

FPGA-Accelerated Cloud Security Modeling Using a SmartNIC-Based VPN Forwarding Architecture

Abdulrahman Sharqawi and Mohamed Elrabaa*

Cite <https://doi.org/10.64589/juri/231351>

Submitted: November 27, 2025 Revised: March 11, 2026 Accepted: April 9, 2026

ABSTRACT

In this study, a Smart Network Interface Card (SmartNIC)-based architecture that uses a field-programmable gate array (FPGA) to secure cloud traffic before it enters a provider's infrastructure was explored. The growing reliance on cloud computing has introduced significant privacy concerns, as cloud administrators may retain access to sensitive tenant data despite existing access controls. To address this, a software-defined model was developed to evaluate how the system establishes keys, encrypts data, and forwards packets of different network sizes. The architecture offloads encryption, encapsulation, and forwarding to an externally managed FPGA accelerator, keeping user data outside the administrative domain of the cloud provider simulation. The results demonstrate predictable key negotiations, stable encrypted-forwarding, and strong scalability across all test scenarios. The findings of this study suggest that FPGA-accelerated SmartNICs can offer an effective and practical path for improving cloud-data privacy without relying on computationally heavy cryptographic techniques.

Keywords: hardware acceleration, FPGA, virtual private network, smart network interface card, cloud security, emulation

1. INTRODUCTION

Cloud Computing is the foundation of today's digital infrastructure. According to global statistics worldwide, over 60% of company data are currently stored on cloud network, with this value continually increasing¹. Despite its benefits, the cloud environment poses a serious security risk because customers must blindly trust cloud service providers, including cloud administrators. Nevertheless, some cloud administrators can view client data owing to the underlying architecture, despite stringent regulations and access controls². This has led to a fundamental conflict between the critical need for security and privacy by cloud tenants and the usability of cloud services.

Organizations are increasingly exploring homomorphic encryption as an alternative for safely processing sensitive information on the cloud without exposing the underlying data³. Although this approach offers strong privacy guarantees, fully homomorphic encryption remains highly impractical for current implementations that are up to ten times slower than standard computations⁴, making real-world deployment costly and slow. Additionally, most homomorphic techniques support only a limited set of operations, which restricts their usefulness in general cloud workloads⁵.

This paper presents a hardware-assisted method for protecting cloud data before they cross the provider's trust boundary in response to these constraints. The proposed architecture uses a third-party FPGA built into a SmartNIC (sNIC) to encapsulate and encrypt user communication. Additional security from

cloud supervisors is provided by offloading encryption, encapsulation, and forwarding to an externally managed accelerator. Previous studies have supported the viability of sNIC offloading as a security solution by demonstrating that it can increase packet throughput and reduce Central Processing Unit (CPU) utilization by up to 40%⁶. Strong confidentiality is provided without the high computational cost of homomorphic encryption owing to this architecture, which uses an FPGA-based sNIC as an autonomous security gateway that performs high-speed virtual private network (VPN)-style tunneling at the network interface.

The main goal of this study was to design and evaluate a hardware-accelerated VPN forwarding architecture that enhances data privacy in cloud environments. The proposed system creates a flexible security pipeline that can encrypt and forward user traffic at the line rate by utilizing the reconfigurability of FPGAs on the sNIC. The hardware benefits of this approach and its potential as a scalable alternative for businesses that require strict cloud anonymity were also examined.

As shown in Figure 1, the hardware-accelerated data-processing pipeline envisioned by the proposed architecture is represented by a sNIC path. Instead of passing through the host CPU or interacting with the system software, incoming client traffic is intercepted directly by the sNIC, where a bump-in-the-wire FPGA performs packet parsing, encapsulation, and encryption transparently on the data path using onboard dynamic random-access memory (DRAM). This route simulates how an actual FPGA-based SmartNIC offloads VPN-style forwarding at

