same attested message generated by `local_send()` as in [112].

6.2 A Generic Transformation Recipe

Transformation properties. We show how to use TNIC APIs to transform an existing (CFT) distributed system into one that targets Byzantine settings. Our transformation is defined as wrapper functions on top of the send and receive operations [68]. For safety, our transformation needs to meet the following properties to provide the same guarantees expected by the original CFT system [68, 93, 94]:

Safety. If a correct receiver receives a message m from a correct sender S, then S must have sent with m.

Integrity. If a correct receiver receives and delivers a message m, then m is a *valid* message.

```
1  void send(P_id, char[] msg) {
2     state = hash(my_state);
3     tx_msg = msg || state || receiver_state;
4     auth_send(follower, tx_msg);
5  }
6  void recv(recv_msg) {
7     auto [att, msg || state || receiver_state] = deliver();
8     [msg, cnt] = verify_msg(msg);
9     verify_sender_state(state);
10    local_verify(receiver_state);
11    verify_system_view(receiver_state); apply(msg);
12 }
```

Listing 1. Generic send and recv wrapper functions using TNIC. TNIC additions are highlighted in orange.

Listing 1 shows our proposed `send` (L1-5) and `recv` (L7-13) operations, providing a general method for transforming a CFT system into a BFT system. We assume a two-node scenario where the first node (sender) receives client requests and forwards them to the second node (receiver). For transmission, the sender sends the client message `msg`, its current state (e.g., the sender's action to the `msg`), and the receiver's previous state (known to the sender). The receiver's state is optional depending on the consistency guarantees of the derived system and can be used to ensure that both nodes have the same system view.

Upon receiving a valid message (L8-9), the receiver *simulates* the sender's state to verify that the sender's action to the request is as expected (L10). The way to simulate the states depends on the applications, e.g., in our BFT protocol implementation (§ 7), each replica maintains copies of counters that represent the expected counter values for all other participating nodes. The simulation allows nodes to avoid replaying the entire message history in order to reconstruct the system's state, as done in [68]. The receiver also ensures that it does not lag, and both nodes have the same "view" of the system inputs by verifying that the sender has *seen* the receiver's latest state (e.g., message) (L11-12).

Our generic transformation satisfies the requirements listed above. First, TNIC's transferable authentication property directly satisfies the safety requirement. A faulty node cannot impersonate correct nodes; if TNIC validates a message m from a sender, the sender must have sent m. TNIC's reliable network operations ensure liveness properties between correct nodes. Second, our transformation satisfies the integrity property. The integrity property is ensured by validating that the sender's response to the client's request follows the protocol specification. The transferable authentication mechanism allows correct receivers to prove the integrity flow by simulating the sender's output and state, e.g., by maintaining a copy of the sender's state.

Consistency property for replication. Our transformation further needs to meet the consistency property [68]. If correct receivers R_1 and R_2 receive valid messages m_i and m_j respectively from sender S, then either (a) Bpg_i is a prefix of Bpg_j, (b) Bpg_j is a prefix of Bpg_i, or (c) $Bpg_i = Bpg_j$ (where Bpg_x is the process behavior that supports the validity of message m_x).

The consistency requirement is enforced through the TNIC's non-equivocation primitive that assigns a (unique) monotonic sequence number to each outgoing message, enforcing a total order on the sender's outgoing messages. Along with the integrity requirement, the total order can prevent equivocation and suffice for consistency. Importantly, TNIC ensures that correct receivers cannot miss any past messages. Following this, two followers that receive from the same sender (using the equivocation-free multicast discussed in § 8.2) follow the same transition (execution) path. TNIC cannot transform systems with non-deterministic specifications.

7 Trusted Distributed Systems

Using TNIC as the foundation, we transform the following four distributed systems to operate in Byzantine environments.

Attested Append-Only Memory (A2M). We design an Attested Append-Only Memory (A2M) [67] leveraging TNIC, which can be used to shield and optimize various systems [43, 64, 72, 115]. The original A2M, and hence our implementation over TNIC, builds append-only (trusted) logs, associating each entry with a monotonically increasing sequence number to combat equivocation. While A2M has a large TCB and ports the log within the TEE, our implementation has only a minimal TCB in hardware and it can robustly store the log in the untrusted host memory, improving memory efficiency [112].

As in the original A2M, we build the `append` and `lookup` operations. The `append` calls into TNIC to generate an attestation for the log entry while associating it with a monotonically increased sequence number (`sent_cnt`). The sequence number denotes the entry's position in the log. The `lookup` operation retrieves entries locally without verification.

Byzantine Fault Tolerance (BFT). We design a Byzantine Fault-Tolerant protocol (BFT) using TNIC. The protocol builds a replicated counter as a foundational service for various systems [76, 97, 100, 143, 163]. Our system model considers a network of replicas with at most f Byzantine replicas out of $N=2f+1$ total replicas. One replica serves as the leader, and the others act as followers. The system prevents equivocation through TNIC, which enforces and validates the ordering of messages. This reduces the number of replicas required and the message complexity compared to the classical BFT ($3f+1$).

Clients send increment counter requests to the leader, who performs the requests and broadcasts the change along with a *proof of execution* (PoE) message to followers. The proof of execution is a TNIC-attested message with the original client's request, the leader's counter value, and metadata. The followers leverage their local state machine to detect a faulty leader (or follower) [94]. Subsequently, if and only if a follower has not applied the message before, it applies the incremented counter value to its state machine before forwarding its own PoE message to all other replicas and replying to the client. A quorum of at least $f+1$ identical messages from different replicas guarantees a correctly committed result for the client.

Chain Replication (CR). We design a Byzantine CR system [160] using TNIC as the replication layer of a Key-Value store. As in the CFT version of CR, the replicas, e.g., head, middle, and tail, are connected in a chained fashion.

Clients execute requests by forwarding them to the head. The head orders and executes the request, creating his own *proof of execution message* (PoE), which is sent along the chain. The PoE consists of the original request and the head's output that TNIC attests. Each node in the chain verifies the previous node's PoE, executes the request, and creates its own PoE, which consists of the last PoE and the node's output.

System	(host) TEE-free	Tamper-proof
SSL-lib	Yes	No
SSL-server/Intel-x86*/AMD	Yes	No
SGX/AMD-sev	No	Yes
TNIC	Yes	Yes

Table 2. Host-sided baselines and TNIC. (*) We use the term SSL-server for this run unless stated otherwise.

Unlike the CFT CR, local operations in the tail (e.g., reads) are untrusted in the BFT model. Therefore, all operations must traverse the entire chain. Replicas reply to clients with their output after forwarding their PoE message, and clients wait for identical replies from all chained nodes. We discuss the performance-security trade-offs of an alternative TEE-based design of porting the entire CR protocol into the TEE (that would allow clients to read only from the tail) in § 8.3.

Accountability (PeerReview). Lastly, we design an accountability system with TNIC based on the PeerReview system [90] to *detect* malicious actions in large deployments [129, 156]. We detect faults impacting the system's network messages logged into the participant's tamper-evident log. We frame the protocol within an overlay multicast protocol for streaming systems where the nodes are organized in a tree topology. Witnesses assigned to each node audit the node's log to detect faults or non-responsive nodes. The witnesses replay the log entries, comparing them with a reference deterministic implementation to identify inconsistencies. Our TNIC prevents equivocation at NIC hardware efficiently, which eliminates the expensive all-to-all communication of the original PeerReview that does not use trusted hardware [112].

8 Evaluation

We evaluate TNIC across three dimensions: *(i)* hardware (§ 8.1), *(ii)* network stack (§ 8.2) and *(iii)* distributed systems (§ 8.3).

Evaluation setup. We perform our experiments on a real hardware testbed using two clusters: AMD-FPGA Cluster and Intel Cluster. AMD-FPGA Cluster consists of two machines powered by AMD EPYC 7413 (24 cores, 1.5 GHz) and 251.74 GiB memory. Each machine also has two Alveo U280 cards [3] that are connected through 100 Gbps QSFP28 ports. Intel Cluster consists of three machines powered by Intel(R) Core(TM) i9-9900K (8 cores, 3.2 GHz) with 64 GiB memory and three Intel Corporation Ethernet Controllers (XL710).

8.1 Hardware Evaluation: T-FPGA

Baselines. We evaluate the performance of `Attest()` of the TNIC's attestation kernel (§ 4.1) compared with four host-sided systems shown in Table 2. For these host-sided versions, we build OpenSSL v3.1 servers that run natively or within a TEE with the same BIOS configuration (AES-NI enabled). The servers attest and forward network messages to the host application. We use the terms Intel-x86 and AMD for a native run of the server process (SSL-server) and SGX and AMD-sev for their tamper-proof versions within a TEE. The TEE baselines

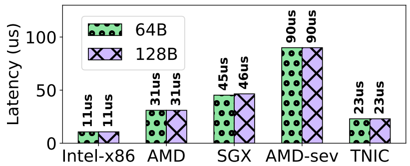

Figure 5. `Attest` function latency.

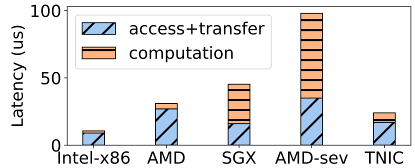

Figure 6. `Attest` latency breakdown.

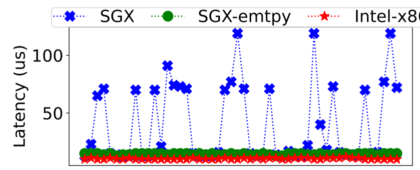

Figure 7. Latency over time (SGX).

follow the same system model as in state-of-the-art hybrid systems [74, 103, 112, 162], where the host BFT application runs on the untrusted CPU and communicates with a separate TEE-based process to generate and verify message attestations. TNIC implements similar abstractions for counter and message attestation. Thus, TNIC does not introduce additional protocol alterations compared to them.The server and host process run in the same machine to eliminate network latency and communicate through TCP sockets. We implement SGX using the SCONE framework [51] while AMD-sev runs in a QEMU VM using the official VM image [6]. In addition, we build (non-temper-proof) SSL-lib, which integrates the `Attest` function as a library.

Methodology and experiments. We use Vitis XRT v2022.2 and the shell `xilinx_u280_gen3x16_xdma_base_1` for the stand-alone evaluation of the TNIC attestation kernel: synchronous data transfers between the host and device (U280). We measure and report the average latency and communication costs by executing an empty function body of `Attest()`.

Results. Figure 5 shows the average latency of `Attest()` based on the HMAC algorithm for 64B and 128B data inputs. The latency of `Verify()` is similar, and as such, it is omitted. Our TNIC achieves performance in the microseconds range (23 us) and outperforms its equivalent TEE-based competitors at least by a factor of 2. Importantly, TNIC is approximately 1.2× faster than AMD, which is not tamper-proof.

