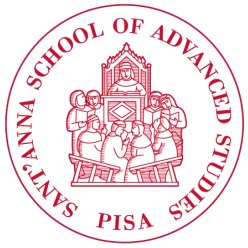
Data Processing Units for Next-Generation Networks: Evaluation across User Plane Function, High-Performance Computing, and Cybersecurity Workload



A thesis submitted in fulfillment of the requirements.

for the degree of

Doctor of Philosophy

in

Emerging Digital Technologies

by

Rana Abu Bakar

to

The Telecommunications, Computer Engineering, and Photonics Institute (TeCIP)

Scuola Superiore Sant'Anna

PISA - 56127, ITALY

Supervisor
Prof. Piero Castoldi
Scuola Superiore Sant'Anna
PISA, 56127, ITALY

Co-Supervisor
Dr. Filippo Cugini
Head of Research Sector
CNIT, PISA, 56127, ITALY

Copyright

DEDICATION

This thesis is dedicated to the memory of my mother, whose love and sacrifices continue to guide me, even though she left this world when I was still a child.

To my father, whose unwavering dedication, strength, and selflessness shaped my life. He chose to devote his entire life to raising us with love and integrity, placing our future above his own.

To my wife, for her patience, resilience, and constant support throughout this long journey.

To my son, who walked this path with me at a very young age, sacrificing parts of his childhood, his language, and his familiar world so that I could pursue this dream. Your strength inspires me every day.

And to my teachers, whose guidance, knowledge, and belief made this work possible.

Acknowledgments

I would like to express my sincere gratitude to my supervisor, Prof. Piero Castoldi, for his continuous guidance and support throughout my doctoral research. His vision, encouragement, and trust allowed me to explore ambitious research directions and to grow as an independent researcher.

I am especially grateful to my co-supervisor, Dr. Filippo Cugini, for his constant technical guidance and deep insight across all areas of my research. His strong understanding of both theoretical and practical aspects of networking and systems research was invaluable. He guided me precisely at every stage of this journey, not only in research-related matters but also in navigating academic and professional life during my three years in Italy. His support extended well beyond supervision, and I am deeply thankful for that.

I would also like to thank Dr. Francesco Paolucci, the GURU. Dr. Andrea Sgambelluri and Dr. Lorenzo De Marinis for their technical guidance and constructive feedback throughout my research journey. Their expertise and discussions helped refine several aspects of my work.

My sincere thanks go to Dr. Muhammad Imran for his valuable guidance and insightful research discussions, often during informal meetings and lunch-time conversations. Although he is not a co-author in my papers, I learned great things from our discussions, particularly through sharing ideas in our native language, which made complex concepts easier to explore and understand.

I am also grateful to Dr. Faris for introducing me to SmartNIC and for providing early guidance that shaped the direction of my research in this area.

Finally, I would like to acknowledge all colleagues and collaborators who contributed directly or indirectly to this work, and whose discussions and support made this research journey both productive and meaningful.

Rana Abu Bakar

LIST OF PUBLICATIONS

1. Bakar, R. A., Paolucci, F., Cugini, F., Vegas Olmos, J. J., & De Marinis, L. (2025, December). IDS-NGNN: A SmartNIC-based intrusion detection system based on reduce and merge nested graph neural networks. Accepted at the IEEE Global Communications Conference (GLOBECOM 2025), Taipei, Taiwan.

2. Bakar, R. A., Ibrahim, A. S. T., Paolucci, F., Sgambelluri, A., Castoldi, P., Cugini, F., & Vegas Olmos, J. J. (2025, December). 6GUPF: A DPU-based programmable user plane function for enhanced flow-based QoS. Accepted at the IEEE Global Communications Conference (GLOBECOM 2025), Taipei, Taiwan.

3. Bakar, R. A., Castoldi, P., Paolucci, F., & Cugini, F. (2025, June). Next-Generation Intrusion Prevention System Using Hardware-Accelerated Data Processing Units (DPUs). In 2025 IEEE International Conference on Communications Workshops (ICC Workshops) (pp. 1438-1443). IEEE.

4. Bakar, R. A., Castoldi, P., Cugini, F., & Paolucci, F. (2025, May). A Hardware-Accelerated Intrusion Prevention System for Attack Mitigation Using DPUs. In IEEE INFOCOM 2025-IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS) (pp. 1-6). IEEE.

5. Bakar, R. A., De Marinis, L., Cugini, F., & Paolucci, F. (2024). FTG-Net-E: A hierarchical ensemble graph neural network for DDoS attack detection. Computer Networks, 250, 110508.

6. Bakar, R. A., Alhamed, F., Castoldi, P., Sgambelluri, A., Olmos, J. J. V., Cugini, F., & Paolucci, F. (2024, July). 5GDAD: A deep learning approach for DDoS attack detection in 5G P4-based UPF. In 2024 IEEE 25th International Conference on High Performance Switching and Routing (HPSR) (pp. 185-190). IEEE.

7. Bakar, R. A., Paolucci, F., Cugini, F., Castoldi, P., & Olmos, J. J. V. (2024). DPUAUT: Secure authentication protocol with SmartNiC integration for trustworthy communications in intelligent swarm systems. IEEE Access, 12, 89986-90004.

8. Lawo, D., Bakar, R. A., Aguilera, A. C., Cugini, F., Imana, J. L., Monroy, I. T., & Olmos, J. V. (2025). Future Data Centers: Hardware-Offloaded Post-Quantum Secure Optical IPsec Tunnel for Confidential AI Training.

9. Castoldi, P., Bakar, R. A., Sgambelluri, A., Olmos, J. J. V., Paolucci, F., & Cugini, F. (2025). Programmable packet-optical network security and monitoring using DPUs with embedded GPUs. Journal of Optical Communications and Networking, 17(2), A178-A195.

10. Javed, M. S., & Bakar, R. A. (2024). A Secure and Green Cognitive Routing Protocol for Wireless Ad-Hoc Networks. IEEE Access.

11. Hinic, S., Bakar, R. A., Marotta, A., & Paolucci, F. (2024, December). Wire-speed DDoS Attack Mitigation using Hardware Acceleration of Programmable DPUs. In GLOBECOM 2024-2024 IEEE Global Communications Conference (pp. 1197-1202). IEEE.

12. Birtane, S., Al-Naday, M., Paolucci, F., Bakar, R. A., Lefebvre, V., Passas, V., ... & Gür, G. (2024, November). Footprint-Optimized Orchestration and Management of Secure Complex Services over 6G Continuum. In 2024 IEEE Conference on Standards for Communications and Networking (CSCN) (pp. 383-388). IEEE.

13. Cugini, F., Bakar, R. A., Sgambelluri, A., Sambo, N., De Marinis, L., Giorgetti, A., ... & Paolucci, F. (2024, September). Data Processing Unit (DPU) and P4 Programmability in Support of the Edge Continuum. In ECOC 2024; 50th European Conference on Optical Communication (pp. 1611-1614). VDE.

14. Lawo, D. C., Abu Bakar, R., Cano Aguilera, A., Cugini, F., Imaña, J. L., Tafur Monroy, I., & Vegas Olmos, J. J. (2024). Wireless and Fiber-Based

Post-Quantum-Cryptography-Secured IPsec Tunnel. Future Internet, 16(8), 300.

15. Aguilera, A. C., Bakar, R. A., Alhamed, F., Garcia, C. R., Imaña, J. L., Monroy, I. T., ... & Olmos, J. V. (2024, March). First line-rate end-to-end post-quantum encrypted optical fiber link using data processing units (dpus). In 2024 Optical Fiber Communications Conference and Exhibition, OFC 2024 (pp. M1G-4). Optica Publishing Group.

16. Castoldi, P., Bakar, R. A., Sgambelluri, A., Olmos, J. J. V., Paolucci, F., & Cugini, F. (2024, March). Programmable packet-optical networks using data processing units (dpus) with embedded gpu. In Optical Fiber Communication Conference (pp. Tu3B-4). Optica Publishing Group.

