A Unified Framework for High-Speed, Secure SDN: A Data Plane Approach

Seyyed Reza Tabatabaei Manesh[1] , Abbas Manavi Nezhad[2]*

1.PhD, Department of Strategic Management, National Defense University, Tehran, Iran.
2. MA, Department of Electrical Engineering, Aalto University, Espoo, Finland (Corresponding author).

*** Corresponding author email address**: abbas.manavi.n@gmail.com

Received: 2024-07-11 **Reviewed:** 2024-08-25 **Revised:** 2024-09-12 **Accepted:** 2024-09-29 **Published:** 2025-01-28

Abstract

The paper presents a scalable, software-centric architecture for secure, high-performance networking in the SDN environment. Our approach merges robust security with near line-rate throughput by integrating high-speed packet processing capabilities with optimized cryptographic operations into one cohesive SDN framework. At the heart of the architecture is the Data Plane Development Kit, which, through user-space processing, zero-copy buffering, advanced memory management, introduces low latency with reduced intruptions for packet handling. It integrates IPsec in such a way as to provide data confidentiality and integrity at the IP layer. The architecture takes advantage of vector packet processing to flexibly manipulate packets, adapt routing decisions on the fly, and make changes according to evolving network requirements. This forms one cohesive system that ties security with speed, giving operators agility to scale services, enforce policies, and protect sensitive data with software-driven efficiencies and minimal reliance on specialty hardware.

Keywords: *Security Improvement, High Throughput, Software Defined Network (SDN), Data Plane Development Kit (DPDK), Vector Packet Processing (VPP).*

How to cite this article:

Tabatabaei Manesh R , Manavi Nezhad A. (2024). A Unified Framework for High-Speed, Secure SDN: A Data Plane Approach. Management Strategies and Engineering Sciences, 6(5), 138-151.

1. Introduction

SDN[*] provides a paradigm shift in managing the network by separating the control plane from the data plane. This can enable the programmability of networks in a centralized and dynamic manner. Traditional architectures have often led to inflexibility, vendor lock-in, and the inability of rapid adaptation against evolving requirements. In contrast, the abstraction of the control plane in SDN offers a holistic view of the network while automating tasks that include policy enforcement, traffic engineering, and orchestrated automation. That essential redesign not only operationalizes the simplicity of networks but also seamlessly integrates new services and technologies.

Despite these advantages, the SDN environment brings in different kinds of security challenges. Unauthorized access to the centralized SDN controller or interception of control messages would risk network policies and data integrity and potentially enable malicious activities [1, 2]. As networks continue to evolve and scale, robust security measures must be seamlessly integrated without degrading performance.

IPSec[†] is a widely adopted suite of protocols that provides confidentiality, integrity, and authentication at the IP layer. Its integration into SDN can enhance the security of the control and data channels in a way that only authorized parties can modify or access critical state information [3, 4]. However, most traditional IPsec gateways rely on the kernel-based Linux protocol stack, entailing multiple data copies and synchronization overhead. This leads to latency and throughput penalties-unacceptable in today's high-speed networking scenarios [5].

These performance bottlenecks are taken care of by the Data Plane Development Kit (DPDK), which processes the packets completely in user space and bypasses the kernel altogether. DPDK achieves faster packet processing by exploiting NUMA[‡]-awareness, huge pages, zero-copy optimizations, and hardware offloading. By accelerating packet handling, DPDK reduces the performance costs caused by IPsec overheads [6]. This synergy of SDN, IPsec, and DPDK forms a secure high-throughput networking foundation.

To further enhance performance and flexibility, Vector Packet Processor (VPP) can be integrated as a modular dataplane engine. This realizes advanced packet manipulation, routing, and switching functionalities without sacrificing the line-rate performance. The SDN, IPsec, DPDK, and VPP together provide the full, secure, effective environment for modern network operation.

2. Literature Review

Software-defined networking (SDN) has revolutionized network management, offering unprecedented flexibility and programmability. However, achieving optimal performance and robust security in SDN environments is a focal point for research and development.

Several studies have addressed the performance limitations of older SDN architectures. One of the areas of focus in SDN research is increasing network speed through the integration of specialized data plane processing techniques. For example, the use of the Data Plane Development Kit (DPDK) has attracted considerable attention due to its ability to speed up packet processing by bypassing the operating system kernel and executing data plane tasks directly in user space. Several studies have shown the effectiveness of DPDK in improving packet processing efficiency and reducing latency in SDN environments. [7, 8] Also, in [9], the authors investigated the effect of using programmable data planes to speed up packet processing, and in [10], the authors investigated the improvement of packet processing efficiency by using hardware techniques such as SmartNIC.

In other researches, load balancing algorithms suitable for SDN environments have also been investigated. In [11-13], machine learning-based approaches for load balancing and optimization of packet processing and routing based on various network criteria and application requirements have been reviewed. Their findings show improved network utilization and reduced congestion compared to traditional load balancing methods.

A result of this research shows that focusing only on one part of the entire packet processing process, such as network controls or traffic load balancing, does not have the ability to create sufficient efficiency to manage the increasing traffic load of today's networks.

SDN's centralized control plane presents opportunities and challenges for security. In parallel, it has become necessary to ensure strong security mechanisms in SDN networks to protect against a wide range of cyber threats. In [14, 15], the authors have investigated the types of attacks in SDN networks and proposed solutions for them such as Avant-Guard and VAVE (Virtual source Address Validation

[*] Software – Defined Networking
[†] Internet Protocol Security
[‡] Non-Uniform Memory Access

Edge) or the use of SSL[*] encryption and entropy analyses. The result of this research shows that each of the solutions can provide protection against a specific attack and are not able to provide a general and integrated protection solution, and the parallel use of several solutions at the same time will create a lot of overheads in packet processing and routing in the network.

In [15], the authors have provided a solution for the integrity and authentication of the entire network traffic by implementing IPSec in SDN, which results in reducing the risk of unauthorized access and data manipulation.

In summary, the literature review shows ongoing efforts in the research community to increase SDN network speed and security through various techniques. While separate studies have investigated the benefits of DPDK and IPsec in isolation, there is a growing interest in exploring the combined use of solutions and technologies to exploit their synergy together in speed and security in SDN networks. . The integration of IPsec into the DPDK-based packet processing path, along with the use of VPP for optimal routing, provides a promising approach to achieve high-performance and secure SDN deployment. However, further research is needed to explore the practical implications, performance trade-offs, and deployment challenges associated with such an integrated solution.