Figure 6 shows the latency breakdown of `Attest()`. Accessing the TNIC device and TEEs can be expensive, ranging from 30% to 90% of the total operation latency among the systems. Regarding TNIC, the transfer time (16us) accounts for 70% of the execution time. We expect that TNIC effectively eliminates this cost by enabling asynchronous (user-space) DMA data transfers. Regarding the TEE-based systems (SGX, AMD-sev), the communication and system call execution costs account for up to 40% of the total execution. To our surprise, this implies that the HMAC computation within any of the two TEEs experiences more than 30× overheads compared to its native run. To analyze TEEs' behavior, we instrument the client's code to measure the operations' individual latency at various periods of time during the experiment accurately.

Figure 7 illustrates the individual operation latency, where SGX-empty indicates SGX without HMAC computation. As shown in Figure 7, the HMAC execution within the TEE often experiences huge latency spikes. We attribute this behavior to the scheduling effects and asynchronous exitless system calls inherent in our SGX framework, SCONE [51]. We observe similar latency variations during executions on AMD systems, spiking up to 200–500 us.

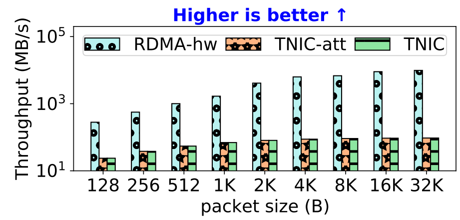

Figure 8. Throughput of send operations across the three selected network stacks.

8.2 Software Evaluation: TNIC Network Stack

Baselines. To evaluate the TNIC performance, we discuss (1) the effectiveness of offloading the network stack to the TNIC hardware and (2) the overheads incurred by the CFT systems transformation for the BFT model. We compare TNIC across four other software/hardware network stacks with different security properties as follows: *(i)* RDMA-hw, an untrusted RoCE protocol on FPGAs, *(ii)* DRCT-IO (direct I/O), untrusted software-based kernel-bypass stack, *(iii)* DRCT-IO-att, altered DRCT-IO that offers trust by sending attested messages but does not verify them, and *(iv)* TNIC-att, altered TNIC that similarly sends attested messages without verification. We build *(i)* RDMA-hw on top of Coyote [15] network stack similarly to TNIC. For *(ii) (iii)* DRCT-IOs, we base our design on eRPC [101] with DPDK [24] that offers similar reliability guarantees with RDMA-hw. The hardware network stacks run on AMD-FPGA Cluster, whereas the rest run on Intel Cluster.

Methodology and experiments. Our experiments measure the latency and throughput for respective network stacks, which run through a single-threaded client-server implementation. For the latency measurement, the client sends one operation at a time, whereas for the throughput measurement, one client can have multiple outstanding operations.

Results. Figure 9 and 8 show the latency and throughput of the send operation with various packet sizes. First, regarding (1) the effectiveness of network stack offloading, RDMA-hw is 3×—5× faster than DRCT-IO, which indicates that the network offloading boosts performance. Although DRCT-IO offers minimal latency (16-16.6us) for small packet sizes up to 1 KiB due to its zero-copy transmission/reception optimizations [101], they are only effective for up to 1460B (MTU is 1500B, but 40B are reserved for metadata), and RDMA-hw still achieves 3× lower latency (5-5.5us). For bigger data transfers,

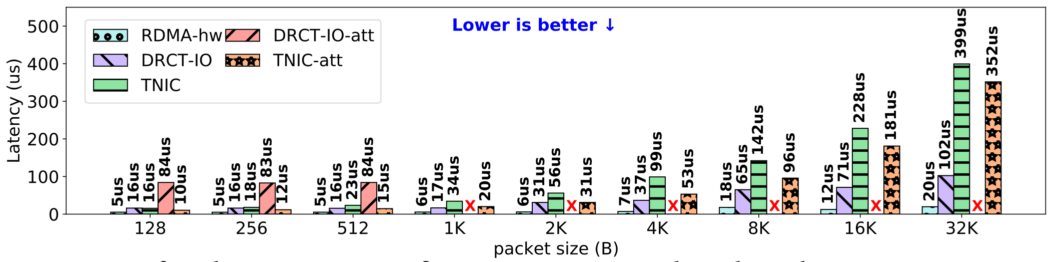

Figure 9. Latency of send operations across five competitive network stacks with various security properties.

the RDMA-hw latency increases steadily up to 19 us, whereas DRCT-IO does not scale well with latencies up to 100us.

Second, regarding (2) the TNIC performance overhead, TNIC offers trusted networking with 3×—20× higher latencies than the untrusted RDMA-hw. The latency increase stems from the HMAC calculation of the TNIC hardware. As this algorithm fundamentally cannot be parallelized, the higher the message size, the higher the latency our TNIC incurs. More specifically, for packet sizes less than 1 KiB, doubling the packet size in TNIC results in a 13%—20% increment in the overall latency. For packet sizes bigger or equal to 1 KiB, doubling the packet size increases the latency by 30%—40%. Compared to DRCT-IO-att (82us), TNIC is up to 5.6× faster. Importantly, DRCT-IO-att reports extreme latencies (2000us or more) for packet sizes larger than 521B, which are omitted to avoid plot distortion. We attribute these latencies to the scheduling effects of SCONE [51].

8.3 Distributed Systems Evaluation

We next evaluate four distributed systems described in § 7.

Methodology and experiments. We execute all four of our codebases on Intel Cluster in three servers (as the minimum required setup capable of withstanding a single failure, $N = 2f+1$, where $f = 1$). We only use a single port of the U280 for network communication because of a limitation introduced in our system by the Coyote codebase [15], on top of which we base TNIC implementation. Due to our limited resources, we cannot install Alveo U280 cards on all these servers. Instead, we build our codebase using the DRCT-IO stack (detailed in § 8.2) and inject busy waits to emulate the delays incurred by TNIC for generating and verifying attested messages.

We evaluate each codebase using five systems that generate and verify the attestations: *(i)* SSL-lib (no tamper-proof), *(ii)* SSL-server (no tamper-proof), *(iii)* SGX, *(iv)* AMD-sev, and *(v)* TNIC. To perform a fair comparison, we integrate into our codebases a library that accurately emulates all latencies (measured in § 8.1) within the CPU. For the AMD latency, we use 30us, representing the lower bound of the latencies measured in § 8.1. We do not emulate the SSL-lib latency.

Given that DRCT-IO, which is used for the emulation, is at least 3× slower than the hardware RDMA network stack (RDMA-hw), the latencies outlined in this section are anticipated to reflect the upper limit for all four systems with TNIC.

We additionally evaluate two TEEs-hosted CFT replication protocols (TEEs-Raft and TEEs-CR) where the entire protocol codebase (Raft [133] and Chain replication [161] respectively) resides within the TEE. We compare the TEEs-hosted systems with TNIC and discuss the trade-offs between their threat model, TCB, and performance.

	Throughput (Op/s)		Latency (us)	
System	append	lookup	append	lookup
SSL-lib	790K	256M	1.26	0.0039
SGX-lib	380K	3.8M	2.6	0.26
AMD-sev	30K	263M	32.37	0.0038
TNIC	158K	257M	6.34	0.0039

Table 3. Throughput and latency of A2M.

A2M. We first evaluate our TNIC-A2M system. We evaluate two TEE baselines: SGX-lib, which places the entire log within the TEE, and AMD-sev, which places the attested log outside the TEE as in the implementation of TrInc [112] and has been shown to be effective. In this experiment, we construct a 9.3GiB log with 100 million entries and then lookup them sequentially/individually.

Results. Table 3 shows the throughput and mean latency of the append/lookup operations. The native execution (SSL-lib) achieves the highest throughput as it incurs no communication costs. Compared to SSL-lib, SGX-lib experiences only a 2× slowdown because we avoid the costly communication w.r.t. an SGX-based server implementation. On the other hand, AMD-sev, which runs the SSL server, incurs a 15× slowdown. Lastly, TNIC incurs 5× and 2.4× slowdown compared to SSL-lib and SGX-lib, respectively, due to the HMAC calculation.

Regarding the lookup operation, SSL-lib, AMD-sev, and TNIC report similar throughput and latency because they lookup untrusted host memory for the requested entry. However, SGX-lib reports a 66× slowdown due to its trusted memory size constraints and expensive paging mechanism [86] because we have to support a log of 9GB within the SGX enclave that only provides 94MB of memory. In contrast, AMD-sev is faster as it only accesses the untrusted host memory. Similar findings have also been demonstrated in [112]. As a result, while TNIC increases append latencies, it greatly optimizes lookup latencies due to its minimal TCB.

BFT. We evaluate the performance of our BFT protocol with various network batching factors. We implement network batching as part of the application's message format.

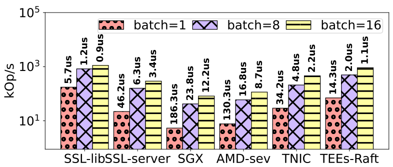

Figure 10. Throughput (and latency numbers) of BFT.

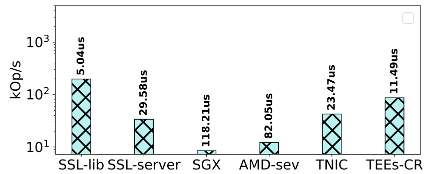

Figure 11. Throughput (and latency numbers) of Chain Replication.

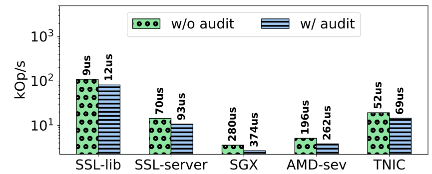

Figure 12. Throughput (and latency numbers) of PeerReview.

Results. Figure 10 shows the throughput and latency of the protocol, which highlights that TNIC significantly outperforms TEE-based versions (SGX, AMD-sev), improving the throughput and latency 4—6×. On the other hand, TNIC incurs 2.4× throughput overhead and up to 7× higher latency compared to SSL-lib. We recall that SSL-lib is not tamper-proof (Table 2) and eliminates the communication overheads incurred by other tamper-proof solutions (SGX, AMD-sev).

We also observe that batching improves the throughput and latency proportionally to the number of batched messages. For all except SSL-lib, the batching factors equal to 8 and 16 achieve 7× and 15× higher throughput than without batching, respectively. For SSL-lib, they are moderately effective: approximately 4—6× faster. It is primarily because the native execution of the attestation function is fast enough to saturate the network bandwidth. As such, conventional techniques can drastically eliminate the overheads for BFT and improve TNIC's adoption into practical systems.