17. De Marinis, L., Paolini, E., Bakar, R. A., Cugini, F., & Paolucci, F. (2023, December). Cascaded look up table distillation of P4 deep neural network switches. In GLOBECOM 2023 IEEE Global Communications Conference (pp. 2111-2116). IEEE.

Contents

List of Figures

List of Tables

Abstract

The rapid evolution of next-generation networks demands high performance, scalability, and security. However, conventional CPU-based systems face fundamental bottlenecks in terms of throughput, latency, and energy efficiency, which limit their ability to support 5G networking, HPC, and cybersecurity workloads. This thesis aims to evaluate the role of programmable Data Processing Units (DPUs) as a unifying hardware platform for accelerating workloads across 5G data networks, high-performance computing (HPC), and network security.

The research investigates three complementary directions. For 5G, user-plane functions such as GTP encapsulation, QoS enforcement, and DDoS mitigation are offloaded to NVIDIA BlueField DPUs using the DOCA framework. For HPC, a comparative study of BlueField-2 and BlueField-3 is conducted using RDMA micro-benchmarks to assess bandwidth, latency, and message-rate performance. For security, the thesis introduces IDS-NGNN, a nested graph neural network for intrusion detection, and proposes Physical Unclonable Function (PUF) based authentication schemes for autonomous swarm systems.

The findings show that DPUs can sustain line-rate packet processing at 400 Gbps in 5G scenarios while reducing latency and improving scalability compared with software-only UPFs. In HPC contexts, comparative benchmarking highlights clear trade-offs between BlueField-2 and BlueField-3 in terms of throughput and latency, underscoring the suitability of DPUs as data movement accelerators. In security, the combination of IDS-NGNN and PUF-based mechanisms demonstrates that AI and hardware primitives can provide robust, low-latency protection against evolving cyber threats. This thesis concludes that DPUs are a cornerstone technology for the 6G era and beyond, enabling infrastructures that are not only faster and more efficient but also inherently more secure and resilient. By bridging advances in 5G networking, HPC, and cybersecurity, the work establishes DPUs as critical enablers of future high-performance, secure digital ecosystems.

Chapter 1

Introduction

The accelerating digital transformation is creating unprecedented demand for network infrastructure that combines high performance, low latency, and robust security. Emerging technologies such as 5G mobile networks, large-scale High-Performance Computing (HPC) platforms, and cyber-physical systems operating at the network edge are pushing traditional computing and networking architectures to their limits. Conventional CPU-centric designs, though versatile, increasingly struggle to sustain line-rate packet processing at hundreds of gigabits per second, to deliver consistent performance for HPC workloads, and to defend against increasingly sophisticated cyber threats. These limitations have motivated the adoption of programmable hardware accelerators, particularly Data Processing Units (DPU)s, to rethink how communication, computation, and security functions are executed in next-generation infrastructures.

DPU represents a new class of SmartNIC that integrates programmable cores, dedicated hardware accelerators, and advanced I/O subsystems within a single device. By offloading intensive packet processing, data movement, and security operations from general-purpose CPUs, DPU can achieve higher throughput, lower latency, and improved scalability and energy efficiency. Their potential makes them especially relevant in domains where performance and security must coexist, such as 5G user-plane processing, HPC clusters, and network intrusion detection systems. The central focus of this dissertation is to examine DPU as a unifying platform across these domains, providing empirical evaluations and methodological contribu-

tions that highlight both its capabilities and limitations.

1.1 Research Scope and Vision

This thesis investigates how programmable SmartNICs and DPUs can be used to support security- and performance-critical functions in modern high-speed networks. Rather than focusing on a single application or optimization, the work adopts a system-level perspective, examining how different networking and security tasks can be distributed across heterogeneous computing resources, including host CPUs, SmartNICs/DPUs, and, where relevant, GPUs.

The unifying theme of the thesis is the placement of functionality under strict operational constraints. Modern networks must sustain very high throughput while maintaining low latency and strong security guarantees. At the same time, they must remain deployable in real infrastructures, where compute, memory, and power resources are finite. Within this context, the thesis explores how SmartNICs can act as an intermediate execution layer, absorbing infrastructure overhead that would otherwise burden the host CPU, while remaining flexible enough to support evolving workloads. Across the thesis, this pipeline can be summarized as an observe decide enforce loop: traffic is first observed and summarized, decisions are derived using lightweight learning-based methods, and enforcement is applied deterministically at wire speed using SmartNIC resources.

The thesis contributions span three closely related domains. First, it studies hardware-accelerated network security, with particular attention to DDoS mitigation and intrusion prevention executed at line rate on programmable DPUs. Second, it investigates learning-based DDoS detection using graph neural networks, focusing on how traffic structure and inter-flow relationships can be leveraged to improve robustness over single-flow approaches. Third, it evaluates SmartNIC offloading for high-performance networking workloads, such as RDMA and user-plane processing, to understand architectural limits and trade-offs that directly affect security and service deployment.

These components are not treated as isolated research problems. Instead, they

are considered parts of a common pipeline, where traffic is observed, analyzed, and acted upon under tight timing constraints. In this pipeline, complex detection logic may require global visibility and richer models, while mitigation and enforcement must operate deterministically and at wire speed. The thesis therefore emphasizes the interaction between detection accuracy, enforcement feasibility, and hardware resource constraints, which together determine whether a given approach can be adopted in practice.

The widespread adoption of digital technologies has also led to a sharp increase in cyber-attacks, causing operational disruption and significant economic damage. Reports from the Federal Bureau of Investigation (FBI) Internet Crime Complaint Center (IC3) indicate a sustained growth in cybercrime incidents and financial losses over recent years [1]. Among these threats, Distributed Denial-of-Service (DDoS) attacks are particularly disruptive, as they overwhelm network and service resources by coordinating large numbers of compromised devices, denying access to legitimate users. High-profile and politically motivated campaigns further highlight the need for scalable and deterministic defense mechanisms [2].

1.2 Problem Statement

Despite rapid advances in network hardware, CPU-centric architectures remain a fundamental bottleneck for enforcing security and performance guarantees at link speeds of 100–400 Gbps. While modern detection techniques, including machine learning models, can achieve high accuracy, they often require global context, stateful analysis, and non-trivial processing latency. In contrast, network enforcement mechanisms must operate deterministically and at wire speed. Existing research typically addresses these challenges in isolation, either focusing on detection accuracy without considering enforcement feasibility and evaluating hardware offload for a single function in isolation. As a result, there is a lack of integrated system-level studies that jointly analyze detection, enforcement, and hardware resource constraints in realistic high-speed deployments.

1.3 Research Questions

The thesis is organized around the following research questions:

RQ1 Which security and networking functions benefit most from SmartNIC/DPU offload, and what performance bottlenecks emerge when moving them from host CPU to the SmartNIC? (Chapter 3, Chapter 4, Chapter 6)

RQ2 When does graph-based DDoS detection (FTG/FTG-Net-E and related models) provide a measurable advantage over single-flow or time-series detectors, and what are the limiting cases? (Chapter 3)

RQ3 Can a programmable DPU perform DDoS mitigation at wire speed using stateful tracking and DPI, while preserving low latency under high attack rates? (Chapter 4)

RQ4 How can SmartNICs be used to accelerate security protocols (e.g., authentication in swarm/edge systems) without compromising deployability, resource isolation, and system maintainability? (Chapter 5)

RQ5 What is the real impact of BlueField-3 architectural changes (e.g., PCIe Gen5, DDR5, DPA) on RDMA latency, bandwidth, and message rate, and how well do theoretical models match measurements? (Chapter 6)

RQ6 How can GTP encapsulation and decapsulation be offloaded using BlueField-3, and what is the resulting impact on latency and throughput in 5G/6G user-plane scenarios? (Chapter 7)

1.4 Contributions and Novelty

The main contributions of this thesis are:

- A hardware-accelerated DDoS mitigation framework on BlueField that combines stateful connection tracking, queue-based DPI, and DOCA QoS integration to sustain high-rate traffic while reducing host load (Chapter 3).