3. Proposed Technologies

3.1. *Data Plane Development Kit (DPDK)*

DPDK is a set of libraries and drivers designed for high-performance packet processing in user space. Key design goals include minimizing CPU overhead, bypassing conventional kernel-based network stacks, and ensuring deterministic packet processing latency. DPDK implements memory management mechanisms that facilitate low-latency and zero-copy packet handling. The underlying memory architecture relies on several key components:

a. Huge Pages and Physical Address Contiguity:

By using huge pages (commonly 2MB or 1GB), DPDK ensures large contiguous memory segments, reducing the Translation Lookaside Buffer (TLB) overhead and minimizing page walks. The large contiguous memory areas help maintain a stable memory mapping and lower the CPU instruction overhead for address translation. Evidence based evaluations have shown that fewer TLB misses translate directly into improved packet throughput [1, 15].

Formally, the number of pages N_p required for a memory size M given a page size S_p is:

$$N_p = \frac{M}{S_p} \quad (1)$$

In Table 1, you can see the difference in the number of pages for 256 GB memory space:

Table 1. Comparison of the number of memory pages using Huge Pages

Page Size	1 GB	2 MB	4 KB
Number of Pages	256	131,072	67,108,864

To quantify memory efficiency, consider the effective latency L_{eff} of memory access:

$$L_{eff} \approx L_{base} + P_{miss} \times L_{miss} \quad (2)$$

where L_{base} is the base latency, P_{miss} is the probability of a TLB miss, and L_{miss} is the penalty incurred. By leveraging huge pages, DPDK minimizes P_{miss}, thereby reducing L_{eff} and improving packet processing speed. This reduction in complexity and latency can be crucial for applications like IPsec, where even slight overheads in memory access translate into higher per-packet processing times.

- *Within DPDK*: Because huge pages are pre-allocated and contiguous, the time needed to perform address translation per memory access is reduced, maintaining stable and predictable performance.
- *Outside DPDK*: Conventional memory allocations rely on smaller pages and may involve complex page table structures. This can increase TLB misses and lead to non-deterministic memory access times.

b. Non-Uniform Memory Access (NUMA)-Aware Allocation:

DPDK is aware of Non-Uniform Memory Access (NUMA) topologies and guides memory allocations to the memory node local to the CPU core. It uses a memory pool

[*] Secure Sockets Layer

represntation per NUMA node, ensuring that each core accesses memory from the nearest NUMA node, minimizing remote memory access:

$$\text{Allocation}(core_c) \rightarrow mempool^{\,NUMA(core_c)}$$

This ensures remote memory accesses are minimized, improving latency effeciency and maintaining consistent performance [15].

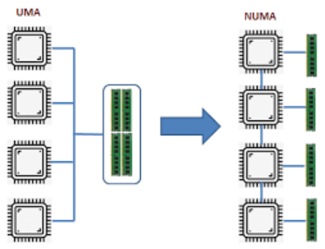


Figure 1. Side-by-side comparison of NUMA and non-NUMA architectures

- *Within DPDK*: NUMA-awareness ensures near-constant and effecient memory access latency by minimizing remote node memory calls.
- *Outside DPDK*: Generic systems often do not control or optimize memory placement. Memory allocated without NUMA-awareness may reside in remote nodes, increasing latency and reducing throughput under heavy loads.

d. Zero-Copy Data Transfer:

DPDK uses zero-copy techniques to avoid unnecessary copying of data between buffers. Traditional packet processing often involves multiple steps in which data is copied from one buffer to another, creating significant overhead. Zero-Copy techniques ensure that data is copied directly from disk to sockets, without entering core buffers, reducing the CPU cycles required to copy data and thus improving performance. This feature minimizes memory copy overhead and these optimizations offload the CPU from basic I/O operations, which is useful when handling large volumes of encrypted traffic [16]

- *Within DPDK*: Packets arrive directly into pre-allocated mbufs, and encryption/decryption (for IPsec) can be offloaded. There is no repeated copying at various layers, reducing CPU cycles and kernel intruptions lead into improving throughput.
- *Outside DPDK*: Traditional stacks may copy packet data multiple times (e.g., from kernel to user space). Such overhead is non-constant and increases latency per packet as network load grows.

c. Direct Memory Access (DMA):

DPDK uses DMA to transfer data directly between the network interface card (NIC) and program memory without CPU intervention. This frees up data transfer tasks to the direct memory access control section of the CPU for other processing tasks and reduces latency by avoiding CPU interrupts. DPDK's use of DMA[*] reduces CPU overhead and increases data transfer speed, which is critical for effective handling encryption/decryption of data packet security protocols such as IPsec. [17]

e. Memory Pools (Mempools) and mbufs:

DPDK’s *rte_mempool* structure pre-allocates a pool of fixed-size buffers (mbufs) at startup. This approach eliminates frequent dynamic memory allocations at runtime. Each mbuf (encapsulated in a structure *rte_mbuf*) holds both packet metadata and data pointers. The fixed-size pre-allocation and pool-based recycling of buffers ensure constant-time buffer allocations and deallocations:

$$T_{alloc}(mbuf) \approx O(1), \quad T_{free}(mbuf) \approx O(1)$$

This deterministic behavior is critical for maintaining low-latency and high-throughput packet processing. [18, 19]

- *Within DPDK*: Because mbufs are pre-allocated and managed in fixed-size pools, allocation and deallocation involve simple pointer operations on lock-free rings, ensuring that each allocation or deallocation operation does not scale with the number of buffers already handled.

[*] Direct Memory Access

- *Outside DPDK*: Standard dynamic memory allocators (e.g., malloc/free) cannot guarantee constant-time complexity, as they often must manage variable-sized requests, coalescing, and complex metadata. This complexity leads to non-constant (potentially $O(\log n)$ or worse) allocation/free operations, degrading performance under load.

f. Local Caches and Bulk Operations:

Each core maintains a local cache of mbufs to further reduce overhead associated with accessing the global Mempool. Bulk allocation and deallocation operations further extinguish the cost of handling buffers by operating on multiple objects simultaneously. Such bulk operations can be expressed as:

$$\text{BulkAlloc}(k) \rightarrow O(1)$$

for a batch of k buffers, significantly reducing per-buffer overhead.