CR. In this experiment, we evaluate the performance of our CR. We allocate one message structure per client request comprising 60B context, 4B operation type, and a 32B signature.

Results. Figure 11 shows the throughput and latency of our Chain Replication. We highlight that our TNIC is 5× and 3.4× faster than SGX and AMD-sev, respectively. While TNIC incurs 4.6× overheads compared to SSL-lib, it is 30% faster than SSL-server, which is not tamper-proof. The performance benefit stems primarily from hardware acceleration by the TNIC's attestation kernel on the transmission/reception data path.

PeerReview. We evaluate our PeerReview system's performance by both activating and deactivating the audit protocol. The system uses one witness for the source node that *periodically* audits its log. In our experiments, the witness audits the log after every send operation in the source node until both clients acknowledge the receipt of all source messages.

Results. Figure 12 shows the throughput and latency of our PeerReview system with and without enabling the audit protocol. Without the audit protocol, the TEE-based systems (SGX, AMD-sev) result in up to 30× slower throughput than SSL-lib, whereas our TNIC mitigates the overheads: 3—5× better throughput compared to AMD-sev and SGX.

Similarly, TNIC outperforms AMD-sev and SGX by 3.7—5× with the audit protocol. Importantly, when using TNIC, the audit protocol itself consumes about 25% (17us) of the overall latency, leading to 1.33× performance slowdown. However, even with the audit protocol, TNIC offers 3.7—5.42× lower latency compared to its TEE-based competitors.

		TCB size (LoC)			
System	Threat model	OS	Att. kernel	App	Total
TEEs-Raft	CFT	2,307K	1,268	856	2,309K
TEEs-CR	CFT	2,307K	1,268	992	2,309K
TNIC	BFT	-	2,114	-	2,114

Table 4. TNIC compared with TEE-hosted applications.

TEEs-hosted baselines. We compare TNIC with TEEs-hosted systems implementing two prototypes based on the failure-free execution of Raft (TEEs-Raft) and CR (TEEs-CR). The code runs within three AMD-sev machines. Prior works [49, 55] suggested this setup for performance—however, at the cost of (1) significantly increased TCB size and (2) a weaker threat model from the application perspective. Table 4 summarizes the security costs. Regarding (1), the TCB of TEEs-hosted systems includes the entire OS [77], OpenSSL libraries for messages authentication [20] (labeled as Att. kernel), and the application codebase, which is over 2M LoCs in total. In contrast, TNIC's TCB only includes our hardware attestation kernel, which is 2,114 LoC of HLS/HDL code. It is only 0.09% of TEE-hosted systems. Regarding (2), the TEE-hosted application can only fail by crashing; it can be thought to remain protected from a potentially Byzantine cloud environment, whereas TNIC targets BFT settings, handling up to f arbitrary failures.

We compare TEE-Raft with our TNIC-based BFT (Figure 10) as both are broadcast-based protocols, and TEEs-CR with our TNIC-based CR (Figure 11) as both require all messages to traverse the entire chain of nodes. TEE-Raft achieves approximately 2.5× higher throughput than TNIC-based BFT. The performance difference is primarily due to Raft's one-phase commitment compared to our TNIC-based BFT. Similarly, TEE-CR achieves 2× higher throughput than the TNIC-based CR. While both versions of CR involve the same number of network Round-Trip Times (RTTs), TNIC involves a higher number of the attestation kernel invocations to verify all the chained messages in the PoE.

8.4 FPGA Resource Usage

Lastly, we perform a resource utilization analysis to show TNIC's scalability capabilities. We measure the resource consumption of TNIC's primary hardware components [92] and estimate maximum connections on the latest FPGA.

Name	LUT (%)		FF (%)		RAMB36 (%)	
U280	1303680	(100)	2607360	(100)	2016	(100)
TNIC	216905	(16.6)	423891	(16.3)	335	(16.6)
XDMA	48258	(3.7)	50701	(1.9)	64	(3.1)
Att. kernel	34138	(2.6)	56914	(2.2)	81	(4.0)
RoCE	30379	(2.3)	75804	(2.9)	46	(2.3)
CMAC	1484	(0.1)	3433	(0.1)	0	(0.0)

Table 5. TNIC's resource usage. The relative (%) compares with the U280 FPGA capacity. TNIC means the entire design.

Table 5 shows the resource consumption details. The overall TNIC design consumes 16.6% of LUTs, 16.3% of Flip-Flops (FF), and 16.6% of RAMB36 (3.46 % of the entire on-chip memory) on the U280 FPGA. Note that TNIC only requires commodity FPGA NIC designs to add the attestation kernel, whose utilization is comparable with the other modules, XDMA and RoCE.

Figure 13 shows the scaling capabilities of TNIC hardware. As the number of network connections increases, we only need to replicate the attestation kernel because the XDMA and CMAC modules are independent of the number of connections, and the RoCE kernel is configured to hold up to 500 connections [148]. The result demonstrates that TNIC can support up to 32 concurrent connections on a single U280 FPGA.

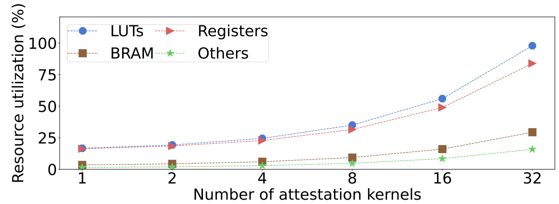

Figure 13. The scalability analysis of TNIC hardware. The resource usage is normalized to the U280 FPGA capacity.

PCIe transaction encryption. TNIC encrypts PCIe transactions for CPU-to-device communication, allowing attackers to modify the PCIe transactions. This vulnerability is not unique to TNIC; it applies to any network stack, including the OS-based ones, since the *untrusted* OS drives PCIe transactions.

8.5 Discussion

TNIC's applicability. As FPGA-based SmartNICs are widely adopted by major cloud providers for hardware acceleration [82], we believe that TNIC has the potential for broader industry application. In addition, ASIC-based NICs can also provide the same functionalities by integrating TNIC's hardware modules into an optimized System-on-Chip (SoC).

Use cases. The paper deliberately focuses on distributed cloud applications as TNIC's primary use cases. Trust in shared third-party clouds is a more critical concern than in other environments, posing unique challenges in trust, performance, and scalability. While the current scope is specific, the underlying principles could extend to other use cases, such as HPC or on-premise computing.

Message drops. TNIC guarantees packet retransmission between two correct nodes until their successful reception extending a RoCE implementation that supports reliable operations. The application need not re-send the message as it receives a different sequence number, which is not accepted (or verified) by the remote TNIC until all preceding messages have been received.

View-change and recovery. Detailing view-change and recovery in TNIC protocols are beyond the scope of our work. TNIC could adopt similar techniques as in TrInc [113] without disrupting these operations. In a new leader's election, replicas can establish new connections with new identifiers. As such, previous connections will not block execution, and state transfers, e.g., view-change, can be performed effectively.

9 Related work

Trustworthy distributed systems. Classical BFT systems [44, 54, 59, 64–66, 152, 157] provide BFT guarantees at the cost of high complexity, performance, and scalability overheads. TNIC bridges the gap between BFT and prior limitations, designing a *silicon root-of-trust* with generic trusted networking abstractions that materialize the BFT security properties.

Trusted hardware for distributed systems. Trustworthy systems [55, 74, 74, 86, 89, 139, 162] leverage trusted hardware to optimize the performance of classical BFT at the cost of generalization and easy adoption. The systems suffer from high latencies (50us—105ms) [103, 112], build large TCBs [55, 86], and rely on specific TEEs [57, 162]. In contrast, our TNIC aims to offer performance and generality, while our minimalistic TCB is verifiable and unified in the heterogeneous cloud.

SmartNIC-assisted systems. Networked systems offer fast network operations with emerging (programmable) SmartNIC devices [3, 9, 11, 28, 30–32, 40]. Some of them offload the network functions to the hardware and reduce the host processing and energy overheads [50, 82, 87, 98, 110, 117, 127, 137, 146, 150, 153, 154] or re-design generic networking protocols, from RDMA/RoCE to TCP/IP network stacks, on top of FPGA-based SmartNICs for performance [5, 15, 83, 95, 141, 148, 165]. Others [106, 111, 114, 116, 118, 120, 121, 124, 134, 136, 142, 144] build generic execution frameworks to optimize a wide variety of distributed systems. Our TNIC follows a similar approach by building a high-performant unified network stack with SmartNICs and extending its security semantics with the properties of non-equivocation and transferable authentication.

Programmable HW for network security. Programmable hardware, SmartNICs, and switches are used to shield networking. Recent systems [104, 155, 164, 170, 173] leverage programmable switches and FPGAs to offload security processing and boost performance in the context of blockchain systems [155] or security functions (e.g., access control, DNS traffic inspection) [104, 170, 173]. Our TNIC similarly offloads security into the hardware, but it carefully uses SmartNICs to overcome the processing bottlenecks of the switches.

Acknowledgements

This work was supported in parts by an ERC Starting Grant (ID: 101077577), the Intel Trustworthy Data Center of the Future (TDCoF), and the Chips Joint Undertaking (JU), European Union (EU) HORIZON-JU-IA, under grant agreement No. 101140087 (SMARTY). The authors acknowledge the financial support by the Federal Ministry of Education and Research of Germany in the programme of "Souverän. Digital. Vernetzt.". Joint project 6G-life, project identification number: 16KISK002.

Appendix

A Full Version

The full version of the paper [35] covers (1) the formal verification proofs of the TNIC security protocols and (2) the implementation details of four distributed systems using TNIC.

B Artifact Appendix

B.1 Abstract

Our artifacts include the TNIC codebase as well as the software artifact with the four TNIC applications, i.e., A2M, BFT, CR, and PeerReview. In addition, we provide the codebases of all the microbenchmarks we discuss in the paper including those of the TEE-based systems. Lastly, we attach the security proofs of TNIC system operations and attestation protocol based on Tamarin [125]. This appendix provides the necessary information to set up, build, and run the experiments we present in the paper.