- A SmartNIC-assisted authentication protocol (DPUAUT) for intelligent swarm systems, with security analysis and an explicit discussion of where SmartNIC acceleration affects latency and trust assumptions (Chapter 4).
- A graph-based DDoS detection approach based on hierarchical traffic representations (FTG) and ensemble/lightweight Graph Neural Networkss (GNNs), including empirical evaluation and analysis of inference cost (Chapter 5).
- A SmartNIC-based UPF implementation and evaluation that clarifies practical resource and latency trade-offs for 6G user-plane offload (Chapter 6).
- A performance study and theoretical model of RDMA offload on SmartNICs, including BF2/BF3 comparison, validation methodology, and service-level recommendations for SmartNIC-aware deployment (Chapter 7).

A key novelty of the thesis is the cross-layer view: detection methods are evaluated together with enforcement feasibility and SmartNIC resource constraints, rather than treating ML detection and data-plane offload as separate problems.

The following chapters build upon several technical foundations, including programmable SmartNIC architectures, learning-based DDoS detection, and hardware offload mechanisms for high-speed networking. These concepts are reviewed in Chapter 2 to provide the necessary background for the proposed methods and evaluations. Taken together, these contributions provide a unified view of SmartNICs as a practical execution layer that bridges learning-based analysis and deterministic, wire-speed enforcement in modern high-performance networks.

1.5 Thesis organization

Chapter 2 provides background and reviews on related work on network security, DDoS detection (including graph learning), SmartNIC-based authentication, UPF offload, and RDMA offload. Chapter 3 describes the design and experimental evaluation of a DPU-based DDoS mitigation framework. Chapter 4 introduces DPUAUT and its security and performance analysis. Chapter 5 presents graph-based DDoS detection models (FTG-Net-E) and the IDS-NGNN extension, together with experimental results. Chapter 6 describes SmartNIC-based UPF design and evaluation. Chapter 7 evaluates RDMA offload on BF2/BF3, validates a theoretical model, and derives deployment recommendations. Chapter 8 concludes the thesis and revisits the research questions in light of the obtained results.

Chapter 2

Background and Related Work

This chapter reviews the background and related work required to position the contributions of this thesis. It begins with the evolution of network interface cards, tracing the transition from traditional NICs to programmable SmartNICs and DPUs, and discussing their architectural differences with emerging SuperNIC designs. The review then examines how these devices are used to offload network security functions, with a focus on DDoS detection, wire-speed mitigation, and hardware-assisted authentication.

The discussion subsequently moves to SmartNIC-based acceleration for cloud and HPC workloads, covering NVMe offload, RDMA offload on BlueField platforms, and GPU packet processing pipelines. Building on this systems perspective, the chapter introduces graph neural networks and ensemble learning techniques as a foundation for learning-based network attack detection. Finally, the review surveys prior work on SmartNIC-based authentication, user-plane function offload in mobile networks, and RDMA performance evaluation.

By organizing the literature along architectural, security, and performance dimensions, this chapter highlights existing limitations and open challenges, motivating the investigation of DPUs as a potential common execution layer for networking, computing, and security in next-generation systems.

Rather than exhaustively covering existing approaches, the focus is on identifying architectural trends, performance bottlenecks, and open challenges in SmartNICs, DPUs, and learning-based network security. These insights directly motivate the

system designs and experimental studies developed in Chapters 3 through 7.

2.1 Network Interface Cards (NICs)

Traditional **Network Interface Cards (NICs)** are hardware components designed to enable network connectivity by managing basic packet transmission and reception. Early NICs primarily served as simple network adapters, forwarding packets between the network and the host system. Over time, modern NICs have evolved to include limited forms of hardware acceleration, such as checksum computation, packet segmentation, and basic offloading capabilities. Despite these improvements, conventional NICs remain constrained in terms of programmability and adaptability. They are typically unable to accommodate the growing complexity and dynamic requirements of contemporary high-speed networks, where flexible and intelligent packet processing is increasingly necessary.

2.2 SmartNICs

The emergence of **Smart Network Interface Cards (SmartNICs)** addresses many of the limitations inherent in traditional NICs. SmartNICs incorporate programmable processing capabilities, often leveraging Field Programmable Gate Array (FPGA), Application Specific Integrated Circuit (ASIC), or Arm-based System on Chip (SoC) architectures, to offload network, storage, and security functions from the host processor. A SmartNIC represents a significant evolutionary step in networking technology, combining heterogeneous domain-specific accelerators with general-purpose cores to efficiently handle infrastructure tasks [3].

Based on their underlying hardware architecture, SmartNICs are typically categorized as ASIC-, FPGA-, or SoC-based [4]. Generally, performance and cost decrease as we move from ASIC- to SoC-based designs, while programmability and ease of development increase. Among these categories, SoC-based SmartNICs strike the best balance between price, performance, and programmability. Therefore, in this work, we focus exclusively on SoC-based SmartNICs, which we refer to simply

as SmartNICs for brevity.

From Traditional NICs to SmartNICs

As illustrated in Fig. 2.1(a), conventional NICs primarily handle packet transmission and reception using Direct Memory Access (DMA) to move packets between the network and host memory, where the host Central Processing Unit (CPU) performs higher-level processing. In addition, they may offload certain repetitive operations, such as checksum computation or error detection, to hardware.

More advanced NICs, including Mellanox ConnectX-4 and ConnectX-5 adapters, extend these capabilities by supporting frameworks like eXpress Data Path (XDP), eBPF, and Data Plane Development Kit (DPDK), which improve packet-processing efficiency in both kernel and user space.

By contrast, a SmartNIC, shown in Fig. 2.1(c), integrates a programmable multi-core SoC running a full-fledged Operating System (OS). This architecture enables flexible programmability and allows infrastructure functions to execute directly on the SmartNIC without involving the host CPU. The intelligence of these devices lies in their ability to operate autonomously, implementing a wide range of tasks from packet filtering and encryption to flow management while remaining adaptable to evolving network demands.

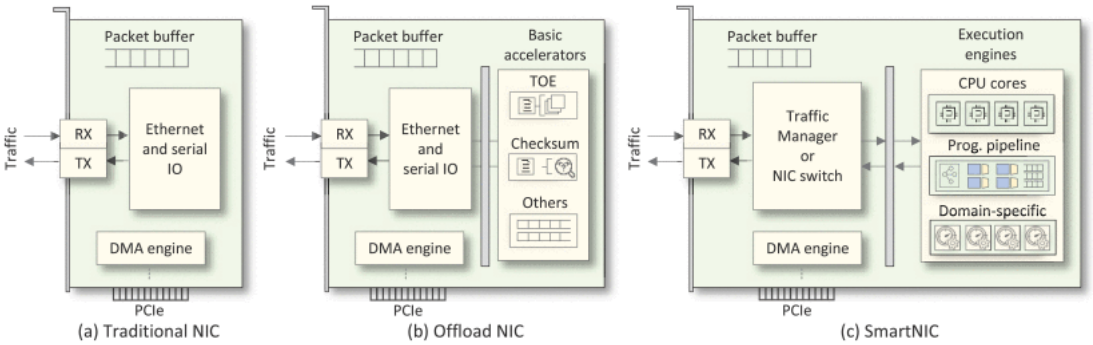


Figure 2.1: Main functional blocks of traditional NICs (a), Offloaded NIC (b), and SmartNIC (c) [3].

In addition to general-purpose processing units, SmartNICs incorporate specialized hardware accelerators that vendors provide to enhance specific tasks, such as packet filtering, encryption, and regular expression matching. These domain-specific

processors achieve high performance for targeted functions and exhibit significantly greater energy efficiency than general-purpose CPU. These hardware accelerators can process network traffic directly, reducing reliance on the SmartNIC CPU, which primarily orchestrates execution rather than performing intensive processing.

Compared with traditional NICs, SmartNICs do more than provide high-speed network connectivity. They also offload infrastructure tasks that were previously handled by the host server. This capability enables applications such as zero-trust security, telemetry, and advanced networking to run within the SmartNIC, thereby conserving valuable compute resources on the host. SmartNICs improve network efficiency, lower operational costs, and enhance security by reducing CPU overhead.