Similar to mbuf allocation and deallocation, bulk operations (allocating or freeing multiple mbufs at once) are also near $O(1)$ per operation since they amortize the overhead over multiple objects, further enhancing throughput. [18, 19]

g. Lockless Data Structures:

In multi-threaded processing, a lock is a synchronization mechanism used to control access to shared resources and ensure that only one thread can access a resource at a time. When a thread holds a lock, other threads that need access to the same resource must wait for the lock to be released, which can cause delays and reduce overall system performance.

DPDK uses lock-free data structures for memory management, which allows multiple CPU cores to simultaneously access and modify memory pools without the overhead of locking mechanisms. This is critical for achieving high performance in multi-core systems, as it avoids the latency associated with locking. [20-22]

- *Within DPDK*: The ring-based lockless approach ensures multi-core scalability. Each operation (enqueue/dequeue) is independent of how many items are already in the ring.
- *Outside DPDK*: Traditional locks may introduce waiting times that scale poorly as more threads and CPU cores compete for shared resources.

h. Additional Memory and Buffer Optimization Features:

DPDK also provides packet burst APIs (e.g., *rte_eth_rx_burst(), rte_eth_tx_burst()*) to fetch and send multiple packets at once. This reduces the overhead of per-packet function calls and enhances instruction cache locality. Prefetch instructions are extensively used within the data path (e.g., *rte_prefetchX()*) to fetch upcoming packet metadata into caches before processing, reducing CPU pipeline stalls.

DPDK Security Library:

The *librte_cryptodev* in DPDK abstracts hardware and software cryptographic accelerators, providing a unified interface for cryptographic operations such as encryption, decryption, hashing, and authentication. The cryptodev library:

- Defines a common API and data-path programming model for crypto operations.
- Supports both dedicated hardware solutions (e.g., Intel QuickAssist adapters) and CPU-based cryptographic extensions (e.g., AES-NI, ARMv8 crypto extensions)
- Uses a queue-based model to submit *rte_crypto_op* structures for batch processing. A *rte_crypto_op* encapsulates all necessary cryptographic operation parameters, keys, and buffer pointers.
- Allows asynchronous processing of cryptographic jobs, enabling pipelines where packet input/output, cryptographic, and other processing stages run concurrently. [23, 24]

Conceptually, the cryptodev subsystem can be modeled as:

$$\begin{tikzcd}\text{Crypto Input} \arrow[r, "\textit{rte\_crypto\_op}"] & \text{Crypto Device} \arrow[r, "\textit{Completion}"] & \text{Processed Packets}\end{tikzcd}$$

This model decouples the cryptographic transformations from the main application logic, enabling easy offloading of IPsec operations and ensuring minimal performance penalty for security.

- *Within DPDK*: Cryptodev integration with IPsec leads to highly efficient cryptographic transformations, often processed in batches and offloaded seamlessly to hardware accelerators.
- *Outside DPDK*: Without cryptodev, implementing high-throughput IPsec would require custom integration of cryptographic engines, often incurring context switches, synchronization overhead, and variable processing times.

Integration with IPsec:

DPDK's cryptodev library forms the backbone of IPsec acceleration. The IPsec protocol stack can leverage the cryptodev APIs to perform Encryption/Authentication (e.g., AES-GCM) on packets inline or via a simple offload model. When combined with DPDK's zero-copy and DMA-based operations, cryptodev ensures cryptographic transformations do not become a bottleneck.

3.2. Internet Protocol Security (IPSec)

As mentioned, IPsec is a set of protocols designed to ensure the confidentiality and integrity of data communications in IP networks. This is possible through two main protocols; Authentication Header (AH), which provides only authentication, and Encapsulating Security Payload (ESP), which provides both authentication and packet confidentiality. These protocols can work either in transport mode, where only the original data is encrypted and the IP data remains intact, or in tunnel mode, where the entire original IP packet (including the IP data and the original data) is encrypted and packed into a new IP packet. The choice of these protocols depends on the security requirements of the network. [25].

IPsec thus provides comprehensive end-to-end security at the network layer, ensuring that sensitive information remains protected regardless of the underlying applications or services.

In the SDN-based, high-performance architecture proposed, IPsec is not only an added feature—it's a strategic choice that complements both DPDK's acceleration capabilities and VPP's modular, vectorized packet processing. By applying cryptographic safeguards at the IP layer, IPsec delivers strong security uniformly across all network functions without requiring application-specific solutions. This uniformity is critical in a modern SDN environment where policies must be administrated centrally yet spreed consistently to all data plane elements. [26] [27]

Key reasons and benefits for integrating IPsec into this framework include:

a. Uniform Security Policy Enforcement:

- IPsec ensures that all traffic, regardless of application or service, is subject to the same security controls.
- This uniformity simplifies network management by centralizing policy definition at the SDN controller, ensuring consistent enforcement across diverse endpoints and network segments.

b. Synergy with DPDK for High Performance:

- Leveraging DPDK's cryptographic acceleration (via the cryptodev library) minimizes the performance overhead typically associated with encryption and authentication tasks.
- Hardware offloads and zero-copy buffering keep latency low and throughput high, ensuring that security does not become a bottleneck.

c. Scalability and Adaptability in SDN:

- As SDN controllers dynamically adjust network flows, IPsec policies can be updated on-the-fly, maintaining secure channels in line with evolving operational requirements.
- This capability ensures that as new routes are provisioned or old ones are reconfigured, the confidentiality and integrity of data remain uncompromised.

d. Reduced Complexity and Dependency on Specialized Hardware:

- Instead of relying on dedicated encryption devices or specialized security appliances, IPsec-enabled packet processing can be implemented on servers running the SDN data plane.
- This approach cuts down on complexity, reduces hardware acquisition costs, and

allows rapid scaling or re-deployment of security functions as the network grows or changes.

e. Interoperability and Industry Support:

- IPsec is widely recognized, standardized, and supported across networking equipment and vendor solutions.
- Using IPsec eases integration with existing infrastructure, partner networks, and cloud services that already rely on these well-understood security protocols.

f. Sustained High Throughput Under Cryptographic Load:

- With DPDK handling the packet I/O efficiently and VPP optimizing packet processing workflows, IPsec encryption/decryption functions can be performed at near line-rate speeds.
- The architecture can handle heavy traffic loads, multiple tunnels, and a wide variety of topologies without significant degradation in performance.

g. Seamless Integration with VPP:

- IPsec nodes can be incorporated into the VPP processing graph, enabling vectorized operations and batch processing of encrypted flows.
- This integration ensures that packet processing, from ingress to secure encapsulation to forwarding, occurs smoothly within a single, unified pipeline.