B.2 Artifact check-list (meta-information)

- **Program:** TNIC hardware implementation codebase. TNIC software codebases that include the systems where TNIC has been applied (run in emulated hardware) and microbenchmarks (e.g., network benchmark). TNIC's security proofs based on Tamarin [125].
- **Compilation:** Requires Vitis HLS [168], Vivado [169], CMake, C++, Boost, eRPC [101], DPDK [24], Tamarin [125].
- **Run-time environment:** Requires NixOS, 5.15.4, SCONE [51] (for SGX-based experiments).
- **Hardware:** Requires Alveo U280 cards [3], Intel(R) Core(TM) i9-9900K with Intel Corporation Ethernet Controllers (XL710) (or any other DPDK compatible NIC) and AMD EPYC 7413.
- **Execution:** The time of the experiments are configurable. Each of our experiments did not take more than 10 minutes. However, the compilation and synthesis phases of the TNIC hardware implementation might take up to 4 hours.
- **Metrics:** Throughput and latency
- **Publicly available:** Yes.
- **Code licenses:** MIT License. TNIC doesn't use any external license.
- **Archived (DOI):** 10.5281/zenodo.14775354

B.3 Description

B.3.1 How to access. The open-source version of the TNIC codebase can be found on GitHub at the following address:
https://github.com/TUM-DSE/TNIC-main.git

B.3.2 Hardware dependencies. For AMD-SEV and TNIC-hardware setups, you need three machines with AMD EPYC 7413 CPU. Each machine is equipped with an Alveo U280 card [3] and one of every U280's QSFP28 ports connects to the 100Gbps network. For Intel SGX setups, you need machines with Intel(R) Core(TM) i9-9900K with Intel Corporation Ethernet Controllers (XL710) (or any other DPDK compatible NIC) for network connection.

B.3.3 Software dependencies. The software build process involves building the low-level Linux kernel driver and the high-level user application layers. All codebases run on top of NixOS, 5.15.4. We provide the appropriate `.nix` files to set up a `nix-shell` environment with all necessary system dependencies.

The code has been built with `Makefile` and `cmake`. The applications, as well as the TEE-based code and application layer, are written in C++17. We depend on Boost libraries and gflgas for the parsing of the command line arguments. We rely on several other dependencies, which we explain in our README files, including; SCONE [51] for SGX-based experiments, Vivado [169] and Vitis HLS [168] for building TNIC hardware, eRPC [101], DPDK [24], and Tamarin [125].

B.4 Installation

The artifact is linked to the repository as submodules. Each repository provides analytical instructions in their `README.md` files of how to build and run the binaries.

To build the TNIC's hardware implementation, please follow the instructions provided in [12].

To build the software including the driver and the benchmarks, please follow the instructions in [13].

To run the experiments for the TNIC hardware implementation, you need to first load the TNIC's kernel module and then run the compiled binary. Detailed instructions are available in [22].

Similar instructions have been documented for the applications [36] and the security proofs [37].

B.5 Evaluation and expected results

Each of the experiments will output information about its progress; this is a hint that the script is still running and hasn't halted. The output of the experiment reports important measurements about the execution. The results are expected not to vary significantly (less than 5%) when compared to the results presented in the paper. However, as discussed, we observed quite a significant variance in some TEE-based systems (Intel SGX and AMD-SEV).

B.6 Methodology

Submission, reviewing, and badging methodology:

- https://www.acm.org/publications/policies/artifact-review-badging
- http://cTuning.org/ae/submission-20201122.html
- http://cTuning.org/ae/reviewing-20201122.html

References

[1] [n. d.]. A Remote Direct Memory Access Protocol Specification. https://datatracker.ietf.org/doc/html/rfc5040. ([n. d.]). https://datatracker.ietf.org/doc/html/rfc5040 Last accessed: Jan, 2021.

[2] [n. d.]. Alveo SN1000 SmartNIC Accelerator Card. ([n. d.]). https://www.xilinx.com/products/boards-and-kits/alveo/sn1000.html Last accessed: February 13, 2025.

[3] [n. d.]. Alveo U280 Data Center Accelerator Card. ([n. d.]). https://www.xilinx.com/products/boards-and-kits/alveo/u280.html February 13, 2025.

[4] [n. d.]. Amazon EC2. https://aws.amazon.com/pm/ec2. ([n. d.]). Accessed: February 13, 2025.

[5] [n. d.]. AMD OpenNIC Project. https://github.com/Xilinx/open-nic. ([n. d.]). Accessed: February 13, 2025.

[6] [n. d.]. AMDSEV. https://github.com/AMDESE/AMDSEV. ([n. d.]). Accessed: February 13, 2025.

[7] [n. d.]. Arm Confidential Compute Architecture. https://www.arm.com/why-arm/architecture/security-features/arm-confidential-compute-architecture. ([n. d.]). Last accessed: February 13, 2025.

[8] [n. d.]. AWS Lambda. https://aws.amazon.com/lambda/. ([n. d.]). Accessed: February 13, 2025.

[9] [n. d.]. AWS Nitro System. ([n. d.]). https://aws.amazon.com/ec2/nitro/ February 13, 2025.

[10] [n. d.]. Azure Functions. https://docs.microsoft.com/en-us/azure/azure-functions/functions-overview. ([n. d.]). Accessed: February 13, 2025.

[11] [n. d.]. Broadcom Stingray SmartNIC Accelerates Baidu Cloud Services. ([n. d.]). https://www.broadcom.com/company/news/product-releases/53106 February 13, 2025.

[12] [n. d.]. Build hardware for TNIC. https://github.com/TUM-DSE/TNIC-hw?tab=readme-ov-file#build-hardware-for-fpga. ([n. d.]).

[13] [n. d.]. Build software for TNIC. https://github.com/TUM-DSE/TNIC-hw?tab=readme-ov-file#build-software. ([n. d.]).

[14] [n. d.]. CockroachDB Labs: Replication layer. https://www.cockroachlabs.com/docs/stable/architecture/replication-layer.html. ([n. d.]). Last accessed: February 13, 2025.

[15] [n. d.]. Coyote: OS for FPGAs. ([n. d.]). https://github.com/fpgasystems/Coyote Last accessed: February 13, 2025.

[16] [n. d.]. Encryption and Authentication - Bootgen user Guide (UG1283). https://docs.amd.com/r/en-US/ug1283-bootgen-user-guide/Encryption-and-Authentication. ([n. d.]).

[17] [n. d.]. FoundationDB. https://apple.github.io/foundationdb/. ([n. d.]).

[18] [n. d.]. Google Compute Engine. https://cloud.google.com/. ([n. d.]). Accessed: February 13, 2025.

[19] [n. d.]. Google Functions. https://cloud.google.com/functions. ([n. d.]). Accessed: February 13, 2025.

[20] [n. d.]. HMAC. https://github.com/openssl/openssl/tree/master/crypto/hmac. ([n. d.]). Accessed: February 13, 2025.

[21] [n. d.]. How big is RocksDB adoption? https://rocksdb.org/docs/support/faq.html. ([n. d.]). Last accessed: May 2021.

[22] [n. d.]. How to run TNIC. https://github.com/TUM-DSE/TNIC-hw?tab=readme-ov-file#run. ([n. d.]).

[23] [n. d.]. InfiniBand Architecture Specification. ([n. d.]). https://www.infinibandta.org/ibta-specification/ February 13, 2025.

[24] [n. d.]. Intel DPDK. http://dpdk.org/. ([n. d.]). Last accessed: Jan, 2021.

[25] [n. d.]. Intel SGX: Not So Safe After All, ÆPIC Leak. https://thenewstack.io/intel-sgx-not-so-safe-after-all-aepic-leak/. ([n. d.]). Last accessed: February 13, 2025.

[26] [n. d.]. Intel TDX. https://www.intel.com/content/www/us/en/developer/articles/technical/intel-trust-domain-extensions.html. ([n. d.]). Last accessed: February 13, 2025.

[27] [n. d.]. Latest SGAxe and CrossTalk Attacks Leak Sensitive Data and Expose New Intel SGX Vulnerability. https://news.hackreports.com/sgaxe-crosstalk-attacks-intel-sgx-vulnerability/. ([n. d.]). Last accessed: February 13, 2025.

[28] [n. d.]. LiquidIO II Smart NICs. ([n. d.]). https://www.marvell.com/products/infrastructure-processors/liquidio-smart-nics/liquidio-ii-smart-nics.html February 13, 2025.

[29] [n. d.]. Microsoft Azure. https://azure.microsoft.com/en-gb. ([n. d.]). Accessed: February 13, 2025.

[30] [n. d.]. Netronome. ([n. d.]). https://www.netronome.com/ February 13, 2025.

[31] [n. d.]. NVIDIA BlueField Data Processing Units. ([n. d.]). https://www.nvidia.com/en-gb/networking/products/data-processing-unit/ February 13, 2025.

[32] [n. d.]. Project Catapult. ([n. d.]). https://www.microsoft.com/en-us/research/project/project-catapult/ February 13, 2025.

[33] [n. d.]. Rackspace. https://www.rackspace.com/en-gb. ([n. d.]). Accessed: February 13, 2025.

[34] [n. d.]. Secure Device Manager. https://www.intel.com/content/www/us/en/docs/programmable/683762/21-3/secure-device-manager.html. ([n. d.]).

[35] [n. d.]. The full version of the TNIC paper. https://github.com/TUM-DSE/TNIC-main/blob/main/TNIC_ASPLOS_25_full.pdf. ([n. d.]).

[36] [n. d.]. TNIC applications. https://github.com/TUM-DSE/TNIC-sw/tree/7df3d62360d5a693ce1f19bd8045ed2f0ab1b78f?tab=readme-ov-file#tnic-sw-evaluation. ([n. d.]).

[37] [n. d.]. TNIC security proofs. https://github.com/TUM-DSE/TNIC-proofs?tab=readme-ov-file#t-nic-protocol-verification. ([n. d.]).

[38] [n. d.]. Ubuntu kernel lifecycle. https://ubuntu.com/kernel/lifecycle. ([n. d.]). Last accessed: February 13, 2025.

[39] [n. d.]. UltraScale+ Integrated 100G Ethernet Subsystem. ([n. d.]). https://www.xilinx.com/products/intellectual-property/cmac_usplus.html February 13, 2025.

[40] [n. d.]. Zero-Copy Optimization for Alibaba Cloud Smart NIC Solution. ([n. d.]). https://www.alibabacloud.com/blog/zero-copy-optimization-for-alibaba-cloud-smart-nic-solution_593986 February 13, 2025.

[41] [n. d.]. ZippyDB a strongly consistent, geographically distributed key-value store at Facebook. https://engineering.fb.com/2021/08/06/core-data/zippydb/. ([n. d.]). Last accessed: February 13, 2025.

[42] [n. d.]. ÆPIC Leak is an Architectural CPU Bug Affecting 10th, 11th, and 12th Gen Intel Core CPUs. https://wccftech.com/aepic-leak-is-an-architectural-cpu-bug-affecting-10th-11th-and-12th-gen-intel-core-cpus/, note = Last accessed: February 13, 2025,. ([n. d.]).