2.3 SuperNICs: The Next Evolution

The advent of artificial intelligence (AI) and hyperscale cloud computing has driven further advances in network acceleration. **SuperNICs** represent a new class of network accelerators designed explicitly for AI workloads, enabling high-performance GPU-to-GPU communication in Ethernet-based cloud infrastructures.

The key attributes of SuperNICs include:

- **Ultra-fast network speeds:** Achieves up to 400 Gbps using Remote Direct Memory Access (RDMA) over Converged Ethernet (RoCE) technology.
- **Packet reordering:** Ensures sequential integrity of data packets when integrated with high-performance network switches.
- **Advanced congestion control:** Utilizes real-time telemetry and network-aware algorithms to mitigate congestion in AI networks.
- **Programmable compute:** Supports customization through I/O path programmability, enabling optimized AI cloud infrastructure.
- **Power efficiency:** Designed to operate within constrained power budgets, making it ideal for large-scale AI workloads.

2.4 SuperNIC vs. SmartNIC

While SmartNICs share some functional similarities with SuperNICs, they serve distinct roles. SmartNICs offload general network, security, and storage workloads from the CPU, whereas SuperNICs are explicitly optimized for AI-driven networking. While NVIDIA BlueField devices share some characteristics with SuperNICs, they are positioned as general-purpose DPUs rather than AI-specialized network adapters.

The key differences include:

- **Purpose:** SmartNIC focuses on general network acceleration, while SuperNICs specialize in AI networking.
- **Performance:** SuperNICs deliver high-throughput, low-latency communication for AI workloads, scaling efficiently at 400 Gbps per GPU.
- **Efficiency:** Unlike SmartNIC, which consumes substantial computational resources, SuperNICs achieve AI networking acceleration with lower power consumption.

In this thesis, SuperNICs are considered primarily as a reference point for AI-centric networking, while the focus remains on DPUs and SmartNICs as general-purpose offload platforms for security and data-plane processing.

2.5 Graph Neural Networks

Several kinds of objects are defined in terms of connections to other things and are thus naturally expressed in a graph form. Molecules, social networks, and citations are just some examples of structures that are naturally described as graphs. While there is an obvious interest in exploiting machine learning on this kind of data, standard neural networks cannot be used to learn on graphs, as they are made to work on arrayed-structured data. For this reason, in 2008 Scarselli et al. [5] introduced graph neural networks, a class of deep learning models developed to

operate on graph-structured data. Graph Neural Networks (GNNs) have gained a rapid interest, and are now successfully applied in several areas such as physics simulations, recommendation systems, antibacterial discovery, and networking [6].

The field of GNNs is vast, and graph networks can differ considerably. As a general consideration, we can define a GNN as a permutation invariant optimizable transformation on graph attributes (nodes, edges, and global parameters) [7]. The permutation invariant attribute refers to the fact that GNNs preserve symmetries. The prediction problems tackled by these networks can be divided in three categories: (i) graph level tasks where predictions try to discover a property of the entire graph (e.g., molecule toxicity), (ii) edge level tasks where predictions concerns the relations between nodes (e.g., reccomendation systems), and (iii) node level tasks where predictions try to find the identity or role of each node of the graph.

2.5.1 Message Passing Neural Networks

The first kind of GNN is the Message Passing Neural Network (MPNN). Let $G = (V, E, U)$ be a graph, where V is the set of vertices (nodes), E is the set of edges, and U is the global features. Consider now a simple problem of node representation learning, where each vertex $v \in V$ has a set of features x_v, represented by a vector of dimension n, namely the hidden state h_v^0. In each iteration t of the process, a message-passing operation occurs, where the hidden states of the nodes are combined with the hidden state of their neighbors (e.g., through addition). This message-passing operation involves two functions: an aggregation function $a(\cdot)$ that consolidates the received hidden states (e.g., through addition), and an update function $u(\cdot)$ that merges the current hidden state of the node with the result of the aggregation (usually employing a differentiable model such as a multilayer perceptron).

At iteration t, the message-passing operation is defined formally as follows:

$$m_{v,w} = m(h_v^t, h_w^t, e_{v,w}) \tag{2.1}$$

$$M_v^{t+1} = a(\{m_{v,w} | w \in N(v)\}) \tag{2.2}$$

$$h_v^{t+1} = u(h_v^t, M_v^{t+1}) \tag{2.3}$$

here hidden state of node v at iteration t is denoted by h_v^t. $e_{v,w}$ is the edge which connects nodes v and w. The message sent from w to v is represented by $m_{v,w}$, whereas the aggregated message received by v is denoted by M_v^t. The neighborhood of v (which comprises all vertices adjacent to v) is represented by $N(v)$.

It is feasible to depict the complete graph as a single vector following the execution of T iterations. This can be accomplished by utilizing a permutation invariant function that operates on all the concealed states of the nodes during a readout phase.

$$y = r(h_v^T | v \in V) \tag{2.4}$$

where y is the final representation vector; r is the permutation invariant function. Finally, y is used to make the final prediction.

2.5.2 Graph Convolutional Networks

First proposed in 2017 by Kipf and Welling [8], Graph Convolutional Networks (GCN) models are graph networks that rely on graph convolutions to support scalability and fast semi-supervised classification of nodes. GCNs propagate information across the graph through a message-passing mechanism and update the node features based on the information in its neighborhood only. This process is repeated for several layers, allowing the network to capture increasingly complex relationships between nodes. The message-passing mechanism in GCNs closely resembles the convolution operation in CNNs, and hence the name. In CNNs, the convolution operation updates an element according to the patterns from its neighbours in an arrayed structure. Similarly, in GCNs, the message-passing mechanism aggregates information from a node's neighbors, effectively capturing local patterns within the graph structure.

The following equations describe the GCN layer update rule:

$$H^{(l+1)} = \sigma(\tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}H^{(l)}W^{(l)}) \tag{2.5}$$

Where $H^{(l)}$ and $H^{(l+1)}$ are the input and output feature matrices of the GCN layer, respectively; $\tilde{D}$ represents the degree matrix of the graph, which is a diagonal matrix where the entries correspond to the number of neighbors for each node; $\tilde{A}$ denotes the normalized adjacency matrix of the graph, which is obtained by adding self-connections to the adjacency matrix A and normalizing it using the square root of the degree matrix $\tilde{D}$; $W^{(l)}$ is the trainable weight matrix of the GCN layer; σ is a non-linear activation function, such as a sigmoid or Rectified Linear Unit (ReLU). The update rule can be interpreted as a weighted average of the node and of its neighbours.

The node-wise perspective of the GCN update rule is described by the following:

$$h_i^{(l+1)} = \sigma\left(\sum_{j \in N(i)} c_{ij} W^{(l)} h_j^{(l)}\right) \tag{2.6}$$

Where $h_i^{(l)}$ and $h_i^{(l+1)}$ are the input and output representations of node i or $i+1$ at layer l. $N(i)$ is the set of neighbors of node i. c_{ij} is a normalization factor defined as $c_{ij} = \frac{1}{\sqrt{|N(i)|}\sqrt{|N(j)|}}$. The node-wise update rule can be interpreted as a weighted average of the node's representation and the representations of its neighbours, where the weights are determined by the normalization factor c_{ij}.

2.5.3 Graph Attention Networks

Graph Attention Network (GAT)s are a type of GNN that uses an attention mechanism to learn the importance of different neighbors for each node in the graph. GATs were first proposed by Velickovic et al. in 2017 [9], and have since become one of the most popular GNN architectures. These attention networks work by first computing an attention weight for each pair of nodes using a learnable function that takes the node features (and edge features if available) as input. The attention weights are then normalized to sum to 1 for each node and used to update the node features. As a result, the updated node features are a sum of the features of the node neighborhood, weighted by the attention weights. The following equation describes the GAT layer update rule:

$$\mathbf{h}_i^{(l)} = \sigma\left(\sum_{j \in \mathcal{N}_i} \alpha_{ij} \mathbf{W}^{(l)} \mathbf{h}_j^{(l)}\right) \tag{2.7}$$

Where $\mathbf{h}_i^{(l)}$ is the feature vector of node i at layer l; σ is a non-linear activation function, such as ReLU; $\mathcal{N}_i$ is the set of neighbors of node i; α_{ij} is the attention weight between nodes i and j; $\mathbf{W}^{(l)}$ is the weight matrix at layer l.