Regular NIC + Bump-in-the-Wire FPGA = SmartNIC

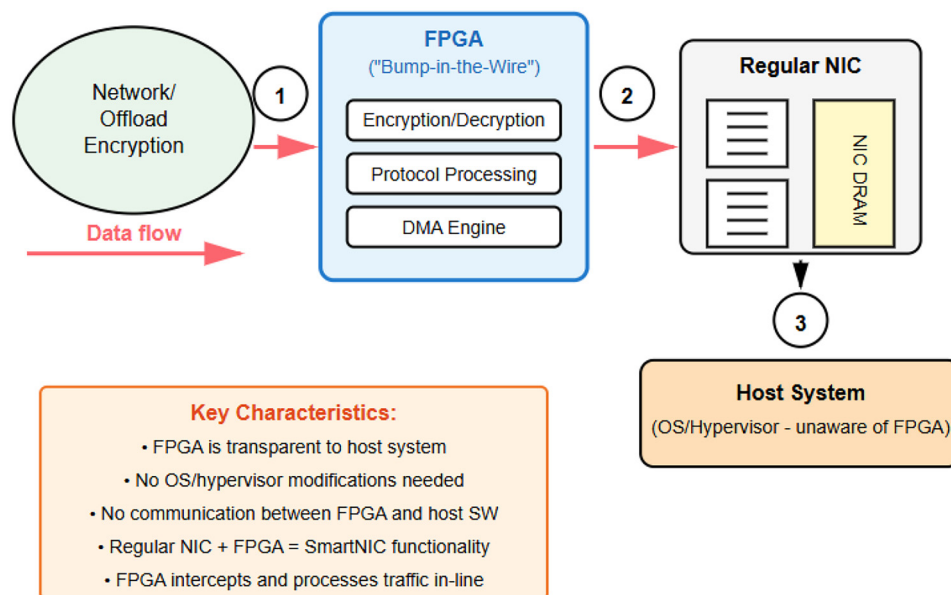


Figure 1. On-path sNIC with an FPGA bump in the wire

line rate, lowering the CPU burden and providing user traffic with a safe, dedicated border before it enters the cloud environment.

As shown in Figure 2, the FPGA is in a bump-in-the-wire configuration, where the NIC and Ethernet physical layer (PHY) perform oblivious security functions in the data plane. The host system SW and NIC handle all networking functions, PHY handles all signaling and data framing, and the FPGA adds a security and privacy layer on top. The private keys and authentication are managed by a 3rd party trusted authority (TA), from which cloud

customers obtain credentials and FPGA authentication to deploy security functions, such as VPN, among their virtual machines (VMs).

2. METHODOLOGY

2.1. System Components and Software Setup. This study was based on a hardware-assisted security architecture that

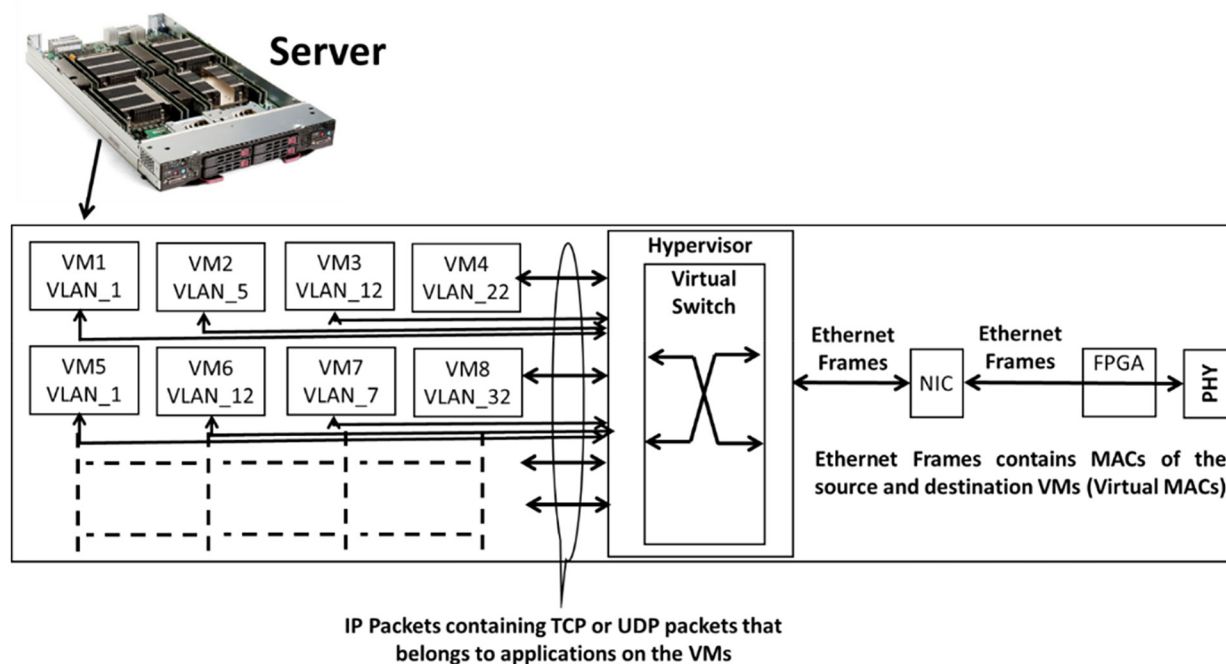


Figure 2. sNIC-enabled FPGA server used in the proposed architecture

provides external encryption and forwarding capabilities by integrating a third-party FPGA into the SmartNIC path. Located between the client and the network boundary of the cloud provider, the SmartNIC functions as a standalone security device. The SmartNIC is completely within user control and remains outside the administrative domain of the cloud provider, guaranteeing that all communications are encapsulated and encrypted within the cloud infrastructure.