[43] Michael Abd-El-Malek, Gregory R. Ganger, Garth R. Goodson, Michael K. Reiter, and Jay J. Wylie. 2005. Fault-scalable Byzantine fault-tolerant services. In *Symposium on Operating Systems Principles*. https://api.semanticscholar.org/CorpusID:2499938

[44] Ittai Abraham, Guy Gueta, and Dahlia Malkhi. 2018. Hot-Stuff the Linear, Optimal-Resilience, One-Message BFT Devil. *CoRR* abs/1803.05069 (2018). arXiv:1803.05069 http://arxiv.org/abs/1803.05069

[45] Ayaz Akram, Venkatesh Akella, Sean Peisert, and Jason Lowe-Power. 2022. SoK: Limitations of Confidential Computing via TEEs for High-Performance Compute Systems. In *2022 IEEE International Symposium on Secure and Private Execution Environment Design (SEED)*. 121–132. https://doi.org/10.1109/SEED55351.2022.00018

[46] Ayaz Akram, Anna Giannakou, Venkatesh Akella, Jason Lowe-Power, and Sean Peisert. 2021. Performance Analysis of Scientific Computing Workloads on General Purpose TEEs. In *2021 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. 1066–1076.

https://doi.org/10.1109/IPDPS49936.2021.00115

[47] Amazon. [n. d.]. Amazon S3 Cloud Object Storage. https://aws.amazon.com/s3. ([n. d.]). Last accessed: Dec, 2018.

[48] AMD. [n. d.]. AMD Secure Encrypted Virtualization (SEV). https://developer.amd.com/sev/. ([n. d.]). https://developer.amd.com/sev/ Last accessed: Jan, 2021.

[49] Sebastian Angel, Aditya Basu, Weidong Cui, Trent Jaeger, Stella Lau, Srinath Setty, and Sudheesh Singanamalla. 2023. Nimble: Rollback Protection for Confidential Cloud Services. In *17th USENIX Symposium on Operating Systems Design and Implementation (OSDI 23)*. USENIX Association, Boston, MA, 193–208. https://www.usenix.org/conference/osdi23/presentation/angel

[50] Mina Tahmasbi Arashloo, Alexey Lavrov, Manya Ghobadi, Jennifer Rexford, David Walker, and David Wentzlaff. 2020. Enabling Programmable Transport Protocols in High-Speed NICs. In *17th USENIX Symposium on Networked Systems Design and Implementation (NSDI 20)*. USENIX Association, Santa Clara, CA, 93–109. https://www.usenix.org/conference/nsdi20/presentation/arashloo

[51] Sergei Arnautov, Bohdan Trach, Franz Gregor, Thomas Knauth, Andre Martin, Christian Priebe, Joshua Lind, Divya Muthukumaran, Dan O'Keeffe, Mark L. Stillwell, David Goltzsche, David Eyers, Rüdiger Kapitza, Peter Pietzuch, and Christof Fetzer. 2016. SCONE: Secure Linux Containers with Intel SGX. In *Proceedings of the 12th USENIX Conference on Operating Systems Design and Implementation (OSDI'16)*. USENIX Association, USA, 689–703.

[52] ars Technica. [n. d.]. New Spectre-like attack uses speculative execution to overflow buffers. https://arstechnica.com/gadgets/2018/07/new-spectre-like-attack-uses-speculative-execution-to-overflow-buffers/. ([n. d.]). Last accessed: February 13, 2025.

[53] Pierre-Louis Aublin, Rachid Guerraoui, Nikola Knežević, Vivien Quéma, and Marko Vukolić. 2015. The Next 700 BFT Protocols. *ACM Trans. Comput. Syst.* 32, 4, Article 12 (jan 2015), 45 pages. https://doi.org/10.1145/2658994

[54] Pierre-Louis Aublin, Sonia Ben Mokhtar, and Vivien Quéma. 2013. RBFT: Redundant Byzantine Fault Tolerance. In *2013 IEEE 33rd International Conference on Distributed Computing Systems*. 297–306. https://doi.org/10.1109/ICDCS.2013.53

[55] Maurice Bailleu, Dimitra Giantsidi, Vasilis Gavrielatos, Do Le Quoc, Vijay Nagarajan, and Pramod Bhatotia. 2021. Avocado: A Secure In-Memory Distributed Storage System. In *2021 USENIX Annual Technical Conference (USENIX ATC 21)*. USENIX Association, 65–79. https://www.usenix.org/conference/atc21/presentation/bailleu

[56] Andrew Baumann, Marcus Peinado, and Galen Hunt. 2014. Shielding Applications from an Untrusted Cloud with Haven. In *Proceedings of the 11th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*.

[57] Johannes Behl, Tobias Distler, and Rüdiger Kapitza. 2017. Hybrids on Steroids: SGX-Based High Performance BFT. In *Proceedings of the Twelfth European Conference on Computer Systems (EuroSys)*.

[58] Nathan Bronson, Zach Amsden, George Cabrera, Prasad Chakka, Peter Dimov, Hui Ding, Jack Ferris, Anthony Giardullo, Sachin Kulkarni, Harry Li, Mark Marchukov, Dmitri Petrov, Lovro Puzar, Yee Jiun Song, and Venkat Venkataramani. 2013. TAO: Facebook's Distributed Data Store for the Social Graph. In *USENIX Annual Technical Conference (USENIX ATC)*.

[59] Ethan Buchman, Jae Kwon, and Zarko Milosevic. 2018. The latest gossip on BFT consensus. *CoRR* abs/1807.04938 (2018). arXiv:1807.04938 http://arxiv.org/abs/1807.04938

[60] Brendan Burns, Brian Grant, David Oppenheimer, Eric Brewer, and John Wilkes. 2016. Borg, Omega, and Kubernetes. *Commun. ACM* (2016).

[61] Mike Burrows. 2006. The Chubby lock service for loosely-coupled distributed systems. In *7th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*.

[62] Brad Calder, Ju Wang, Aaron Ogus, Niranjan Nilakantan, Arild Skjolsvold, Sam McKelvie, Yikang Xu, Shashwat Srivastav, Jiesheng Wu, Huseyin Simitci, Jaidev Haridas, Chakravarthy Uddaraju, Hemal Khatri, Andrew Edwards, Vaman Bedekar, Shane Mainali, Rafay Abbasi, Arpit Agarwal, Mian Fahim ul Haq, Muhammad Ikram ul Haq, Deepali Bhardwaj, Sowmya Dayanand, Anitha Adusumilli, Marvin McNett, Sriram Sankaran, Kavitha Manivannan, and Leonidas Rigas. 2011. Windows Azure Storage: A Highly Available Cloud Storage Service with Strong Consistency. In *Proceedings of the 23rd ACM Symposium on Operating Systems Principles (SOSP)*.

[63] Miguel Castro and Barbara Liskov. 1999. Practical Byzantine fault tolerance. In *Proceedings of the Third Symposium on Operating Systems Design and Implementation (OSDI '99)*. USENIX Association, USA, 173–186.

[64] Miguel Castro and Barbara Liskov. 2002. Practical Byzantine Fault Tolerance and Proactive Recovery. *ACM Trans. Comput. Syst.* (2002).

[65] T-H. Hubert Chan, Rafael Pass, and Elaine Shi. 2018. PaLa: A Simple Partially Synchronous Blockchain. *IACR Cryptol. ePrint Arch.* 2018 (2018), 981.

[66] T-H. Hubert Chan, Rafael Pass, and Elaine Shi. 2018. PiLi: An Extremely Simple Synchronous Blockchain. Cryptology ePrint Archive, Paper 2018/980. (2018). https://eprint.iacr.org/2018/980 https://eprint.iacr.org/2018/980.

[67] Byung-Gon Chun, Petros Maniatis, Scott Shenker, and John Kubiatowicz. 2007. Attested Append-only Memory: Making Adversaries Stick to Their Word. In *Proceedings of Twenty-first ACM SIGOPS Symposium on Operating Systems Principles (SOSP)*.

[68] Allen Clement, Flavio Junqueira, Aniket Kate, and Rodrigo Rodrigues. 2012. On the (Limited) Power of Non-Equivocation. In *Proceedings of the 2012 ACM Symposium on Principles of Distributed Computing (PODC '12)*. Association for Computing Machinery, New York, NY, USA, 301–308. https://doi.org/10.1145/2332432.2332490

[69] James C. Corbett, Jeffrey Dean, Michael Epstein, Andrew Fikes, Christopher Frost, J. J. Furman, Sanjay Ghemawat, Andrey Gubarev, Christopher Heiser, Peter Hochschild, Wilson Hsieh, Sebastian Kanthak, Eugene Kogan, Hongyi Li, Alexander Lloyd, Sergey Melnik, David Mwaura, David Nagle, Sean Quinlan, Rajesh Rao, Lindsay Rolig, Yasushi Saito, Michal Szymaniak, Christopher Taylor, Ruth Wang, and Dale Woodford. 2013. Spanner: Google's Globally Distributed Database. (2013).

[70] M. Correia, N.F. Neves, and P. Verissimo. 2004. How to tolerate half less one Byzantine nodes in practical distributed systems. In *Proceedings of the 23rd IEEE International Symposium on Reliable Distributed Systems, 2004.*

[71] Victor Costan and Srinivas Devadas. 2016. Intel SGX Explained. (2016).

[72] James Cowling, Daniel Myers, Barbara Liskov, Rodrigo Rodrigues, and Liuba Shrira. 2006. HQ replication: a hybrid quorum protocol for byzantine fault tolerance. In *Proceedings of the 7th Symposium on Operating Systems Design and Implementation (OSDI '06)*. USENIX Association, USA, 177–190.

[73] Giuseppe DeCandia, Deniz Hastorun, Madan Jampani, Gunavardhan Kakulapati, Avinash Lakshman, Alex Pilchin, Swaminathan Sivasubramanian, Peter Vosshall, and Werner Vogels. 2007. Dynamo: Amazon's Highly Available Key-Value Store. *ACM SIGOPS Operating Systems Review (SIGOPS)* (2007).

[74] Jérémie Decouchant, David Kozhaya, Vincent Rahli, and Jiangshan Yu. 2022. DAMYSUS: Streamlined BFT Consensus Leveraging Trusted Components. In *Proceedings of the Seventeenth European Conference on Computer Systems (EuroSys '22)*. Association for Computing Machinery, New York, NY, USA, 1–16.

[75] Carole Delporte-Gallet, Hugues Fauconnier, Felix Freiling, Lucia Penso, and Andreas Tielmann. 2007. From Crash-Stop to Permanent Omission: Automatic Transformation and Weakest Failure Detectors, Vol. 4731. 165–178. https://doi.org/10.1007/978-3-540-75142-7_15

[76] Cong Ding, David Chu, Evan Zhao, Xiang Li, Lorenzo Alvisi, and Robbert Van Renesse. 2020. Scalog: Seamless Reconfiguration and Total Order in a Scalable Shared Log. In *17th USENIX Symposium on Networked Systems Design and Implementation (NSDI 20)*. USENIX Association, Santa Clara, CA, 325–338. https://www.usenix.org/conference/nsdi20/presentation/ding

[77] Kailun Qin Cedric Xing Pramod Bhatotia Dmitrii Kuvaiskii, Dimitrios Stavrakakis and Mona Vij. 2024. Gramine-TDX: A Lightweight OS Kernel for Confidential VMs. In *Proceedings of the 2024 ACM SIGSAC Conference on Computer and Communications Security (CCS '24)*.