3.3. Vector Packet Processing (VPP)

VPP is an open-source, high-performance packet processing engine that applies vectorization principles to packet operations. Rather than processing one packet at a time (scalar processing), VPP processes batches (vectors) of packets through a directed graph of processing nodes to minimize instruction cache misses and improve throughput. Each node applies a specific function to the entire vector before moving on [28].

Key Architectural Elements of VPP:

a. I-Cache Efficiency and Vectorization:

In scalar processing, each packet may cause new instructions to load into the CPU's instruction cache (I-cache), increasing the probability of I-cache thrashing. With vector processing, the CPU executes the same instructions on multiple packets consecutively, significantly reducing instruction cache misses. Once the I-cache is "warm" for the first packet in the vector, subsequent packets in that vector get the benefit:

$$\text{Processing}(V_{pkts}) = \sum_{i=1}^{|V_{pkts}|} f(packeti) \text{ with reduced per-packet overhead. (3)}$$

b. Graph-Based Extensibility:

VPP models the data plane as a graph G=(V,E), where each node v∈V in the graph represents a distinct feature or function (e.g., L2 input, IP4 input, IPsec tunnel input) and Edges e∈E represent how packets flow between features. Packets traverse nodes as edges dictate. This modular structure allows easy insertion of new features (e.g., IPsec nodes) or hardware offload blocks:

G=(V,E): Each v∈V is a function; edges E define flow o f packets.

c. Parallelism and Scalability:

VPP exploits multi-core CPU architectures by distributing graph nodes across threads and cores. Combined with DPDK's lockless memory structures, VPP can scale linearly with CPU cores:

- *Within DPDK*: The synergy of vectorized packet processing with pre-allocated mbufs and zero-copy techniques improves throughput and latency stability.
- *Without DPDK*: Other packet processing frameworks may not achieve such efficient vectorization or may rely on kernel-based stacks causing unpredictable latency and lower throughput.

d. Integration with IPsec:

Integrating IPsec within VPP's graph nodes (e.g., ESP encapsulation/decapsulation nodes) ensures that cryptographic transformations occur inline and benefit from vectorization. Leveraging DPDK's cryptodev library, VPP can offload cryptographic operations and process packets in batches, reducing overhead per packet and maintaining stable line-rate performance even under encryption-intensive workloads

This vectorized architecture substantially reduces CPU pipeline stalls and I-cache thrashing, making VPP particularly suitable for high-throughput, low-latency networking functions required in advanced SDN deployments. When combined with DPDK's underlying fast I/O and IPsec hardware acceleration, VPP enables a highly modular, scalable, and efficient software-based networking stack [29-33].

The following is architectural view of the VPP layer. [32]

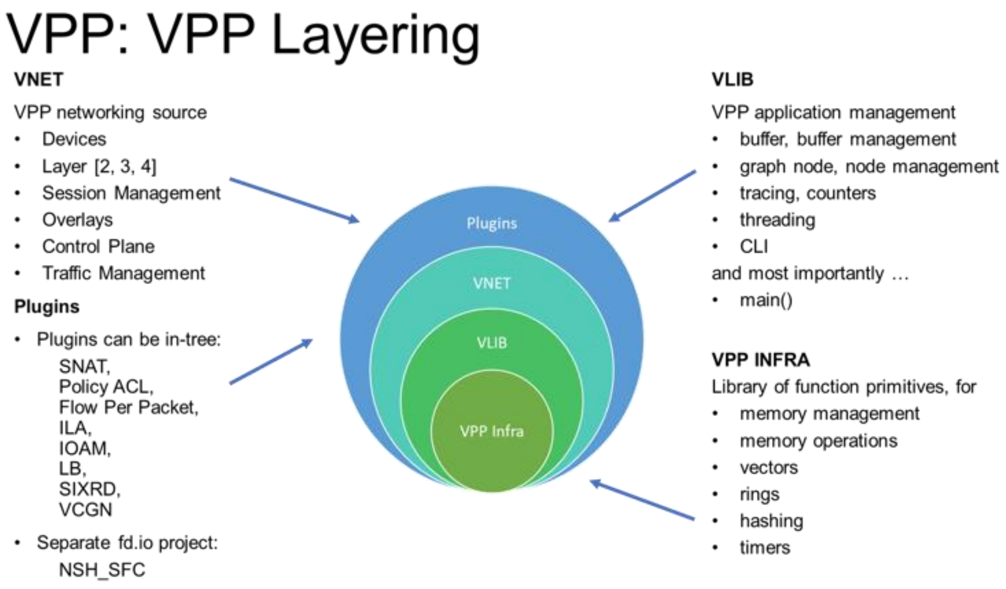


Figure 2. VPP Layers

VPP Infra:

The infrastructure layer of VPP, which contains the source code of the kernel library. This layer performs memory operations, works with vectors and loops, performs key lookups in tables, and works with timers to dispatch graph nodes.

VLIB (Vector Library):

The VLIB is at the core of VPP. It provides fundamental data structures (vectors, hash tables, memory allocators), scheduling routines for graph nodes, and thread management. The vector abstraction reduces I-cache thrashing by ensuring that the CPU repeatedly executes the same sequence of instructions on a batch of packets before moving to another node.

VNET (VPP Network Stack):

The VNET layer builds on VLIB and provides network-specific functionalities: IP routing, switching, and session management. It interacts closely with DPDK for fast I/O and can integrate IPsec processing nodes that rely on cryptodev accelerators.

Plugins and Extensibility:

VPP supports plugins that can introduce new graph nodes or modify existing ones. For IPsec, there are dedicated plugin nodes handling ESP encapsulation/decapsulation. These nodes invoke cryptodev-like interfaces to offload cryptographic tasks if available.