2.5.4 Ensemble Learning

Ensemble learning is a machine learning technique that combines multiple models' outputs to improve the overall prediction accuracy. It has emerged as a powerful tool for DDoS detection, where attackers are constantly developing new and sophisticated methods. One of the key advantages of ensemble learning is that it can leverage the strengths of different models to capture diverse patterns in the data. This is akin to assembling a team of experts with different backgrounds and perspectives to solve a problem. Just as a team of experts is likely to outperform any individual expert, an ensemble of models will likely outperform any individual model. Another advantage of ensemble learning is that it reduces the risk of overfitting: ensembles are less prone to over-learn the specific features of the training data and better generalize [10].

GNNs based ensemble learning combines the predictions of multiple graph networks to produce a more robust and accurate prediction. This is done by training multiple GNNs on different subsets of the data, and then averaging the predictions of the individual models.

Among the various ways to ensemble neural networks, a common approach is to use a weighted average of the predictions of the individual models [11]. The weights can be assigned based on the performance of the individual network on the training data, or they can be learned using a hyperparameter tuning algorithm. Another approach to ensemble GNNs is to use a stacking technique, where the individual model predictions are used as input to a meta-learner. The meta-learner combines the predictions of multiple models instead of only weighted averages. The meta-learner is then trained to predict the final output.

The following equations describe the ensemble method for a GNN with equal

weights:

$$\hat{y}_i = \frac{1}{M} \sum_{m=1}^{M} g_m(\mathbf{h}_i^{(l)}) \tag{2.8}$$

Where $\hat{y}_i$ is the predicted output for node i; M is the number of GNNs in the ensemble; g_m is the mth GNN in the ensemble; $\mathbf{h}_i^{(l)}$ is the feature vector of node i at layer l.

2.6 Network Security Function Offloading Using SmartNICs

Modern network security relies on ensuring the CIA triad: confidentiality, integrity, and availability of data in transit. While encryption and authentication protect confidentiality and integrity, ensuring availability remains challenging, particularly against volumetric and protocol-level attacks such as Distributed Denial of Service (DDoS).

2.6.1 DDoS Attack Detection and Mitigation Strategies

DDoS attacks are a prevalent cyber threat characterized by a malicious attempt to overwhelm a target's online resources, rendering them unavailable to legitimate users. These attacks can be categorized into three primary types, each with unique characteristics and implications.

1. Volumetric Attacks: Flooding the Network with Legitimate Traffic - Volumetric DDoS attacks flood the target network with an overwhelming volume of legitimate traffic, typically generated by a botnet, a network of infected devices controlled by an attacker. This massive influx of traffic saturates the network's bandwidth, choking it to a standstill. As a result, legitimate users cannot access the target website, application, or service.

2. Protocol Attacks: Exploiting Layer 3 and 4 Vulnerabilities - Protocol attacks target network protocols operating at Layer 3 (network layer) and 4 (transport

layer). These attacks aim to consume the target server's resources and those of intermediate communication equipment, such as firewalls and load balancers, rendering them unable to handle legitimate traffic. Common protocol attacks include SYN flood attacks, UDP flood attacks, and ICMP flood attacks.

3. Application Attacks: Exploiting Vulnerabilities at Layer 7 - Application layer attacks target the application layer (Layer 7), the highest layer of the network stack. These attacks exploit legitimate requests to consume computing resources, such as database queries or file reads. Unlike volumetric and protocol attacks, application-layer attacks may be more challenging to detect as they often blend in with legitimate traffic.

A comprehensive detection system is crucial for effectively detecting and mitigating DDoS attacks. This system should be able to analyze both aggregated traffic patterns and specific traffic flows to identify anomalies that may indicate an attack. Analyzing aggregated traffic provides insights into overall network behavior, while examining particular traffic flows helps identify suspicious activity between two endpoints.

Combining these perspectives allows a flexible detection system to distinguish between legitimate traffic and malicious attacks. The DDoS detection system must reach a trade-off between (i) the delay due to massive receiving traffic under analysis (to deliver an adequate overview to monitor probable malicious patterns) and (ii) the reactivity to implement appropriate mitigation and security rules before consequential damages.

Moreover, DDoS mitigation strategies should also account for the overall network's computational capacity. Overburdening the network with intensive processing tasks can lead to performance bottlenecks, hindering the ability to respond to other network traffic.

Detection of DDoS attacks using traditional Machine Learning (ML) and Deep Learning (DL) techniques involves using traffic-level or flow-level features that are statistical representations of the data associated with transmitted packets within a given flow or time window. These features are significant in identifying potential

attacks. Those approaches perform well when trained on well-known Intrusion Detection System (IDS) datasets. Still, these IDS are not widely adopted in real-time systems due to their inability to generalize and adapt to diverse traffic profiles and network conditions [12].

To tackle this issue, we can shift from statistical aggregation to a deeper analysis of traffic flow structure. This task involves examining sequences of exchanged packets between two endpoints and considering aggregated traffic, defined as the collection of flows established among endpoints within a specific time window.

This approach, augmented topological knowledge, enables the detection of common structural patterns in specific types of DDoS attacks. A recent breakthrough has emerged with the application of GNN to the problem of DDoS [13]. This approach harnesses topological information to enhance both robustness and detection accuracy. However, analyzing intra- and inter-entity flow-level connections is essential for capturing all potential attack patterns.

Recent research has explored graph-based representations of network traffic to improve DDoS detection robustness. Hierarchical traffic graphs have been proposed to jointly capture flow-level and aggregated traffic structures, enabling models to exploit both local and global patterns. Ensemble learning techniques further improve robustness by combining multiple lightweight graph models trained on different traffic substructures. These approaches show promise in improving generalization and resilience to evolving attack strategies, but raise open questions regarding inference cost and real-time deployability.

2.6.2 Wire-Speed DDoS Attack Detection

Network security has become a critical issue due to the increasing reliance on the Internet for various aspects of daily life [14], [15]. Network attacks, such as DDoS attacks, malware, and botnets, constantly threaten network security [16]. Intrusion detection systems (IDSs) and intrusion prevention systems (IPSs) are essential tools to detect and prevent these attacks [17]. Traditional software-based IDSs and IPSs struggle to analyze network traffic in real-time due to rising speeds and volumes

[18]. Recent studies explored hardware-accelerated solutions for real-time intrusion detection and DDoS mitigation[17]. For example, ThunderSecure leverages stream-based features and one-class classification networks to achieve high detection rates and low false positives on 100 Gbps networks [19]. While many previous works have explored the integration of FPGAs and SmartNICs with IDS/IPS processing [20]–[26], these efforts predominantly focused on offloading specific functionalities, such as regular expression matching, to the FPGA/SmartNIC. However, these solutions fail to achieve 100 Gbps processing within a single server. For example, even with optimized regular expression matching, Snort 3.0 achieves only 400 Mbps per core, requiring 250 cores to sustain line rates. Recent DPU platforms enable the offloading of selected IDS/IPS primitives, such as traffic filtering and DPI, to programmable network devices, offering an alternative to FPGA-centric designs.

DPUs are programmable and scalable, and eliminate specialized hardware designs like FPGAs, making them a cost-effective, high-performance alternative for 100 Gbps IDS/IPS processing.

In this context, recent work has explored hardware-accelerated frameworks for live traffic analysis and attack mitigation using SmartNICs, integrating DPDK-based packet processing, regular-expression matching, and open-source IDS engines such as Suricata.

Building on this architectural foundation, the remainder of the chapter shifts focus from capability to application. In particular, it examines how these programmable network devices are leveraged to support security-critical and performance-sensitive workloads, where strict latency, throughput, and resource constraints apply. This transition motivates the discussion of network security mechanisms, learning-based attack detection, and hardware-assisted enforcement, which are addressed in the following sections.