[78] Aleksandar Dragojević, Dushyanth Narayanan, Miguel Castro, and Orion Hodson. 2014. FaRM: Fast Remote Memory. In *11th USENIX Symposium on Networked Systems Design and Implementation (NSDI 14)*.

[79] Sisi Duan, Michael Reiter, and Haibin Zhang. 2018. BEAT: Asynchronous BFT Made Practical. 2028–2041. https://doi.org/10.1145/3243734.3243812

[80] Gunawi et al. 2014. What Bugs Live in the Cloud? A Study of 3000+ Issues in Cloud Systems. In *Proceedings of the ACM Symposium on Cloud Computing (SoCC)*.

[81] Shufan Fei, Zheng Yan, Wenxiu Ding, and Haomeng Xie. 2021. Security Vulnerabilities of SGX and Countermeasures: A Survey. *ACM Comput. Surv.* 54, 6, Article 126 (jul 2021), 36 pages. https://doi.org/10.1145/3456631

[82] Daniel Firestone, Andrew Putnam, Sambhrama Mundkur, Derek Chiou, Alireza Dabagh, Mike Andrewartha, Hari Angepat, Vivek Bhanu, Adrian Caulfield, Eric Chung, Harish Kumar Chandrappa, Somesh Chaturmohta, Matt Humphrey, Jack Lavier, Norman Lam, Fengfen Liu, Kalin Ovtcharov, Jitu Padhye, Gautham Popuri, Shachar Raindel, Tejas Sapre, Mark Shaw, Gabriel Silva, Madhan Sivakumar, Nisheeth Srivastava, Anshuman Verma, Qasim Zuhair, Deepak Bansal, Doug Burger, Kushagra Vaid, David A. Maltz, and Albert Greenberg. 2018. Azure Accelerated Networking: SmartNICs in the Public Cloud. In *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18)*. USENIX Association, Renton, WA, 51–66. https://www.usenix.org/conference/nsdi18/presentation/firestone

[83] Alex Forencich, Alex C. Snoeren, George Porter, and George Papen. 2020. Corundum: An Open-Source 100-Gbps Nic. In *2020 IEEE 28th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*. 38–46. https://doi.org/10.1109/FCCM48280.2020.00015

[84] Vasilis Gavrielatos, Antonis Katsarakis, and Vijay Nagarajan. 2021. Odyssey: The Impact of Modern Hardware on Strongly-Consistent Replication Protocols. In *Proceedings of the 16th European Conference on Computer Systems EuroSys'21*. Association for Computing Machinery (ACM), United States, 245–260. https://doi.org/10.1145/3447786.3456240 16th ACM EuroSys Conference on Computer Systems, EuroSys 2021 ; Conference date: 26-04-2021 Through 28-04-2021.

[85] Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung. 2003. The Google File System. In *Proceedings of the 19th ACM Symposium on Operating Systems Principles*. Bolton Landing, NY, 20–43.

[86] Dimitra Giantsidi, Maurice Bailleu, Natacha Crooks, and Pramod Bhatotia. 2022. Treaty: Secure Distributed Transactions. In *2022 52nd Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*. 14–27. https://doi.org/10.1109/DSN53405.2022.00015

[87] Stewart Grant, Anil Yelam, Maxwell Bland, and Alex C. Snoeren. 2020. SmartNIC Performance Isolation with FairNIC: Programmable Networking for the Cloud. In *Proceedings of the Annual Conference of the ACM Special Interest Group on Data Communication on the Applications, Technologies, Architectures, and Protocols for Computer Communication (SIGCOMM '20)*. Association for Computing Machinery, New York, NY, USA, 681–693. https://doi.org/10.1145/3387514.3405895

[88] Chuanxiong Guo, Haitao Wu, Zhong Deng, Gaurav Soni, Jianxi Ye, Jitu Padhye, and Marina Lipshteyn. 2016. RDMA over Commodity Ethernet at Scale. In *Proceedings of the 2016 ACM SIGCOMM Conference (SIGCOMM '16)*. 202–215.

[89] Suyash Gupta, Sajjad Rahnama, Shubham Pandey, Natacha Crooks, and Mohammad Sadoghi. 2023. Dissecting BFT Consensus: In Trusted Components We Trust!. In *Proceedings of the Eighteenth European Conference on Computer Systems (EuroSys '23)*. Association for Computing Machinery, New York, NY, USA, 521–539. https://doi.org/10.1145/3552326.3587455

[90] Andreas Haeberlen, Petr Kouznetsov, and Peter Druschel. 2007. PeerReview: Practical Accountability for Distributed Systems. In *Proceedings of Twenty-First ACM SIGOPS Symposium on Operating Systems Principles (SOSP)*.

[91] Marcus Hähnel, Weidong Cui, and Marcus Peinado. 2017. High-Resolution Side Channels for Untrusted Operating Systems. In *2017 USENIX Annual Technical Conference (USENIX ATC 17)*.

[92] Zhenhao He, Dario Korolija, and Gustavo Alonso. 2021. EasyNet: 100 Gbps Network for HLS. In *2021 31st International Conference on Field-Programmable Logic and Applications (FPL)*. 197–203. https://doi.org/10.1109/FPL53798.2021.00040

[93] Chi Ho, Danny Dolev, and Robbert Van Renesse. 2007. Making Distributed Applications Robust. 232–246. https://doi.org/10.1007/978-3-540-77096-1_17

[94] Chi Ho, Robbert Van Renesse, Mark Bickford, and Danny Dolev. 2008. Nysiad: Practical Protocol Transformation to Tolerate Byzantine Failures. In *5th USENIX Symposium on Networked Systems Design and Implementation (NSDI 08)*. USENIX Association, San Francisco, CA.

[95] Morteza Hoseinzadeh and Steven Swanson. 2021. Corundum: Statically-Enforced Persistent Memory Safety. In *Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS 2021)*. Association for Computing Machinery, New York, NY, USA, 429–442. https://doi.org/10.1145/3445814.3446710

[96] Patrick Hunt, Mahadev Konar, Flavio P. Junqueira, and Benjamin Reed. [n. d.]. ZooKeeper: Wait-free Coordination for Internet-scale Systems. In *Proceedings of the 2010 USENIX Conference on USENIX Annual Technical Conference*.

[97] Hyperledger Fabric. [n. d.]. Ordering service implementations. ([n. d.]). https://hyperledger-fabric.readthedocs.io/en/release-2.5/orderer/ordering_service.html Last accessed: February 13, 2025.

[98] Stephen Ibanez, Muhammad Shahbaz, and Nick McKeown. 2019. The Case for a Network Fast Path to the CPU. In *Proceedings of the 18th ACM Workshop on Hot Topics in Networks (HotNets '19)*. Association for Computing Machinery, New York, NY, USA, 52–59. https://doi.org/10.1145/3365609.3365851

[99] Intel Software Guard Extensions [n. d.]. Intel Software Guard Extensions (Intel SGX). https://software.intel.com/en-us/sgx. ([n. d.]). Last accessed: Jan, 2021.

[100] Zhipeng Jia and Emmett Witchel. 2021. Boki: Stateful Serverless Computing with Shared Logs. In *Proceedings of the ACM SIGOPS 28th Symposium on Operating Systems Principles (SOSP '21)*. Association for Computing Machinery, New York, NY, USA, 691–707. https://doi.org/10.1145/3477132.3483541

[101] Anuj Kalia, Michael Kaminsky, and David Andersen. 2019. Datacenter RPCs can be General and Fast. In *16th USENIX Symposium on Networked Systems Design and Implementation (NSDI)*.

[102] Anuj Kalia, Michael Kaminsky, and David G. Andersen. 2016. Design Guidelines for High Performance RDMA Systems. In *2016 USENIX Annual Technical Conference (USENIX ATC 16)*. USENIX Association, Denver, CO, 437–450. https://www.usenix.org/conference/atc16/technical-sessions/presentation/kalia

[103] Rüdiger Kapitza, Johannes Behl, Christian Cachin, Tobias Distler, Simon Kuhnle, Seyed Vahid Mohammadi, Wolfgang Schröder-Preikschat, and Klaus Stengel. 2012. CheapBFT: Resource-Efficient Byzantine Fault Tolerance. In *Proceedings of the 7th ACM European Conference on Computer Systems (EuroSys '12)*. Association for Computing Machinery, New York, NY, USA, 295–308.

https://doi.org/10.1145/2168836.2168866

[104] Alexander Kaplan and Shir Landau Feibish. 2022. Practical handling of DNS in the data plane. In *Proceedings of the Symposium on SDN Research (SOSR '22)*. Association for Computing Machinery, New York, NY, USA, 59–66. https://doi.org/10.1145/3563647.3563654

[105] Antonios Katsarakis, Vasilis Gavrielatos, M.R. Siavash Katebzadeh, Arpit Joshi, Aleksandar Dragojevic, Boris Grot, and Vijay Nagarajan. 2020. Hermes: A Fast, Fault-Tolerant and Linearizable Replication Protocol. In *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS '20)*. Association for Computing Machinery, New York, NY, USA, 201–217. https://doi.org/10.1145/3373376.3378496

[106] Antoine Kaufmann, SImon Peter, Naveen Kr. Sharma, Thomas Anderson, and Arvind Krishnamurthy. 2016. High Performance Packet Processing with FlexNIC. In *Proceedings of the Twenty-First International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS '16)*. Association for Computing Machinery, New York, NY, USA, 67–81. https://doi.org/10.1145/2872362.2872367

[107] Hsuan-Chi Kuo, Jianyan Chen, Sibin Mohan, and Tianyin Xu. 2020. Set the Configuration for the Heart of the OS: On the Practicality of Operating System Kernel Debloating. *Proc. ACM Meas. Anal. Comput. Syst.* 4, 1, Article 03 (may 2020), 27 pages. https://doi.org/10.1145/3379469

[108] Leslie Lamport. 1998. The Part-Time Parliament. *ACM Trans. Comput. Syst.* 16, 2 (may 1998), 133–169. https://doi.org/10.1145/279227.279229

[109] Leslie Lamport, Robert Shostak, and Marshall Pease. 1982. The Byzantine Generals Problem. *ACM Trans. Program. Lang. Syst.* (1982).