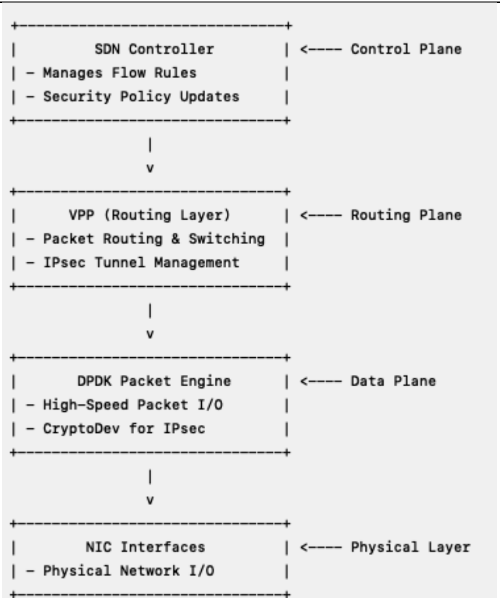
4. Proposed System Architecture

The proposed system architecture leverages DPDK for accelerated packet I/O and cryptographic offloads, integrates IPsec security features for data confidentiality and integrity, and employs VPP's vectorized packet processing to maintain high throughput at scale. An SDN controller provides centralized logic for dynamic policy adjustments, key management, and network configuration. The combination of these technologies ensures secure, flexible, and performant traffic handling suitable for modern SDN environments.

At a high level, the data plane—composed of DPDK and VPP—handles packets entirely in user space. This eliminates kernel-based context switching, interuptions and overhead, achieving near line-rate processing even under cryptographic loads. The control plane—managed by an SDN controller—enforces IPsec policies, updates Security Associations (SAs), and manage VPP graph configurations in real-time, enabling dynamic adaptation to changing network conditions. Each component plays a specialized role:

Table 2. Proposed System Architect Component Relationship

Component	Role	Dependency
SDN Controller	Policy and rule management	VPP
VPP	Dynamic routing and IPsec setup	DPDK, SDN Controller
DPDK	Fast packet I/O and encryption	CryptoDev, NICs
CryptoDev	IPsec encryption and decryption	DPDK
NICs	Physical packet transmission	DPDK



```
+--------------------------------+
|        SDN Controller          | <---- Control Plane
| - Manages Flow Rules           |
| - Security Policy Updates      |
+--------------------------------+
                |
                v
+--------------------------------+
|     VPP (Routing Layer)        | <---- Routing Plane
| - Packet Routing & Switching   |
| - IPsec Tunnel Management      |
+--------------------------------+
                |
                v
+--------------------------------+
|      DPDK Packet Engine        | <---- Data Plane
| - High-Speed Packet I/O        |
| - CryptoDev for IPsec          |
+--------------------------------+
                |
                v
+--------------------------------+
|       NIC Interfaces           | <---- Physical Layer
| - Physical Network I/O         |
+--------------------------------+
```

Figure 3. Proposed system Architect Diagram

4.1. *System Data Flow*

The following outlines the end-to-end flow of a packet through the system:

1. **Packet Ingress:** Packets arrive at a DPDK-enabled NIC and are received directly into mempools using the Poll Mode Driver (PMD). This bypasses the kernel, eliminating interrupt-driven overhead.
2. **IPsec Processing:** Packets requiring encryption or decryption are handed to CryptoDev for IPsec transformations:
    - The IPsec policies and Security Associations (SAs) are defined and secure IPsec tunnels established and maintain by using the IKEv2 protocol.
    - CryptoDev performs encryption (e.g. AES-GCM) or decryption based on these policies.
3. **Routing and Switching (VPP):** VPP retrieves the encrypted/decrypted packets and processes them through a graph of nodes. The VPP IPsec nodes integrate with CryptoDev to offload cryptographic operations efficiently.
4. **Dynamic Control:** The SDN Controller interfaces with VPP and IPSec to:
    - Update IPsec policies and cryptographic keys dynamically.

- Modify VPP graph configurations to adapt to real-time network changes.

5. **Packet Egress:** VPP forwards the processed packets back to DPDK, which transmits them through the NIC to the next hop.

This pipeline ensures minimal latency and efficient utilization of CPU and hardware resources.

4.2. Key Components

4.2.1. DPDK: High-Performance Packet Processing

DPDK is the foundation for the data plane, enabling zero-copy, user-space packet processing.

Key Features:

1. **Poll Mode Driver (PMD)**:
   - Captures packets directly from the NIC, bypassing the kernel to reduce latency.
   - Processes packets using a continuous polling mechanism, eliminating interrupts.
2. **Memory Management**:
   - Mempools: Pre-allocated memory pools for packet buffers.
   - Huge Pages: Uses large memory pages (2 MB) to reduce TLB misses and improve memory access speed.
3. **CryptoDev Library**:
   - Provides an interface for offloading encryption/decryption tasks to hardware accelerators.
   - Ensures encryption mrthods for IPsec is performed at line rate.

4.2.2. IPsec: Secure Communication

IPsec ensures data confidentiality, integrity, and authentication at the network layer. It is selected for its modular design, compliance with IKEv2 standards, and support for hardware acceleration to enhance performance.

Key Features:

1. **IKEv2 Daemon:**
   - Negotiates secure IPsec tunnels between endpoints.
   - Manages key exchanges and tunnel lifetimes.
2. **ESP (Encapsulating Security Payload):**
   - Encrypts and authenticates packet payloads, commonly using different algorithm (e.g. AES-GCM) for both encryption and integrity.
3. **Integration with DPDK:**
   - Utilizes DPDK’s CryptoDev for hardware-accelerated encryption.
   - Interfaces with VPP through tap interfaces to ensure efficient packet processing and routing.

4.2.3. VPP: Dynamic Routing and Switching

VPP is responsible for routing and switching packets efficiently.

Key Features:

1. **Graph-Based Processing:**
   - Packet flows are processed through modular graph nodes, each performing a specific function (e.g., IPsec processing, routing).
2. **IPsec Tunnel Management:**
   - VPP applies IPsec tunnels and manages packet forwarding dynamically.
3. **Interface Management:**
   - VPP supports physical interfaces (e.g., NICs) and virtual interfaces like tap devices.