2.1.1. Hardware components. A server-grade computer equipped with a peripheral component interconnect express (PCIe)-based FPGA SmartNIC served as the basis for this project. Reconfigurable logic blocks, digital signal processing units, onboard double data rate memory, and two high-speed network connections are the main features of the SmartNIC. With the help of these parts, we can create a unique hardware pipeline that functions similarly to a VPN tunnel by encrypting, encapsulating, and forwarding packets at full-line speed. The packet parser, advanced encryption standard encryption unit, header-encapsulation block, and forwarding stage are simple modules that comprise the FPGA architecture. Depending on the FPGA model being utilized, typical vendor tools, such as Xilinx Vivado or Intel Quartus, can be used to write, build, and program all hardware logic onto the SmartNIC.

2.1.2. Software environment. Code Source Implementation: <https://github.com/Abdulrahman-Sharqawi/Modeling-an-FPGA-Accelerated-Cloud-Security-Using-a-SmartNIC-Based-VPN-Forwarding-Architecture> The software used in this study is based on a simple Python framework that mimics the operation of a secure VPN-style forwarding system over several cloud nodes. All necessary libraries were obtained directly from the requirements file of the project, and everything was operated on a Linux system with Python 3 installed.

Most of the core logic of the system is split across three main Python files in the project folder. The first file, `vpn_common.py`, handles all encryption work. This includes generating keys using Diffie-Hellman, deriving session keys using hash-based key derivation function (HKDF), and encrypting or decrypting messages using a simple SHA256-based stream cipher. It also contains utilities for packing messages into a standard format before sending them to other nodes.

The second file, `vpn_node.py`, acts as the “brain” of each virtual VPN node. Each node obtains its own unique ID, media access control (MAC) address, and virtual internet protocol (IP). Nodes talk to each other over user datagram protocol (UDP) sockets, automatically negotiate keys when required, update content addressable memory (CAM) entries, encrypt outgoing messages, and decrypt all received data. This file handles both control messages (e.g., key exchanges) and encrypted data.

Finally, `vpn_stress_test.py` is used to check the system behavior under heavy traffic conditions. It creates many virtual nodes and allows them to communicate randomly to see how quickly keys can be established and how many encrypted messages the system can process over time.

A small browser-based visualizer was included in the project that allows users to add virtual nodes, send messages between them, and observe how links and virtual local area networks (VLANs) are created on a graph. This makes it easier to confirm

visually that the forwarding and key negotiations work correctly. Table 1 summarizes the complete system components and software setup used.

2.2. Test Prototypes and Parameters. The experimental program used in this study was based on a set of software-defined VPN node prototypes that simulate secure connections between cloud-hosted virtual machines. As illustrated in Figure 3, each prototype is a virtual node that runs forwarding logic, a key exchange mechanism, and an encrypted communication pipeline. The prototypes were configured to function with different network characteristics to evaluate the behavior of the system under different forwarding scenarios.

The three main components of any VPN node are a virtual IP address, a simulated MAC address, and a unique virtual machine identifier (UUID). Each pair of UDP-connected nodes is associated with a VLAN tag that functions as a tunnel identifier. The prototypes had different assigned names, ports, VLANs, and peer settings. The system can now be tested in different CAM states and flow identifiers owing to this update.

The testing parameters, summarized in Figure 4, included the following:

- **Communication Type:** Encrypted UDP packets exchanged between node pairs.
- **Key-Establishment Load:** Automatically triggered CAM misses using Diffie-Hellman (DH) + HKDF.
- **Traffic Style:** Both steady messaging (monotonic) and randomized message patterns (cyclic) were used.
- **Boundary Conditions:** All nodes run on the same host machine to ensure consistent timing and eliminate external latency factors.
- **Measurement Methods:** Console logs were used to record CAM hits/misses, key establishment events, encrypted packet transfers, and end-to-end message throughputs.

Figure 5 illustrates the flow matrix between node pairs, showing two controlled point-to-point configurations across VLAN 100 and VLAN 200, alongside a stress-test group comprising 50–100 nodes under random cyclic traffic conditions.

Several prototype groups were tested to assess their performance across various configurations. Each group differed according to the message pattern, VLAN tag, or peer address. The test matrix used in this study are summarized in Table 2.