[110] Yanfang Le, Hyunseok Chang, Sarit Mukherjee, Limin Wang, Aditya Akella, Michael M. Swift, and T. V. Lakshman. 2017. UNO: uniflying host and smart NIC offload for flexible packet processing. In *Proceedings of the 2017 Symposium on Cloud Computing (SoCC '17)*. Association for Computing Machinery, New York, NY, USA, 506–519. https://doi.org/10.1145/3127479.3132252

[111] Giuseppe Lettieri, Alessandra Fais, Gianni Antichi, and Gregorio Procissi. 2023. SmartNIC-Accelerated Stream Processing Analytics. In *2023 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN)*. 135–140. https://doi.org/10.1109/NFV-SDN59219.2023.10329593

[112] Dave Levin, John R Douceur, Jacob R Lorch, and Thomas Moscibroda. 2009. TrInc: Small Trusted Hardware for Large Distributed Systems.. In *NSDI*, Vol. 9. 1–14.

[113] Dave Levin, John R. Douceur, Jacob R. Lorch, and Thomas Moscibroda. 2009. TrInc: Small Trusted Hardware for Large Distributed Systems. In *Proceedings of the 6th USENIX Symposium on Networked Systems Design and Implementation (NSDI)*.

[114] Bojie Li, Zhenyuan Ruan, Wencong Xiao, Yuanwei Lu, Yongqiang Xiong, Andrew Putnam, Enhong Chen, and Lintao Zhang. 2017. KV-Direct: High-Performance In-Memory Key-Value Store with Programmable NIC. In *Proceedings of the 26th Symposium on Operating Systems Principles (SOSP '17)*. Association for Computing Machinery, New York, NY, USA, 137–152. https://doi.org/10.1145/3132747.3132756

[115] Jinyuan Li, Mn Krohn, D Mazières, and Dennis Shasha. 2004. Secure untrusted data repository (SUNDR). In *Proceedings of 6th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*.

[116] Junru Li, Youyou Lu, Qing Wang, Jiazhen Lin, Zhe Yang, and Jiwu Shu. 2022. AlNiCo: SmartNIC-accelerated Contention-aware Request Scheduling for Transaction Processing. In *2022 USENIX Annual Technical Conference (USENIX ATC 22)*. USENIX Association, Carlsbad, CA, 951–966. https://www.usenix.org/conference/atc22/presentation/li-junru

[117] Jiaxin Lin, Kiran Patel, Brent E. Stephens, Anirudh Sivaraman, and Aditya Akella. 2020. PANIC: A High-Performance Programmable NIC for Multi-tenant Networks. In *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*. USENIX Association, 243–259. https://www.usenix.org/conference/osdi20/presentation/lin

[118] Junyi Liu, Aleksandar Dragojevic, Shane Flemming, Antonios Katsarakis, Dario Korolija, Igor Zablotchi, Ho-cheung Ng, Anuj Kalia, and Miguel Castro. 2023. Honeycomb: ordered key-value store acceleration on an FPGA-based SmartNIC. (03 2023).

[119] Jian Liu, Wenting Li, Ghassan O. Karame, and N. Asokan. 2016. Scalable Byzantine Consensus via Hardware-assisted Secret Sharing. *CoRR* abs/1612.04997 (2016). arXiv:1612.04997 http://arxiv.org/abs/1612.04997

[120] Ming Liu, Tianyi Cui, Henry Schuh, Arvind Krishnamurthy, Simon Peter, and Karan Gupta. 2019. Offloading Distributed Applications onto SmartNICs Using IPipe. In *Proceedings of the ACM Special Interest Group on Data Communication (SIGCOMM '19)*. Association for Computing Machinery, New York, NY, USA, 318–333. https://doi.org/10.1145/3341302.3342079

[121] Ming Liu, Simon Peter, Arvind Krishnamurthy, and Phitchaya Mangpo Phothilimthana. 2019. E3: Energy-Efficient Microservices on SmartNIC-Accelerated Servers. In *2019 USENIX Annual Technical Conference (USENIX ATC 19)*. USENIX Association, Renton, WA, 363–378. https://www.usenix.org/conference/atc19/presentation/liu-ming

[122] Youyou Lu, Jiwu Shu, Youmin Chen, and Tao Li. 2017. Octopus: an RDMA-enabled Distributed Persistent Memory File System. In *2017 USENIX Annual Technical Conference (USENIX ATC 17)*. USENIX Association, Santa Clara, CA, 773–785. https://www.usenix.org/conference/atc17/technical-sessions/presentation/lu

[123] Pieter Maene, Johannes Götzfried, Ruan de Clercq, Tilo Müller, Felix Freiling, and Ingrid Verbauwhede. 2018. Hardware-Based Trusted Computing Architectures for Isolation and Attestation. *IEEE Trans. Comput.* 67, 3 (2018), 361–374. https://doi.org/10.1109/TC.2017.2647955

[124] Francis Matus. 2020. Distributed Services Architecture. In *2020 IEEE Hot Chips 32 Symposium (HCS)*. 1–17. https://doi.org/10.1109/HCS49909.2020.9220629

[125] Simon Meier, Benedikt Schmidt, Cas Cremers, and David Basin. 2013. The TAMARIN Prover for the Symbolic Analysis of Security Protocols. In *Proceedings of the 25th International Conference on Computer Aided Verification (CAV)*.

[126] Melanox. [n. d.]. RDMA Aware Networks Programming User Manual. ([n. d.]). Last accessed: February 13, 2025.

[127] William M. Mellette, Rajdeep Das, Yibo Guo, Rob McGuinness, Alex C. Snoeren, and George Porter. 2020. Expanding across time to deliver bandwidth efficiency and low latency. In *17th USENIX Symposium on Networked Systems Design and Implementation (NSDI 20)*. USENIX Association, Santa Clara, CA, 1–18. https://www.usenix.org/conference/nsdi20/presentation/mellette

[128] Jämes Ménétrey, Christian Göttel, Anum Khurshid, Marcelo Pasin, Pascal Felber, Valerio Schiavoni, and Shahid Raza. 2022. Attestation Mechanisms For Trusted Execution Environments Demystified. In *Distributed Applications and Interoperable Systems: 22nd IFIP WG 6.1 International Conference, DAIS 2022, Held as Part of the 17th International Federated Conference on Distributed Computing Techniques, DisCoTec 2022, Lucca, Italy, June 13-17, 2022, Proceedings*. Springer-Verlag, Berlin, Heidelberg, 95–113. https://doi.org/10.1007/978-3-031-16092-9_7

[129] Alan Mislove, Ansley Post, Andreas Haeberlen, and Peter Druschel. 2006. Experiences in building and operating ePOST, a reliable peer-to-peer application. *SIGOPS Oper. Syst. Rev.* 40, 4 (apr 2006), 147–159. https://doi.org/10.1145/1218063.1217950

[130] Subhas C. Misra and Virendra C. Bhavsar. 2003. Relationships between Selected Software Measures and Latent Bug-Density: Guidelines for Improving Quality. In *Proceedings of the 2003 International Conference on Computational Science and Its Applications: PartI (ICCSA'03)*. Springer-Verlag, Berlin, Heidelberg, 724–732.

[131] Christopher Mitchell, Yifeng Geng, and Jinyang Li. 2013. Pilaf: Using One-Sided RDMA Reads to Build a Fast, CPU-Efficient Key-Value Store. *Atc '13* (2013).

[132] Rui Oliveira, José Pereira, and André Schiper. 2001. Primary-Backup Replication: From a Time-Free Protocol to a Time-Based Implementation. 14–23. https://doi.org/10.1109/RELDIS.2001.969730

[133] Diego Ongaro and John Ousterhout. 2014. In Search of an Understandable Consensus Algorithm. In *Proceedings of the 2014 USENIX Conference on USENIX Annual Technical Conference (ATC)*.

[134] Racyus D. G. Pacífico, Lucas F. S. Duarte, Luiz F. M. Vieira, Barath Raghavan, José A. M. Nacif, and Marcos A. M. Vieira. 2023. eBPFlow: A Hardware/Software Platform to Seamlessly Offload Network Functions Leveraging eBPF. *IEEE/ACM Transactions on Networking* (2023), 1–14. https://doi.org/10.1109/TNET.2023.3318251

[135] Arttu Paju, Muhammad Owais Javed, Juha Nurmi, Juha Savimäki, Brian McGillion, and Billy Bob Brumley. 2023. SoK: A Systematic Review of TEE Usage for Developing Trusted Applications. In *Proceedings of the 18th International Conference on Availability, Reliability and Security (ARES '23)*. Association for Computing Machinery, New York, NY, USA, Article 34, 15 pages. https://doi.org/10.1145/3600160.3600169

[136] Andrew Putnam, Adrian M. Caulfield, Eric S. Chung, Derek Chiou, Kypros Constantinides, John Demme, Hadi Esmaeilzadeh, Jeremy Fowers, Gopi Prashanth Gopal, Jan Gray, Michael Haselman, Scott Hauck, Stephen Heil, Amir Hormati, Joo-Young Kim, Sitaram Lanka, James Larus, Eric Peterson, Simon Pope, Aaron Smith, Jason Thong, Phillip Yi Xiao, and Doug Burger. 2014. A reconfigurable fabric for accelerating large-scale datacenter services. In *2014 ACM/IEEE 41st International Symposium on Computer Architecture (ISCA)*. 13–24. https://doi.org/10.1109/ISCA.2014.6853195

[137] Sivasankar Radhakrishnan, Yilong Geng, Vimalkumar Jeyakumar, Abdul Kabbani, George Porter, and Amin Vahdat. 2014. SENIC: Scalable NIC for End-Host Rate Limiting. In *11th USENIX Symposium on Networked Systems Design and Implementation (NSDI 14)*. USENIX Association, Seattle, WA, 475–488. https://www.usenix.org/conference/nsdi14/technical-sessions/presentation/radhakrishnan

[138] The Register. [n. d.]. Boffins show Intel's SGX can leak crypto keys . https://www.theregister.com/2017/03/07/eggheads_slip_a_note_under_intels_door_sgx_can_leak_crypto_keys/. ([n. d.]). Last accessed: February 13, 2025.

[139] Microsoft Research. [n. d.]. The Confidential Consortium Framework. https://microsoft.github.io/CCF/main/research. ([n. d.]). Last accessed: February 13, 2025.

[140] RISC-V. [n. d.]. Keystone Open-source Secure Hardware Enclave. https://keystone-enclave.org/. ([n. d.]). https://keystone-enclave.org/ Last accessed: Jan, 2021.