Network security has become a critical issue due to the increasing reliance on the Internet for various aspects of daily life. Various types of network attacks, such as DDoS attacks, malware, and botnets, constantly threaten network security. Intrusion detection systems (IDSs) and intrusion prevention systems (IPSs) are essential

tools to detect and prevent these attacks [27].

However, with the increase in network speeds and traffic volumes, traditional software-based IDSs and IPSs face significant challenges in real-time processing and analyzing network traffic. Traditional intrusion detection systems (IDSs) face several challenges, such as high false positive rates and limited scalability, which hinder their effectiveness in detecting modern cyber threats. Scalability concerns at the switch level have been tackled partially by introducing data plane programmability with P4-based feature extraction [28] and in-network machine learning pipelines [29]. At the smart NIC level, the paper [30] ONLAD-IDS presents an innovative approach that uses SmartNIC and the Online Aggregation (ONLAD) algorithm to perform intrusion detection. However, the paper has some limitations, including limited evaluation of hardware utilization, lack of comparison with state-of-the-art IDSs, and scalability. To address this challenge, hardware-based IDSs, and IPSs must be developed to offload network traffic processing from the CPU to specialized hardware devices such as smart network interface cards (NICs). The paper [23] proposes an FPGA-based approach for intrusion detection in cloud systems, which offloads detection tasks to FPGA-based SmartNICs for improved efficiency and reduced overhead. This model achieves high throughput, low latency, and high detection accuracy while providing benefits such as improved scalability and reduced power consumption. Although the paper has limitations, such as limited evaluation and lack of comparison with other FPGA-based approaches, it presents an innovative solution to the limitations of software-based intrusion detection in cloud systems. Further research and development could significantly enhance cloud systems and data center security. Another paper [31] presents an interesting approach to intrusion detection using extended Berkeley Packet Filter (eBPF) technology in the Linux kernel, but some potential limitations should be considered. Firstly, the effectiveness is decreased on larger networks with higher traffic volumes, which could limit its scalability. Secondly, false positives and negatives are not addressed. Finally, the paper does not evaluate the potential overhead of the system on system resources, such as CPU and memory usage. These limitations should be considered when considering the

feasibility and applicability of the proposed system in real-world scenarios.

The paper [32] proposes a real-time network intrusion detection system that uses a deferred decision and hybrid classifier approach to improve detection accuracy and reduce false alarms. The authors provide a comprehensive overview of the system architecture and implementation details, and the experimental results show promising results on a publicly available dataset. However, the paper could be improved by providing a more detailed comparison with existing IDSs, discussing limitations and potential future directions, and evaluating the system's effectiveness on other datasets and real-world scenarios. ThunderSecure [19]: deploying real-time intrusion detection for 100G research networks by leveraging stream-based features and one-class classification network" by Gong et al. presents a novel intrusion detection system (IDS) called ThunderSecure that achieves high detection rates and low false positives on a real-world 100Gbps research network. A potential limitation is that ThunderSecure is only evaluated on a single network and is not compared to other IDS solutions.

Baresellotti et.al [33] an idea to highlight the potential benefits of using DPUs to enable edge computing, which can help mitigate the impact of DDoS attacks. Edge computing allows for data processing closer to the source, reducing the dependence on centralized cloud servers. By distributing processing power across multiple DPUs at the edge, organizations can better handle the impact of DDoS attacks on their central servers. For example, if a data center and cloud system are under a DDoS attack, the traffic can be diverted to a DPU at the edge, where it can be analyzed and filtered for malicious traffic. This can help reduce the load on the central server and prevent the DDoS attack from affecting legitimate traffic. While the paper does not discuss DDoS attacks in depth, it does highlight the potential benefits of using DPUs at the edge to enable edge computing, which can help mitigate the impact of DDoS attacks on centralized cloud servers.

2.6.3 Machine Learning-based DDoS Attacks

Previous work to detect DDoS attacks through ML and DL have concentrated on balancing detection performance with introduced latency. This is because these methods typically utilize complex models that require significant computational resources to train and execute, leading to increased latency. Hence, these limitations must be addressed to enable a real-time application of ML/DL based DDoS attack detection. One promising approach is using GNNs, a neural network designed to work with graph data. The graph represents relationships between entities, such as the relationships between nodes in a network. GNNs are effective for various graph data tasks, such as fraud detection and social network analysis [34]. In this section, we evaluate some GNN-based DDoS attack detection techniques, including ensemble-based GNN approaches.

2.6.4 DDoS Attack Detection

Anomaly-based detection methodologies are used to detect and identify DDoS attacks by accumulating enough data [35], [36], [37]. Over the past few years, significant research has focused on designing and implementing high-performance IDS/IPS solutions. These systems are typically based on signature-based or anomaly-based approaches [23].

[38] optimized Snort 3.0 IDS using DPDK to improve detection rates and address performance issues. They introduced an alternative packet polling technique to reduce CPU interruptions and memory operations. However, the study lacks an evaluation of scalability, complex traffic, and multi-threaded efficiency.

[39] conducted a comparative analysis of three widely used intrusion detection systems (IDS)Snort, Bro, and Suricata. They examined resource utilization, packet processing capacity, and packet drop rates that limit network intrusion detection systems (NIDSs) from handling high traffic volumes in large-scale networks. They highlighted the limitations of default configurations, detection method optimizations, and DAQ (Data Acquisition) configuration modifications in improving capture performance. Their findings indicated that specific pattern-matching algorithms could

mitigate traffic volumes but high CPU utilization. The Suricata CPU usage exceeded 93% when processing traffic above 10 Gbps with only 20 flows. Furthermore, Snort and Zeek (formerly Bro) could not sustain high detection rates at traffic speeds exceeding 5-7 Gbps due to their reliance on software-only processing.

In their follow-up research, [40] addressed the scalability challenges of open-source IDS systems in high-speed networks. By leveraging a transparent IDS with eXpress Data Path (XDP), they achieved a maximum data transfer rate of 60 Gbps with Suricata. However, once the data rate reached this threshold, packet drop rates increased as Suricata started discarding packets due to the bottleneck. This is a limitation in software-only architectures for real-time detection and prevention in high-speed networks.

[41] presented DPI library named Libprotoident that identifies application layer protocols for network flows. Unlike other methods that require capturing the entire packet payload, Libprotoident utilizes the first four bytes of the payload transmitted in each direction, the size of the first packet containing the payload in each direction, and the TCP or UDP port numbers relevant to the flow. However, Libprotoident processing is performed by the CPU without hardware offloading or connection tracking, which limits its scalability for high-speed traffic analysis.

[42] presented the nDPI library, an application-layer detection of protocols independent of the port being used. This allows for the identification of known protocols on non-standard ports and the detection of unknown protocols. However, iterating over many protocol dissectors increases CPU cycles and causes branch mispredictions. The nDPI maintains a per-flow state (e.g., protocol detection status, metadata) that often involves pointer chasing and frequent, small memory allocations. However, dynamic per-flow state management leads to cache and TLB misses. Finally, the intensive payload scanning for pattern matching further strains the CPU. These factors in nDPI limit throughput in high-speed networks.

[43] presented OpenDPI, which leverages behavioral and statistical analysis to infer transmission types, but its low-level processing faces bottlenecks. Specifically, computing detailed statistical indicators for each flow introduces significant CPU

overhead, mainly when processing large traffic volumes. Because OpenDPIs approach depends on access to unencrypted payload data, its performance depends on encryption, which limits the effectiveness of signature extraction [44].

[23] proposed an FPGA-based approach for intrusion detection in cloud systems, which offloads detection tasks to FPGA-based SmartNICs for improved efficiency and reduced overhead. This model achieves high throughput, low latency, and high detection accuracy but limits the number of flows that can be processed.

Wang et al. [31] used extended Berkeley Packet Filter (eBPF) technology in the Linux kernel to overcome the limitation of a number of flows. However, eBPF-based solutions struggle in larger networks with higher traffic volumes, which could limit their scalability.

Many DPI methods have also been proposed to classify network traffic, but these solutions are typically resource-intensive and fail to achieve consistent performance at high speeds, especially in scenarios requiring live traffic analysis and mitigation of large-scale DDoS attacks.