4.3. Implementation Approach

The implementation is divided into three key stages:

1. **System Initialization and Configuration:**
   - Setting up DPDK to handle packet ingress and egress efficiently.
   - Configuring CryptoDev for hardware-accelerated IPsec operations.
   - Initializing VPP to manage routing, switching, and tunnel configurations.
2. **Packet Processing Pipeline:**
   - Ingesting packets from NICs using DPDK's Poll Mode Drivers (PMD).
   - Encrypting or decrypting packets with IPsec through CryptoDev.
   - Routing packets dynamically using VPP based on SDN controller policies.

3. **Performance Tuning and Optimization:**
    - Fine-tuning buffer sizes, queue depths, and other system parameters to handle high throughput.
    - Implementing parallelism across cores for scalability.

4.4. Important Configuration and Code Details

This section provides essential configurations and code snippets. It highlights the parameters critical for replication and execution.

4.4.1. DPDK Initialization Code

The following code initializes the DPDK environment and configures memory pools and network interfaces.

```
// Initialize Environment Abstraction Layer (EAL)
int ret = rte_eal_init(argc, argv);
if (ret < 0) {
  rte_exit(EXIT_FAILURE, "EAL initialization failed\n");
}

// Memory Pool for Packet Buffers
struct rte_mempool *mbuf_pool = rte_pktmbuf_pool_create(
  "MBUF_POOL", 8192, 250, 0, RTE_MBUF_DEFAULT_BUF_SIZE, rte_socket_id()
);
if (mbuf_pool == NULL) {
  rte_exit(EXIT_FAILURE, "Cannot create memory pool\n");
}

// Configure NIC Ports
uint16_t port_id;
RTE_ETH_FOREACH_DEV(port_id) {
  struct rte_eth_conf port_conf = {
    .rxmode = { .max_rx_pkt_len = RTE_ETHER_MAX_LEN },
  };
  rte_eth_dev_configure(port_id, 1, 1, &port_conf);
  rte_eth_rx_queue_setup(port_id, 0, 512, rte_eth_dev_socket_id(port_id), NULL, mbuf_pool);
  rte_eth_tx_queue_setup(port_id, 0, 512, rte_eth_dev_socket_id(port_id), NULL);
  rte_eth_dev_start(port_id);
}
```

Parameter Descriptions:

- **rte_pktmbuf_pool_create:**
    - "MBUF_POOL": Name of the memory pool.
    - 8192 (NUM_MBUFS): Number of buffers in the pool. Should match packet throughput to avoid dropping packets.
    - 250 (MBUF_CACHE_SIZE): Number of buffers cached per core to reduce mempool access overhead.
    - RTE_MBUF_DEFAULT_BUF_SIZE (2 048): Size of each buffer. This must account for packet size + headers.
- **rte_eth_rx_queue_setup / rte_eth_tx_queue_setup:**
    - 512: Queue size. Challenge: Queue size must match traffic burst size to avoid packet drops or excessive latency.

4.4.2. CryptoDev Configuration for IPsec

```
// Configure Crypto Device
struct rte_cryptodev_config dev_conf = {
  .socket_id = rte_socket_id(),
  .nb_queue_pairs = 2, // Supports parallelism
};
rte_cryptodev_configure(crypto_dev_id, &dev_conf);

// Configure IPsec Cipher and Authentication
struct rte_crypto_sym_xform cipher_xform = {
  .type = RTE_CRYPTO_SYM_XFORM_CIPHER,
  .cipher = {
    .algo = RTE_CRYPTO_CIPHER_AES_GCM,
    .key.length = 32, // AES-256
  },
};

struct rte_crypto_sym_xform auth_xform = {
  .type = RTE_CRYPTO_SYM_XFORM_AUTH,
  .auth = {
    .algo = RTE_CRYPTO_AUTH_SHA256_HMAC,
    .key.length = 32, // 256-bit HMAC
  },
};

// Link Cipher and Auth
```

cipher_xform.next = &auth_xform;

Parameter Descriptions:

- **rte_cryptodev_config:**
  - socket_id: NUMA-aware placement of cryptographic operations.
  - nb_queue_pairs: Number of queue pairs. This enables parallel processing of crypto tasks across cores.
- **RTE_CRYPTO_SYM_XFORM_CIPHER / RTE_CRYPTO_SYM_XFORM_AUTH:**
  - RTE_CRYPTO_CIPHER_AES_GCM: AES-GCM is chosen for efficiency as it combines encryption and integrity.
  - key.length: Key size (32 bytes = 256-bit) ensures strong security.

4.4.3. VPP Configuration for Routing and IPsec

```
bash
Copy code
# VPP Tap Interface
vppctl create tap id 0 host-if-name vpp-tap0
vppctl set int state tap0 up
vppctl set int ip address tap0 10.0.0.1/24

# IPsec Tunnel Configuration
vppctl ipsec add tunnel src 10.0.0.1 dst 192.168.1.1 local-spi 100 remote-spi 200 crypto-alg aes-gcm-256 integ none
vppctl set interface state ipsec0 up

# Static Route Configuration
vppctl ip route add 192.168.2.0/24 via 10.0.0.2 tap0
```

Parameter Descriptions:

- **create tap id 0 host-if-name vpp-tap0:**
  - id 0: Tap interface ID.
  - vpp-tap0: Interface name visible to the host. Challenge: Interface names must not conflict with existing interfaces.
- **ipsec add tunnel:**
  - src / dst: Source and destination IP addresses for the IPsec tunnel.
  - local-spi / remote-spi: Security Parameter Indexes (SPI) for the tunnel. These values must match on both endpoints.
  - crypto-alg aes-gcm-256: Specifies AES-GCM for combined encryption and integrity.
  - integ none: GCM already provides integrity, so no additional algorithm is needed.
- **ip route add:**
  - 192.168.2.0/24: Remote subnet routed through the IPsec tunnel.
  - via 10.0.0.2: Next-hop address.

SPI mismatches or incorrect routes can result in dropped packets or failed tunnel setups.

Each code snippet above plays a critical role in building a high-performance, secure SDN network. Key challenges include:

1. **Resource Allocation:** Ensuring sufficient hugepages, memory pools, and NIC queues.
2. **CryptoDev Hardware Support:** Hardware acceleration must be enabled to achieve optimal IPsec performance.
3. **VPP Tunnel Integrity:** SPI values, encryption algorithms, and routes must align between endpoints.
4. **NUMA Optimization:** Cores, memory, and NIC placement must be NUMA-aware to reduce latency.