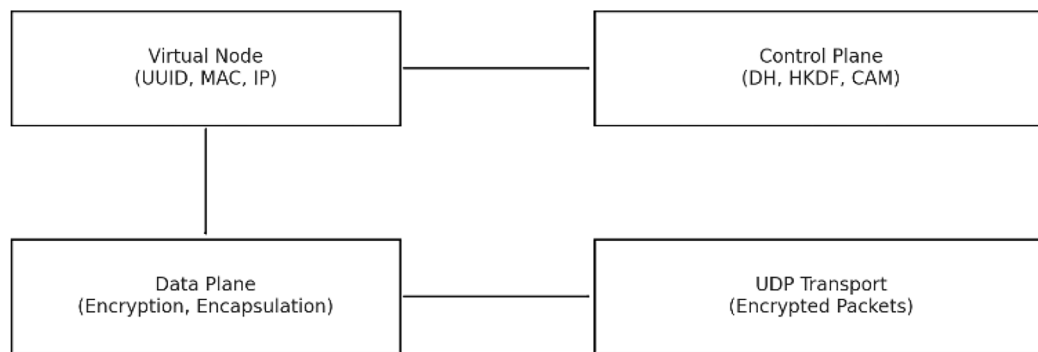
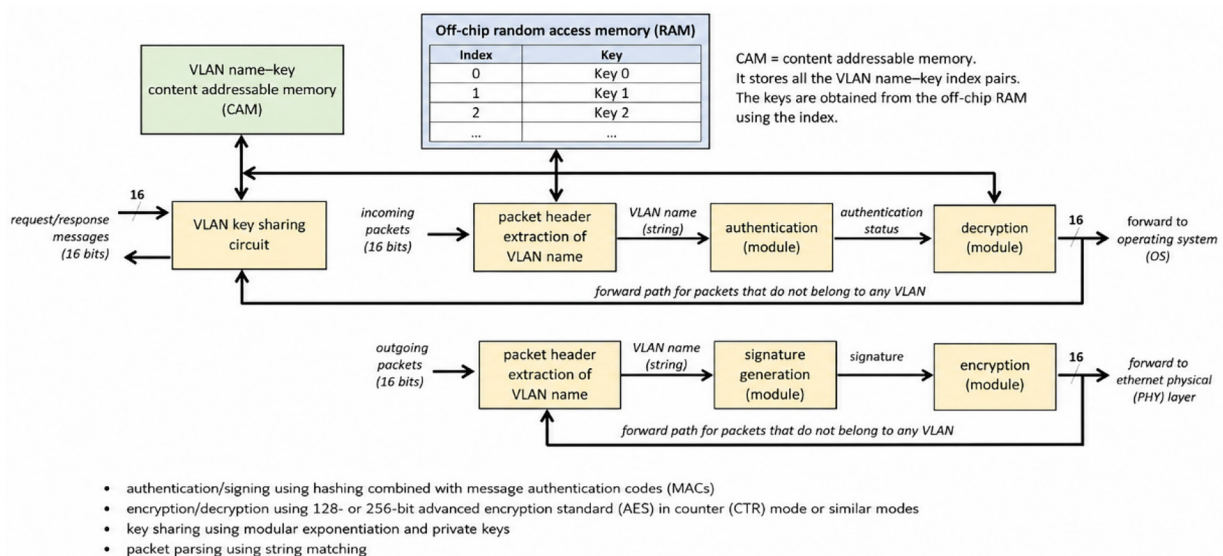
The prototype configurations used to assess the system behavior under various traffic and network situations are listed in Table 2. A baseline for monotonic message flows was provided using fixed ports, specified peer addresses, and VLAN tags in the first four configurations, which were controlled point-to-point tests. To simulate larger and more dynamic environments, the remaining entries expand the system to 10, 50, and 100 nodes utilizing randomized ports and peer mappings. To evaluate scalability, forwarding stability, and performance under increasing loads, these stress tests produced cyclic or extremely variable traffic patterns.

3. EXPERIMENTAL WORK

3.1. Test Setup. A software-defined test environment that mimics the forwarding and encryption of traffic between cloud

Table 1. System components and software setup

Component	Description
Host System	Linux machine running Python 3
Main Program Files	vpn_node.py, vpn_common.py, vpn_stress_test.py
Crypto	Diffie–Hellman (DH) key exchange, HKDF (SHA256), simple SHA256-CTR stream cipher
Message Structure	JSON header + encrypted binary payload
Node Identity	UUID-based VM ID, auto-generated MAC address, virtual IP
Forwarding Features	CAM lookup, flow ID mapping, automatic key setup
Transport Method	UDP sockets for communication between nodes
Testing Tool	vpn_stress_test.py for large-scale message testing
Visualizer	HTML-based node and VLAN visualizer (index.html)
Dependencies	Packages from requirements.txt (numpy, flask, tenseal, etc.)
How it Runs	<code>python vpn_node.py -name A -listen host:port -peer host:port -vlan 100</code>
Testing Conditions	Same VLAN tag, same crypto settings, same network setup across all tests