There are numerous proposed methods and frameworks for detecting DDoS attacks. This subsection emphasises the ML and DL based approaches. These techniques have proven effective in identifying various DDoS attack types, such as volumetric attacks, protocol attacks, and application-layer attacks.

Previous work on DDoS attack detection has also explored hardware acceleration techniques to improve detection performance and reduce latency. Musumeci et al. [28] introduced a Software Defined Networking (SDN) based DDoS detection system. In this system, the DDoS attack detection module receives stateful traffic information based on a sliding window from the system. They utilized Barefoot Tofino switches enabled with P4, which offload a part of the detection process to the data plane. Moreover, the recent work of De Marinis et al. [45] demonstrated that it is possible to onboard the logic of a DNN inside a programmable switch pipelines made of a cascade of flow tables. The paper evaluated the mapping and the onboarding of a P4-based DDoS detector DNN applied at the packet level [46], achieving a F1-score above 92%. This approach, however, assumes the availability

of specialized programmable hardware in the network operator ecosystem.

The LUCID technique, as proposed by Doriguzzi-Corin et al. [47] is employed to detect DDoS attacks, offering lightweight execution with minimal processing overhead and detection time. The distinctive traffic preprocessing mechanism is designed to feed the CNN model with network traffic for online DDoS attack detection. Their evaluation compared LUCID with DeepDefense 3Long Short-Term Memory (LSTM) across multiple datasets, including ISCX2012, CIC2017, CSECIC2018, and UNB201X, yielding comparable results. Notably, LUCID outperformed 3LSTM in terms of detection time. Additionally, LUCID was compared with other contemporary methods, including DeepGFL, Multilayer Perceptron (MLP), LSTM, 1D-Convolutional Neural Network (CNN), and 1D-CNN LSTM, with validation on the CIC2017 dataset. The evaluation results demonstrated that LUCID achieves good performance from existing state-of-the-art techniques. The study verified that LUCID effectively learns domain information by calculating kernel activations for each feature. Remarkably, the training time of LUCID on a GPU development board was 40 times faster than the authors' implementation of DeepDefense 3LSTM. However, LUCID has not investigated the case of small flows (i.e., mouse flows with limited bytes exchanged between endpoints) with padding mechanism, which may suffer from scalability issues.

2.6.5 Graph Neural Networks based Attacks Detection

GNNs help to identify potential anomalies by analyzing the flow of each feature in the dataset. [48]. The GNN looks promising in different cyber security fields, such as threat intelligence, vulnerability, and malware detection [49]–[52]. However, the detection of DDoS needs to be correctly implemented to get the full benefit of GNN nature using hierarchical traffic filtering and tackle adversarial attacks. The previous solutions[51], [52] for attack detection used measuring each flow or multiple sketches for independent flows without hierarchical traffic filtering. It is essential to detect elephant flows and aggregate mice flow to improve detection and reduce computational cost [53]. The ensemble-based intrusion detection enhances overall

robustness against adversarial attacks since combining GNN models present different sensitivities to attacks [54].

Several recent studies have explored the use of self-supervised GNNs for various tasks, including Network Intrusion Detection System (NIDS), Knowledge Tracing (KT), and graph-based clustering. Pujol-Perich et al. [55] highlighted the importance of constructing a graph representation of network flows to unveil meaningful structural patterns for developing robust and accurate NIDS. They articulate network flows and their interconnections within the network using a graph structure. Subsequently, a message-passing function is introduced to effectively glean insights from host connection graphs. The model's efficacy is assessed using the CIC-IDS2017 dataset and is benchmarked against various machine learning classifiers. The evaluation involves artificially modifying flow features relevant to the attack scenarios, such as packet size and inter-arrival times. The results demonstrate that the proposed GNN model maintains baseline accuracy compared to other models that exhibit degraded performance when exposed to altered traffic flows.

The study conducted by Li et al. [56] introduced the GraphDDoS model, designed to detect both low-rate and high-rate DDoS attacks by considering the relationships among packets within a single flow and between different flows. In pursuit of this goal, network packets with the same source and destination IP addresses are grouped, forming the basis for constructing an endpoint graph as the final representation. Message-passing operations are then executed on these graphs using a Graph Isomorphism Network (GIN), and a readout function computes the resulting graph embeddings. The efficacy of the model is evaluated on the CIC-IDS2017 and CIC-DoS2017 datasets.

Song et al. [57] proposed a self-supervised GNN model for KT called Bi-CLKT. Bi-CLKT learns node and graph embeddings using positive and negative pair graph training. The authors showed that Bi-CLKT outperforms state-of-the-art KT models.

Guo et al. [58] introduced GLD-Net, a model adept at integrating topological structure and traffic features. The approach involves segmenting traffic data into

time slots and constructing a subgraph for each slot. Topology information is incorporated as node features in these subgraphs, while flow statistics serve as edge features. The subgraphs undergo processing using a GAT, allowing simultaneous analysis of both traffic and topological features. The resulting outputs are then interpreted as a time series, inputting a LSTM) network.

Yang et al. [59] proposed another self-supervised GNN model for graph-based clustering called Variational Co- embedding Learning Model for Attributed Network Clustering (VCLANC). This model uses node embeddings and attributes to learn local and global mutual affinities between graph elements. The authors showed that VCLANC outperforms other graph-based clustering methods.

Lo et al. [60], proposed E-GraphSAGE, a GNN-based IDS for the Internet of Things (IoT). The authors argue that GNNs are well-suited for IDS because they can learn the complex relationships between the devices and nodes in an IoT network. The proposed E-GraphSAGE model is based on the GraphSAGE algorithm [61], a GNN that can learn node representations in a graph. E-GraphSAGE extends GraphSAGE by incorporating edge features and topological patterns into learning. This makes E-GraphSAGE more effective at detecting malicious network flows in IoT networks. E-GraphSAGE has been tested on four benchmark IoT NIDS datasets, outperforming other state-of-the-art models on all four datasets. It is the first successful, practical, and extensively evaluated approach to applying GNNs to the problem of network intrusion detection for IoT using flow-based data. Their study observed a significant enhancement in botnet detection performance by utilizing their proposed model. This improvement was particularly evident when comparing the outcomes achieved by logistic regression and the pre-existing botnet detection tool, BotGrep [62]. Alshammari et al. [63] proposed a parameter-free graph reduction method for graph-based clustering. The authors showed their approach can achieve competitive performance with parameter-based graph-based clustering methods without dataset-specific parameter tuning.

Xiao et al. [64] proposed a traditional graph embedding approach to perform network intrusion detection. The first step they took was to convert the network

flows into graphs based on the source and destination IP and port pairings. After that, they used traditional graph embedding techniques like DeepWalk (skip-gram) for network intrusion detection. However, there is a significant drawback to this approach: it utilizes conventional transductive graph embedding methods [65] which are not capable of generalizing to unseen node embeddings, such as IP addresses and port numbers that were not included during the training phase. This means that the approach is not suitable for most practical NIDS application scenarios, as it cannot rely on every IP address and port pair appearing in the training data. This means that the model cannot detect new attacks that use IP addresses or port numbers that the model has never seen before. This is a major limitation, as attackers constantly develop novel attacks and techniques.

In contrast, Anomal-E [66] is a self-supervised GNN model that does not require labeled data to train and can be generalized to unseen node embeddings. This makes Anomal-E a more promising approach to network intrusion detection, as it can detect known and unknown attacks. Anomal-E has been developed for network intrusion and anomaly detection. It incorporates and leverages edge features to learn the complex relationships between different devices and nodes in a network. This allows Anomal-E to detect malicious network activity, even if it has never seen that type of attack before. Anomal-E has outperformed baseline methods on two benchmark NIDS datasets regarding accuracy and generalization capability. This demonstrates the potential of Anomal-E to improve the security of networks by more effectively detecting known and unknown attacks. In addition to its performance, it offers several other advantages over traditional NIDS approaches. First, it is a self-supervised model, meaning it does not require labeled data to train, a significant advantage in network intrusion detection, where labeled data is scarce and expensive. Second, Anomal-E can leverages edge features, providing additional information about the relationships between different devices and nodes in a network. This information is essential for detecting sophisticated attacks that exploit vulnerabilities in the network topology. Despite these advances, many graph-based intrusion detection systems remain computationally expensive or insufficiently inte-

grated with hardware offload mechanisms, limiting their applicability in high-speed production networks.