Use-Case Scenarios and future AI approach

Here are practical examples of how the system operates in real-world scenarios:

1. **Secure Communication Between Data Centers**:
   - Traffic between two data centers is encrypted using IPsec tunnels.
   - DPDK handles the high-throughput data plane, and VPP dynamically routes the traffic based on SDN policies.
2. **Dynamic Policy-Based Routing**:
   - The SDN Controller updates flow rules in real time (e.g., due to congestion or a DDoS attack).
   - VPP immediately adapts and reroutes traffic without interrupting packet flow.
3. **High-Performance Firewall**:
   - Incoming packets are inspected, encrypted/decrypted (using CryptoDev), and forwarded efficiently by VPP through DPDK.

Artificial intelligence approaches to achieve network security in SDN networks

The application of artificial intelligence (AI) and machine learning (ML) techniques to SDN environments is gaining momentum as network operators seek more dynamic, autonomous, and robust solutions. AI-driven strategies can enable real-time analysis of traffic patterns, predictive modeling of network conditions, and automated decision-making to enhance both performance and security.

Studies have demonstrated the effectiveness of AI-driven methods for optimizing routing and load balancing within SDN. For example, work in [34] employed reinforcement learning to leverage real-time feedback from the network, dynamically adjusting paths and improving throughput. By periodically evaluating network conditions, the controller can determine when and how to reroute traffic, thus improving resilience and efficiency. Similarly, the authors in [35] utilized Q-learning to improve data transmission rates in SDN scenarios. Their approach tested single-metric and multi-metric variants of Q-Routing algorithms, revealing that single-metric Q-Routing significantly outperformed traditional heuristic-based routing in both static and dynamic topologies, while also converging faster than shortest-path (K-Shortest Path) algorithms.

These findings indicate that employing AI within SDN can substantially enhance network performance and intelligence. By integrating such AI-based routing and decision-making mechanisms with the secure, high-throughput architecture proposed in this paper—where SDN is fortified by DPDK, IPsec, and potentially VPP—network operators can ensure that optimization and security operate in tandem. As future research continues to refine AI techniques and incorporate more metrics, the fusion of AI-based decision engines with secure, high-speed SDN frameworks may set a new standard for both performance and protection.

5. Conclusion

This paper presents a conceptual integration of IPsec, DPDK, and VPP within an SDN architecture, aiming to achieve both high-performance packet handling and stringent security measures. The technologies discussed exhibit well-documented capabilities that strongly suggest synergistic gains. DPDK, by bypassing the kernel and employing advanced memory management techniques, delivers line-rate packet processing essential for sustaining high throughput in increasingly complex networks. IPsec, integrated through hardware-accelerated cryptodev interfaces, ensures robust end-to-end data protection without imposing prohibitive overheads.

VPP contributes an additional layer of agility and scalability, allowing for the seamless introduction of custom routing and switching functionalities. When managed by a centralized SDN controller, the resulting framework maintains low latency, secure operations, and dynamic adaptability. This aligns well with evolving network demands, where rapid response to changing traffic conditions and threat landscapes is critical.

Future empirical validations, performance benchmarks, and expanded AI-driven routing approaches can further refine and solidify this integrated solution. As research and development progress, the combined strengths of SDN, IPsec, DPDK, and VPP stand to shape next-generation networking paradigms, optimizing for both efficiency and security across diverse and distributed environments.

Authors' Contributions

Authors equally contributed to this article.

Acknowledgments

Authors thank all participants who participate in this study.

Declaration of Interest

The authors report no conflict of interest.

Funding

According to the authors, this article has no financial support.

Ethical Considerations

All procedures performed in this study were under the ethical standards.

References

[1] A. Haggag, "Network optimization for improved performance and speed for SDN and security analysis of SDN vulnerabilities," *International Journal of Computer Networks and Communications Security*, vol. 5, pp. 83-90, May 2019.

[2] A. D. Al-Ani and N. I. Abdullah, "Software defined networks challenges and future direction of research," *International Journal of Research*, vol. 1, pp. 618-629, Jan 2019.

[3] A. Coly, M. Mbaye, and S.-L. Gaston Berger University, "S-SDS: a framework for security deployment as service in software defined networks," May 2019.

[4] G. Lopez-Millan, R. Marin-Lopez, and F. Pereniguez-Garcia, "Towards a standard SDN-based IPsec management framework," *Journal of Computer Standards & Interfaces*, vol. 66, May 2019, doi: 10.1016/j.csi.2019.103357.

[5] Dpdk.org, "Poll Mode Driver - Data Plane Development Kit 24.03.0 documentation." [Online]. Available: http://doc.dpdk.org/guides-24.03/prog_guide/poll_mode_drv.html.

[6] L. Linguaglossa, D. Rossi, S. Pontarelli, C. Telecom ParisTech, V. University of Rome Tor, and I. Cisco Systems, "High-speed data plane and network functions virtualization by vectorizing packet processing," *Journal of Computer Networks*, vol. 149, pp. 187-199, Feb 2019, doi: 10.1016/j.comnet.2018.11.033.

[7] J. Pak and K. Park, "A High-Performance implementation of an IoT system using DPDK," *Journal of Applied Sciences*, vol. 8, no. 4, p. 550, Apr 2018, doi: 10.3390/app8040550.

[8] A. Belkhiri, M. Pepin, M. Bly, M. Polytechnique, and I. Ciena, "Performance analysis of DPDK-based applications through tracing," *Journal of Parallel and Distributed Computing*, vol. 173, pp. 1-19, Mar 2023, doi: 10.1016/j.jpdc.2022.10.012.

[9] S. Kaur, K. Kumar, N. Aggarwal, E. University Institute of, and P. U. C. I. Technology, "A review on P4-Programmable data planes: Architecture, research efforts, and future directions," *The International Journal for the Computer and Telecommunications*, vol. 170, pp. 109-129, Mar 2021, doi: 10.1016/j.comcom.2021.01.027.

[10] T. Döring, H. Stubbe, K. Holzinger, A. Chair of Network, and D. o. I. T. U. o. M. G. Services, "SmartNICs: Current trends in research and industry," May 2021.