[141] Mario Ruiz, David Sidler, Gustavo Sutter, Gustavo Alonso, and Sergio López-Buedo. 2019. Limago: An FPGA-Based Open-Source 100 GbE TCP/IP Stack. In *2019 29th International Conference on Field Programmable Logic and Applications (FPL)*. 286–292. https://doi.org/10.1109/FPL.2019.00053

[142] Hugo Sadok, Nirav Atre, Zhipeng Zhao, Daniel S. Berger, James C. Hoe, Aurojit Panda, Justine Sherry, and Ren Wang. 2023. Enso: A Streaming Interface for NIC-Application Communication. In *17th USENIX Symposium on Operating Systems Design and Implementation (OSDI 23)*. USENIX Association, Boston, MA, 1005–1025. https://www.usenix.org/conference/osdi23/presentation/sadok

[143] Enrique Saurez, Bharath Balasubramanian, Richard Schlichting, Brendan Tschaen, Zhe Huang, Shankaranarayanan Puzhavakath Narayanan, and Umakishore Ramachandran. 2018. METRIC: A Middleware for Entry Transactional Database Clustering at the Edge. In *Proceedings of the 3rd Workshop on Middleware for Edge Clouds & Cloudlets (MECC'18)*. Association for Computing Machinery, New York, NY, USA, 2–7. https://doi.org/10.1145/3286685.3286686

[144] Henry N. Schuh, Weihao Liang, Ming Liu, Jacob Nelson, and Arvind Krishnamurthy. 2021. Xenic: SmartNIC-Accelerated Distributed Transactions. In *Proceedings of the ACM SIGOPS 28th Symposium on Operating Systems Principles (SOSP '21)*. Association for Computing Machinery, New York, NY, USA, 740–755. https://doi.org/10.1145/3477132.3483555

[145] Gianluca Scopelliti, Sepideh Pouyanrad, Job Noorman, Fritz Alder, Frank Piessens, and Jan Tobias Mühlberg. 2021. POSTER: An Open-Source Framework for Developing Heterogeneous Distributed Enclave Applications. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security (CCS '21)*. Association for Computing Machinery, New York, NY, USA, 2393–2395. https://doi.org/10.1145/3460120.3485341

[146] Yizhou Shan, Will Lin, Ryan Kosta, Arvind Krishnamurthy, and Yiying Zhang. 2022. SuperNIC: A Hardware-Based, Programmable, and Multi-Tenant SmartNIC. (2022). arXiv:cs.DC/2109.07744

[147] Shweta Shinde, Zheng Leong Chua, Viswesh Narayanan, and Prateek Saxena. 2016. Preventing page faults from telling your secrets. In *Proceedings of the 11th ACM on Asia Conference on Computer and Communications Security (ASIA CCCS)*.

[148] David Sidler, Zeke Wang, Monica Chiosa, Amit Kulkarni, and Gustavo Alonso. 2020. StRoM: Smart Remote Memory. In *Proceedings of the Fifteenth European Conference on Computer Systems (EuroSys '20)*. Association for Computing Machinery, New York, NY, USA, Article 29, 16 pages. https://doi.org/10.1145/3342195.3387519

[149] Atul Singh, Tathagata Das, Petros Maniatis, and Timothy Roscoe. 2008. BFT Protocols Under Fire. In *5th USENIX Symposium on Networked Systems Design and Implementation (NSDI 08)*. USENIX Association, San Francisco, CA. https://www.usenix.org/conference/nsdi-08/bft-protocols-under-fire

[150] Anirudh Sivaraman, Thomas Mason, Aurojit Panda, Ravi Netravali, and Sai Anirudh Kondaveeti. 2020. Network architecture in the age of programmability. *SIGCOMM Comput. Commun. Rev.* 50, 1 (mar 2020), 38–44. https://doi.org/10.1145/3390251.3390257

[151] Yee Jiun Song, Flavio Junqueira, and Benjamin Reed. 2009. BFT for the skeptics. (01 2009).

[152] J. Sousa and A. Bessani. 2012. From Byzantine Consensus to BFT State Machine Replication: A Latency-Optimal Transformation. In *2012 Ninth European Dependable Computing Conference (EDCC)*.

[153] Brent Stephens, Aditya Akella, and Michael Swift. 2019. Loom: Flexible and Efficient NIC Packet Scheduling. In *16th USENIX Symposium on Networked Systems Design and Implementation (NSDI 19)*. USENIX Association, Boston, MA, 33–46. https://www.usenix.org/conference/nsdi19/presentation/stephens

[154] Brent Stephens, Aditya Akella, and Michael M. Swift. 2018. Your Programmable NIC Should be a Programmable Switch. In *Proceedings of the 17th ACM Workshop on Hot Topics in Networks (HotNets '18)*. Association for Computing Machinery, New York, NY, USA, 36–42. https://doi.org/10.1145/3286062.3286068

[155] Guangda Sun, Mingliang Jiang, Xin Zhe Khooi, Yunfan Li, and Jialin Li. 2023. NeoBFT: Accelerating Byzantine Fault Tolerance Using Authenticated In-Network Ordering. In *Proceedings of the ACM SIGCOMM 2023 Conference (ACM SIGCOMM '23)*. Association for Computing Machinery, New York, NY, USA, 239–254. https://doi.org/10.1145/3603269.3604874

[156] Inc. Sun Microsystems. [n. d.]. NFS: Network File System Protocol Specification. https://www.ietf.org/rfc/rfc1094.txt. ([n. d.]). Last accessed: Jan, 2021.

[157] Florian Suri-Payer, Matthew Burke, Zheng Wang, Yunhao Zhang, Lorenzo Alvisi, and Natacha Crooks. 2021. Basil. In *Proceedings of the ACM SIGOPS 28th Symposium on Operating Systems Principles CD-ROM*. ACM. https://doi.org/10.1145/3477132.3483552

[158] Jörg Thalheim, Harshavardhan Unnibhavi, Christian Priebe, Pramod Bhatotia, and Peter Pietzuch. 2021. Rkt-Io: A Direct I/O Stack for Shielded Execution. In *Proceedings of the Sixteenth European Conference on Computer Systems (ACM EuroSys 21)*.

[159] Chia-Che Tsai, Donald E Porter, and Mona Vij. 2017. Graphene-SGX: A practical library OS for unmodified applications on SGX. In *Proceedings of the USENIX Annual Technical Conference (USENIX ATC)*.

[160] Robbert van Renesse, Chi Ho, and Nicolas Schiper. 2012. Byzantine Chain Replication. In *Principles of Distributed Systems*, Roberto Baldoni, Paola Flocchini, and Ravindran Binoy (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 345–359.

[161] Robbert van Renesse and Fred B. Schneider. 2004. Chain Replication for Supporting High Throughput and Availability. In *Proceedings of the 6th Conference on Symposium on Operating Systems Design & Implementation - Volume 6 (OSDI)*.

[162] Giuliana Veronese, Miguel Correia, Alysson Bessani, Lau Lung, and Paulo Veríssimo. 2013. Efficient Byzantine Fault-Tolerance. *Computers, IEEE Transactions on* 62 (01 2013), 16–30. https://doi.org/10.1109/TC.2011.221

[163] Guozhang Wang, Joel Koshy, Sriram Subramanian, Kartik Paramasivam, Mammad Zadeh, Neha Narkhede, Jun Rao, Jay Kreps, and Joe Stein. 2015. Building a Replicated Logging System with Apache Kafka. *Proc. VLDB Endow.* 8, 12 (Aug. 2015), 1654–1655. https://doi.org/10.14778/2824032.2824063

[164] Tao Wang, Xiangrui Yang, Gianni Antichi, Anirudh Sivaraman, and Aurojit Panda. 2022. Isolation Mechanisms for High-Speed Packet-Processing Pipelines. In *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)*. USENIX Association, Renton, WA, 1289–1305. https://www.usenix.org/conference/nsdi22/presentation/wang-tao

[165] Zeke Wang, Hongjing Huang, Jie Zhang, Fei Wu, and Gustavo Alonso. 2022. FpgaNIC: An FPGA-based Versatile 100Gb SmartNIC for GPUs. In *2022 USENIX Annual Technical Conference (USENIX ATC 22)*. USENIX Association, Carlsbad, CA, 967–986. https://www.usenix.org/conference/atc22/presentation/wang-zeke

[166] Ofir Weisse, Valeria Bertacco, and Todd Austin. 2017. Regaining Lost Cycles with HotCalls: A Fast Interface for SGX Secure Enclaves. *SIGARCH Comput. Archit. News* (2017).

[167] Ofir Weisse, Valeria Bertacco, and Todd Austin. 2017. Regaining Lost Cycles with HotCalls: A Fast Interface for SGX Secure Enclaves. In *Proceedings of the 44th Annual International Symposium on Computer Architecture (ISCA '17)*. Association for Computing Machinery, New York, NY, USA, 81–93. https://doi.org/10.1145/3079856.3080208

[168] AMD Xilinx. [n. d.]. AMD Vitis HLS. https://www.amd.com/de/products/software/adaptive-socs-and-fpgas/vitis/vitis-hls.html. ([n. d.]). Last accessed: February 13, 2025.

[169] AMD Xilinx. [n. d.]. Vivado ML Edition. https://www.xilinx.com/products/design-tools/vivado.html. ([n. d.]). Last accessed: February 13, 2025.

[170] Jinli Yan, Lu Tang, Junnan Li, Xiangrui Yang, Wei Quan, Hongyi Chen, and Zhigang Sun. 2019. UniSec: a unified security framework with SmartNIC acceleration in public cloud. In *Proceedings of the ACM Turing Celebration Conference - China (ACM TURC '19)*. Association for Computing Machinery, New York, NY, USA, Article 9, 6 pages. https://doi.org/10.1145/3321408.3323087

[171] Junfeng Yang, Can Sar, and Dawson Engler. 2006. EXPLODE: a lightweight, general system for finding serious storage system errors. In *Proceedings of the 7th USENIX Symposium on Operating Systems Design and Implementation - Volume 7 (OSDI '06)*. USENIX Association, USA, 10.

[172] Junfeng Yang, Paul Twohey, Dawson Engler, and Madanlal Musuvathi. 2006. Using model checking to find serious file system errors. *ACM Trans. Comput. Syst.* 24, 4 (nov 2006), 393–423. https://doi.org/10.1145/1189256.1189259

[173] Myoungsung You, Jaehyun Nam, Minjae Seo, and Seungwon Shin. 2023. HELIOS: Hardware-assisted High-performance Security Extension for Cloud Networking. In *Proceedings of the 2023 ACM Symposium on Cloud Computing (SoCC '23)*. Association for Computing Machinery, New York, NY, USA, 486–501. https://doi.org/10.1145/3620678.3624786

[174] Mark Zhao, Mingyu Gao, and Christos Kozyrakis. 2022. ShEF: Shielded Enclaves for Cloud FPGAs. In *Proceedings of the 27th ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS '22)*. Association for Computing Machinery, New York, NY, USA, 1070–1085. https://doi.org/10.1145/3503222.3507733