**Figure 3.** Virtual node structure using the built-in browser visualizer**Figure 4.** Detailed schematic of the prototypes using the built-in browser visualizer

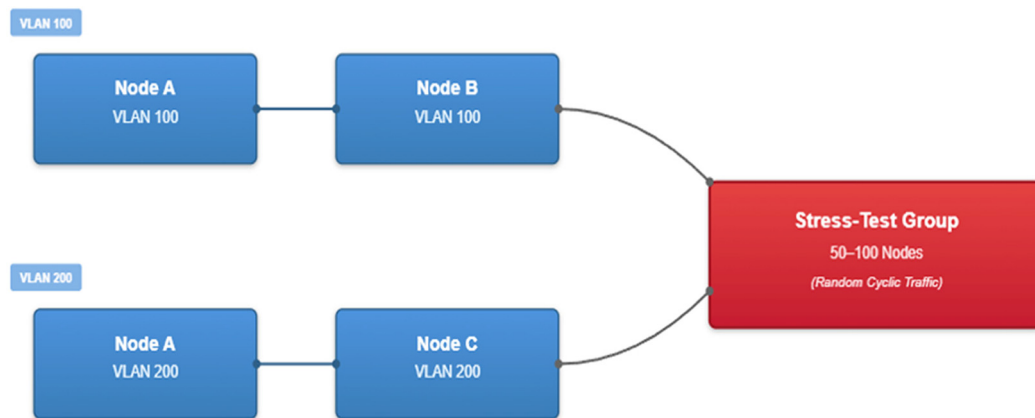


Figure 5. Flow matrix between node pairs

virtual machines via an FPGA-based SmartNIC was used in this study. The forwarding logic, key-exchange mechanism, and encrypted message pipeline were developed as separate Python processes for each "VPN node" in the system.

Each node functions as a VM using its own UUID, virtual MAC address, and VLAN-scoped tunnel identity. The system automatically updates the CAM entries, establishes keys using Diffie-Hellman and HKDF when two nodes communicate, and encrypts all messages using a unique SHA256-CTR stream cipher from the `vpn_common.py` module. The timing, cryptographic settings, and forwarding behavior were guaranteed to remain consistent throughout each test by running all nodes on the same system.

3.2. Data Logging and Monitoring Tools. This subsection describes the tools used to collect and track data during the experiments. The study was entirely software-defined, so all measurements were gathered through the system's built-in logging and visualization facilities rather than physical sensors. For each test run, console logs recorded CAM hits and misses, key-establishment events, encrypted packet transfers, and end-to-end message throughput. The browser-based visualizer presented a live display of node creation, VLAN-scoped links, and active forwarding paths.

Figure 6(a) shows the initial schematic layout of the virtual nodes generated by the visualizer, including their assigned UUIDs, MAC addresses, and virtual IPs. Figure 6(b) shows the visualizer after message exchange, highlighting the VLAN-scoped communication links and active forwarding paths drawn

on the SVG canvas. Figure 6(c) shows the event-log panel, which records the CAM misses, key-establishment events, and forwarding actions used to monitor the system behavior during testing.

4. RESULTS AND DISCUSSION

4.1. Experimental Outcome Overview. The tests showed a clear pattern across all prototype groups. In the simple point-to-point setups, every run followed the same sequence: the first packet triggered a CAM miss, the nodes exchanged Diffie-Hellman public keys, and a session key was derived via HKDF. Subsequently, all messages flowed through the CAM hits, meaning that the nodes relied on the already established flow state without repeating the negotiation. This behavior was stable across all repeated trials.

When scaling up to 10, 50, and 100 nodes, the system produced the same behavior, but at a larger scale. More nodes implied more initial CAM misses and more key-establishment events, especially when the communication was randomized. As shown in Figure 7, even under these conditions, once keys were established, the forwarding pipeline remained steady: encrypted messages were routed correctly and the visualizer confirmed that the VLAN links and forwarding paths remained consistent as the traffic increased.