[11] X. Yang and L. Wang, "SDN Load Balancing Method based on K-Dijkstra," *International Journal of Performability Engineering*, vol. 14, no. 4, pp. 709-716, Apr 2018, doi: 10.23940/ijpe.18.04.p14.709716.

[12] A. Kumar, D. Anand, and M. Chandigarh University, "Load balancing for software defined network using machine learning," *Turkish Journal of Computer and Mathematics Education*, vol. 12, no. 12, pp. 527-535, Apr 2021, doi: 10.17762/turcomat.v12i2.876.

[13] D. Todorov, H. Valchanov, and V. Aleksieva, "Load Balancing model based on Machine Learning and Segment Routing in SDN," in *2020 International Conference Automatics and Informatics (ICAI)*, Varna, Bulgaria, Oct 2020, pp. 1-4, doi: 10.1109/ICAI50593.2020.9311385.

[14] J. Spooner and S. Y. Zhu, "A review of solutions for SDN-Exclusive security issues," *International Journal of Advanced Computer Science and Applications*, vol. 7, no. 8, 2016.

[15] A. Pradhan and R. Mathew, "Solutions to vulnerabilities and threats in Software defined networking (SDN)," in *Third International Conference on Computing and Network Communications*, Jan 2020, vol. 171, pp. 2581-2589, doi: 10.1016/j.procs.2020.04.280.

[16] Dpdk.org, "IPv4 Multicast Sample Application - Data Plane Development Kit 24.03.0 documentation." [Online]. Available: http://doc.dpdk.org/guides-24.03/sample_app_ug/ipv4_multicast.html.

[17] J. Kubálek and T. Brno University of, "High-speed DMA packet transfer in system DPDK," May 2018.

[18] Dpdk.org, "Mempool Library - Data Plane Development Kit 24.03.0 documentation." [Online]. Available: https://doc.dpdk.org/guides/prog_guide/mempool_lib.html.

[19] Dpdk.org, "Mbuf Library - Data Plane Development Kit 24.03.0 documentation." [Online]. Available: https://doc.dpdk.org/guides/prog_guide/mbuf_lib.html.

[20] A. Baumstark and C. Pohl, "Lock-free data structures for data stream processing," *Datenbank-Spektrum Journal*, vol. 19, pp. 209-218, Oct 2019, doi: 10.1007/s13222-019-00329-4.

[21] J. Kong, "DPDK Optimization on arm," *Tools, Software and IDEs - Arm Community*, 2022. [Online]. Available: https://community.arm.com/arm-community-blogs/b/tools-software-ides-blog/posts/dpdk-optimization-on-arm.

[22] Dpdk.org, "RCPU Library - Data Plane Development Kit 24.03.0 documentation." [Online]. Available: https://doc.dpdk.org/guides/prog_guide/rcu_lib.html.

[23] Dpdk.org, "Cryptography Device Library - Data Plane Development Kit 24.03.0 documentation." [Online]. Available: https://doc.dpdk.org/guides/prog_guide/cryptodev_lib.html.

[24] Dpdk.org, "Security Library - Data Plane Development Kit 24.03.0 documentation." [Online]. Available: https://doc.dpdk.org/guides/prog_guide/rte_security.html.

[25] E. Barker, Q. Dang, S. Frankel, K. Scarfone, and P. Wouters, "Guide to IPSEC VPNs," *National Institute of Standards and Technology*, Jun 2020, doi: 10.6028/NIST.SP.800-77r1.

[26] M. Vajaranta, J. Kannisto, J. Harju, and T. Tampere University of, "IPSEC and IKE as functions in SDN controlled network," in *11th International Conference on Network and System Security*, Helsinki, Finland, Aug 2017, pp. 521-530, doi: 10.1007/978-3-319-64701-2_39.

[27] O. Abolade, A. Okandeji, A. Oke, M. Osifeko, and A. Oyedeji, "Overhead effects of data encryption on TCP throughput across IPSEC secured network," *Journal of Scientific African*, vol. 13, p. e00855, Sep 2021, doi: 10.1016/j.sciaf.2021.e00855.

[28] Fd.io, "What is vector packet processing? - Vector Packet Processor 01 documentation." [Online]. Available: https://fdio-vpp.readthedocs.io/en/latest/overview/whatisvpp/what-is-vector-packet-processing.html.

[29] D. Barach, L. Linguaglossa, D. Marion, P. Pfister, S. Pontarelli, and D. Rossi, "High-Speed Software Data Plane via Vectorized Packet Processing," *IEEE Communications Magazine*, vol. 56, no. 12, pp. 97-103, Dec 2018, doi: 10.1109/MCOM.2018.1800069.

[30] Fd.io, "Scalar vs Vector packet processing - The Vector Packet Processor v24.06-rc1-0-gb3304b2b7 documentation." [Online]. Available: https://s3-docs.fd.io/vpp/24.06/aboutvpp/scalar-vs-vector-packet-processing.html.

[31] Fd.io, "VPP Technology." [Online]. Available: https://fd.io/technology/.

[32] Fd.io, "The Packet Processing Graph - The Vector Packet Processor v22.10-0-g07e0c05e6 documentation." [Online]. Available: https://docs.fd.io/vpp/22.10/aboutvpp/extensible.html?highlight=modular.

[33] The Linux Foundation, "FD.io doubles packet throughput performance to terabit levels - Linux Foundation," Sep 13, 2022. [Online]. Available: https://www.linuxfoundation.org/press/press-release/fd-io-doubles-packet-throughput-performance-to-terabit-levels.

[34] M. A. Jameel, T. Kanakis, S. Turner, A. Al-Sherbaz, and W. S. Bhaya, "A Reinforcement Learning-Based Routing for Real-Time Multimedia Traffic Transmission over Software-Defined Networking," *International Journal of Electronics*, vol. 11, no. 15, p. 2441, Aug 2022, doi: 10.3390/electronics11152441.

[35] D. Harewood-Gill, T. Martin, and R. Nejabati, "The Performance of Q-Learning within SDN Controlled Static and Dynamic Mesh Networks," in *2020 6th IEEE Conference on Network Softwarization (NetSoft)*, Ghent, Belgium, 2020, pp. 185-189, doi: 10.1109/NetSoft48620.2020.9165530.