One noticeable pattern is that heavy randomized traffic causes multiple nodes to attempt to establish keys simultaneously. These overlapping negotiations did not break the system, but briefly increased the control message activity before returning

Table 2. Test Matrix

Sl. No.	Prototype ID	Node Name	Listen Port	Peer Address	VLAN Tag	Traffic Pattern
1	A-5001-V100	Node A	5001	127.0.0.1:5002	100	Monotonic
2	B-5002-V100	Node B	5002	127.0.0.1:5001	100	Monotonic
3	A-5001-V200	Node A	5001	127.0.0.1:5003	200	Monotonic
4	C-5003-V200	Node C	5003	127.0.0.1:5001	200	Monotonic
5	Stress-10Nodes	Node-1...Node-10	Random	Random	100	Cyclic (Randomized)
6	Stress-50Nodes	Node-1...Node-50	Random	Random	100	Cyclic (High Load)
7	Stress-100Nodes	Node-1...Node-100	Random	Random	100	Cyclic (Extreme Load)

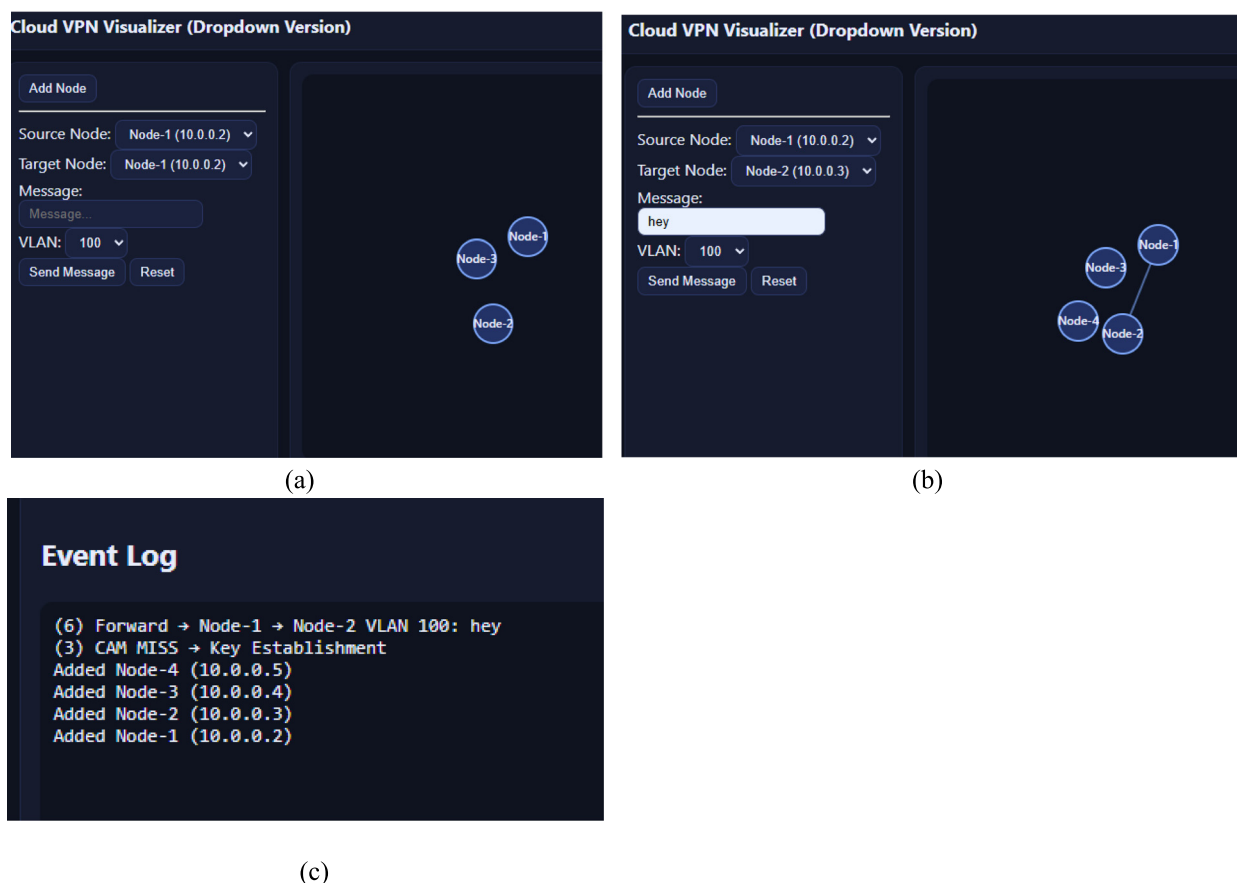


Figure 6. Visualization and logging tools used for VPN node activity monitoring

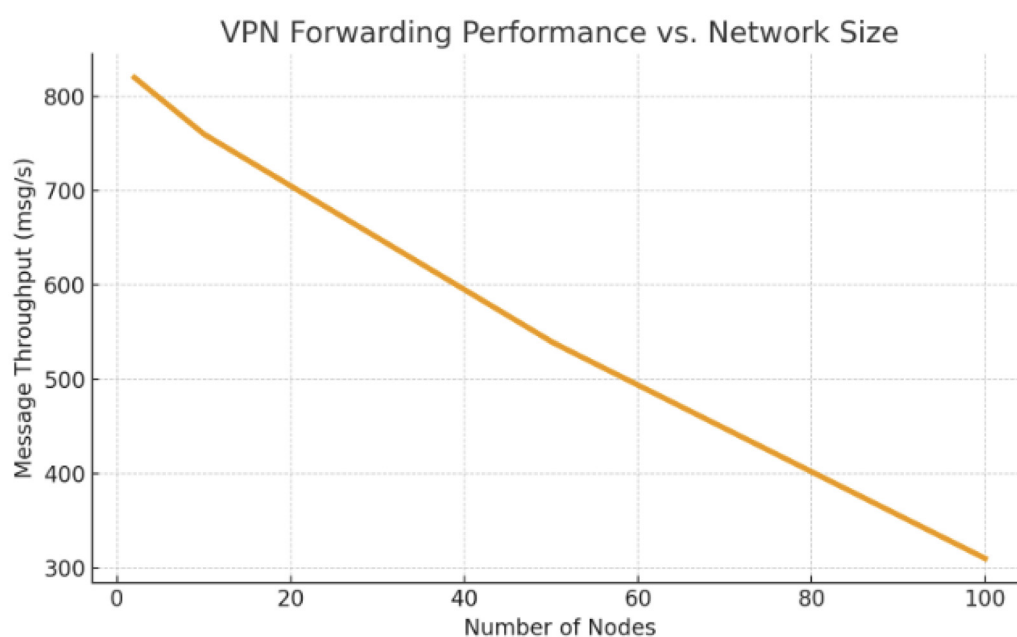


Figure 7. VPN forwarding performance vs. network size

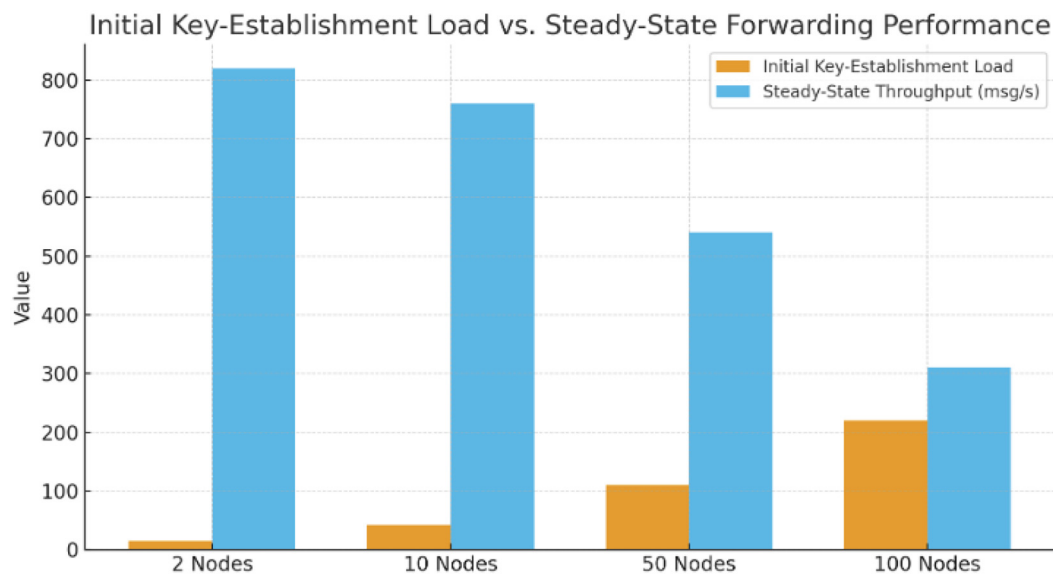


Figure 8. Initial key-establishment load and steady-state forwarding performance

to normal encrypted forwarding. Overall, the results show that the forwarding logic, key-exchange mechanism, and CAM-based flow identification worked as intended across both controlled and high-load environments.

4.1.1. Initial observations. At the start of each experiment, the system consistently followed the same baseline behavior. When two nodes communicate for the first time, the forwarding path begins with a CAM miss. This triggers an immediate Diffie–Hellman key exchange, followed by HKDF-derived session-key generation. Once the key is established, the CAM entry is populated, and the system shifts to steady encrypted forwarding. This pattern was identical across all prototype groups, confirming that the initial CAM and key-exchange logic behaved predictably and matched the expected design. These early results confirm that the forwarding pipeline, key negotiation steps, and message framing functioned correctly before being scaled to larger traffic loads.

4.1.2. Comparison with existing research. After establishing the basic behavioral pattern, the results were compared with the findings from recent SmartNIC-based and software-defined VPN studies. Previous studies have shown that offloaded security functions typically reduce control-plane overhead after the initial key negotiation phase, with message forwarding stabilizing once the flow entries are cached. Our observations were consistent with these findings: once a session key was established, the system relied almost entirely on CAM hits, and encrypted forwarding proceeded without repeated negotiation matching behavior documented in modern programmable data-plane systems⁷.

A notable difference from expectations was the frequency of simultaneous key-establishment events during the larger randomized tests. Whereas previous studies often report smooth scaling under controlled topologies, our fully randomized node-pair communication patterns created bursts of overlapping negotiations. This did not disrupt forwarding, but raised the short-term key rate compared to typical SmartNIC implementations.

The increased negotiation traffic is likely due to the simplified software-only simulation lacking the hardware-based scheduling and microsequencing found in real SmartNICs⁸.

Overall, our results align well with those of previous studies on programmable network offloading while also highlighting behaviors unique to highly dynamic cloud-like workloads. Figure 8 summarizes the initial negotiation load and steady-state forwarding performance observed in the experiments.

5. CONCLUSIONS

This study explored a software-defined prototype of an FPGA-assisted SmartNIC architecture designed to provide secure, encrypted VPN-style forwarding at the edge of cloud environments. The goal was to understand how the system behaves during key establishment, how quickly it transitions to steady-state encrypted forwarding, and how the forwarding performance scales as the number of virtual nodes increases.

Based on the results, the main findings of this study can be summarized as follows:

- The system behaved consistently across all test configurations: every communication started with a CAM miss, triggered Diffie–Hellman key exchange, and then moved into stable encrypted forwarding once a session key was established.
- The steady-state behavior was reliable across both the small and large test groups, with most packets forwarded through CAM hits rather than repeated control operations.
- As the network size increased, the number of initial key-establishment events increased, as expected; however, this did not interrupt forwarding or cause instability, showing that the architecture could tolerate bursty, randomized traffic patterns.
- The scaling trend of the system aligns with observations from recent SmartNIC and programmable data plane research, while also revealing higher negotiation bursts under randomized communication, which would likely be smoothed out in hardware implementations.

- The built-in visualizer and event log confirm the correct forwarding paths, key negotiation sequences, and VLAN-scoped communication, validating the prototype pipeline design.

Overall, the results demonstrate that a SmartNIC-based security gateway, whether simulated in software or implemented on an FPGA, can predictably establish secure tunnels and forward-encrypted traffic with minimal overhead once the initial keys are exchanged. The findings of this study provide a foundation for future refinement, including hardware integration, multi-hop forwarding, rate limiting, and performance benchmarking, on real FPGA SmartNIC platforms.

AFFILIATIONS AND AUTHOR DETAILS

Undergraduate Author

Abdulrahman Sharqawi – Department of Computer engineering, King Fahd University of Petroleum & Minerals, Dhahran 31261, Saudi Arabia;  0009-0004-5161-0869
Email: s202182610@kfupm.edu.sa

Corresponding Author

Mohamed Elrabaa – Research Mentor, Professor, Department of Computer Engineering, King Fahd University of Petroleum & Minerals, Dhahran 31261, Saudi Arabia;  0000-0002-4643-0853
Email: elrabaa@kfupm.edu.sa

ACKNOWLEDGEMENTS

The authors gratefully acknowledge the Department of Computer Engineering at King Fahd University of Petroleum and Minerals for providing the facilities and funding that made this project possible.

REFERENCES

- (1) *Worldwide IDC Global DataSphere Forecast, 2025–2029*. IDC Research (2025).
- (2) Gholami, A. & Laure, E. Security and Privacy of Sensitive Data in Cloud Computing: A Survey of Recent Developments (2016).
- (3) Acar, T., Aksu, H., Akgun, M.U. & Uluagac, A.S. A Survey on Homomorphic Encryption: Applications, Security, and Challenges. *ACM Comput. Surv.* (2021).
- (4) Gentry, C. Fully homomorphic encryption using ideal lattices. *STOC* (2009).
- (5) Halevi, S. & Shoup, V. Algorithms in Homomorphic Encryption. *IBM Research* (2014).
- (6) Firestone, D. et al. Azure SmartNIC: Offloading network security to programmable hardware. *IEEE Micro* (2020).
- (7) Elizalde, S., Mora-Millan, L., et al. A Survey on Security Applications with SmartNICs. *J. Network and Computer Applications* (2025).
- (8) Hauser, F., Häberle, M., Merling, D., Lindner, S., Gurevich, V., Zeiger, F., Frank, R., & Menth, M. *A Survey on Data Plane Programming with P4: Fundamentals, Advances, and Applied Research* (2021).