These limitations motivate hierarchical and flow-aware graph constructions that balance detection accuracy with inference cost, which are further explored in Chapter 5.

2.6.6 Ensemble GNN Learning

GNN models are a powerful tool for learning from graph-structured data. They are used in various applications, including social network analysis, recommendation systems, and computer vision [67]. However, deep learning models can be complex and computationally expensive to train, and they can overfit the training data leading to poor performance on unseen data. Despite these challenges, GNNs have the potential to improve the performance of intrusion detection systems significantly.

Hou et al. [68] proposed a novel approach to aspect-level sentiment classification that leverages the power of GNNs and ensemble learning. Aspect-level Sentiment Classification (ALSC) is a promising solution that uses GNNs to construct multiple dependency trees for an input sentence, each of which captures a different aspect of the sentence's syntactic structure. Some graph networks are then applied to each dependency tree to learn representations of the words and phrases in the sentence. These representations are combined using an ensemble learning technique to predict the aspect-level sentiment. This approach has been shown to outperform state-of-the-art methods on several benchmark ALSC datasets. For example, in the SemEval 2014 Task 4 dataset, the graph ensemble learning approach achieved an F1-score of 88.45%, compared to 85.62% for the best baseline method.

Wei et al. [69] proposed a new GNN-Ensemble approach to address the challenges of complexity, computational expense, and overfitting in intrusion detection systems. This approach reduces the overall complexity and computational cost of training while also improving the robustness of the model to overfitting. It randomly constructs multiple substructures from the input graph and trains a GNN on each substructure. The predictions of the individual models are then combined

using a weighted voting scheme. The authors evaluate GNN-Ensemble on several benchmark datasets and show that it outperforms other GNN-based methods in accuracy and generalization. While it is a promising new approach, it has some limitations. First, implementing it is more complex, as it resorts on multiple models. Second, ensemble GNNs may require more training data than single ones as they need to train multiple models, each of which is trained on a different subgraph of the network traffic data. Additionally, ensemble models can be more sensitive to the data quality, so it is vital to ensure that it is well-labeled and representative of the real-world data on which the model will be deployed [70]. Finally, it is still open to research how best to design the substructures and voting scheme for different tasks.

Wang et al. [71] proposed a novel GNN-based NIDS approach called Spatio-Temporal Graph Attention Network (N-STGAT). It leverages a graph attention mechanism to learn representations of network traffic that incorporate the spatial and temporal features of the data. The authors evaluate their approach on the latest flow-based dataset for near-earth remote sensing systems. Their results show it outperforms state-of-the-art GNN-based NIDS approaches by a significant margin. The author's work substantially contributes to the NIDS. Their approach can improve computer network security by making NIDS more effective against novel and zero-day attacks, especially in near-earth remote sensing systems.

Qi et al. [72] proposed a novel Graph Ensemble Network (GENet) for traffic prediction. GENet is a deep learning model that learns to predict traffic flow by leveraging the power of GNNs. A graph-based representation of the traffic network captures the relationships between road segments and intersections. A multi-scale GNN architecture allows GENet to learn representations of traffic at different levels of granularity. In this work, a customized loss function is designed to improve the performance of GENet on traffic prediction tasks.

The above papers introduce ensemble-based GNN use cases for some real-world applications. In the following sectrion, we will address ensemble-based GNN techniques for the detection of network attacks.

2.6.7 Ensemble-based GNN Techniques for Attacks Detection

Control Area Network (CAN)s are widely used in vehicles and other embedded systems to communicate between components. However, CANs are vulnerable to various attacks, including message injection, suspension, and falsification. These attacks can have serious consequences, such as disrupting vehicle operations or causing accidents. Traditional anomaly detection mechanisms for CANs are either slow or ineffective against some attacks. Additionally, these mechanisms often require a lot of training data, which can take some effort to collect. Zhu et al. [73] proposed a Federated Graph Neural Network (FGNN) for fast anomaly detection in CANs. This novel approach leverages the power of GNNs to learn representations of CAN traffic that can be used to detect anomalies. This technique is federated, meaning it can be trained on a distributed network without compromising privacy. Their approach works by constructing a graph from the CAN traffic data. The nodes of the graph represent CAN messages, and the edges of the graph represent the relationships between different notes. They trained their model to learn representations of the nodes and, in turn, to detect anomalies in the CAN traffic. The authors compared their model to several state-of-the-art CAN anomaly detection methods, including Support Vector Machine (SVMs), random forests, and other GNN-based methods. The results showed that their approach outperformed all baseline methods regarding accuracy and speed. Specifically, they achieved an accuracy of 99.9% on the UNSW-NB15 dataset and an accuracy of 99.8% on the Kvaser dataset. The F1-score of their model is 99.1% with binary classification and but due to the unbalanced dataset, they achieved a weighted F1-score is 90.7% with multi-classification of attacks. Additionally, FGNN detected anomalies in CAN traffic in as little as three milliseconds. However, more research is needed to evaluate the performance of FGNN on large-scale real-world CAN systems and to investigate how it can be used to detect other types of anomalies.

Esmaeili et al. [74] reported a novel GNN-based adversarial malware detection framework for IoT devices. The proposed framework is designed to be more effective

against IoT malware than traditional malware detection methods, as it leverages the power of GNNs to learn representations of IoT devices and their interactions. The proposed framework works by constructing a graph where the nodes represent IoT devices and the edges represent their interactions. The proposed framework then uses a GNN to learn representations of the nodes and edges of the graph and capture the complex relationships between the devices and their interactions. These representations are then used to detect malware on IoT devices. The results showed that the proposed framework could detect malware with an accuracy of 99.18% but they have not reported weighted F1-scores. Their detector achieved a 98.96% detection rate, 1.17% higher than the previous best method. Additionally, their detector had a 5.95% lower False Positive Rate (FPR) than the previous best method. Moreover, their detector achieved an F1 score as good as of 0.9658. The proposed framework also has a fast response time of 3 milliseconds. However, this framework has not been evaluated on a large-scale, real-world IoT system, nor how it can detect other types of anomalies in IoT data, such as malicious node network traffic behavior over the network, and how these malicious nodes are used to conduct DDoS attacks.

2.6.8 SmartNIC for Secure Authentication

The rise of swarm intelligence-based autonomous systems in next-generation networks has brought significant advancements to transportation. These systems effectively handle various tasks, from scheduling to real-time navigation and tracking. However, the heavy computational and communication load can limit their full potential [75]. To address these challenges, edge-centric systems have emerged as a viable solution. These systems mitigate load management issues, communication delays, and traffic overhead by deploying edge-centric servers and edge micro data centers as private cloud servers closer to swarm devices by using a DPU at both ends (e.g., Bluefield SmartNiC)[76]. Local data processing at the DPU improves latency and optimizes bandwidth utilization. However, the proximity of private cloud servers introduces new security concerns for swarm intelligence-based autonomous

systems [77], [78]. Adversaries can exploit wireless mediums to intercept, eavesdrop, tamper with, delete, or replay information, jeopardizing device security and privacy [79]. An active attacker who gains access to an open network and location can easily extract details about the device's location, messages exchanged, and time spent at the location [80]. Consequently, designing a secure and efficient authentication protocol becomes crucial to protect devices in these systems [81]. Edge-centric computing encompasses proximity, intelligence, trust, and management control by humans or machines on the edge [82]. Edge computing extends cloud-based services to the network's edge and enhances various services in edge-centric Autonomous Swarm Devices (ASD)s systems. From this viewpoint, the incorporation of privacy-preserving reputation updating (PPRU) into edge computing serves as an illustration and improves the overall effectiveness of the network[83]. Several authors have provided a comprehensive overview of security challenges, security attacks, security concerns, and potential solutions in edge computing systems [84]–[93].

Physical Unclonable Function (PUF)s have been considered essential for ensuring physical security in authentication protocols. However, recent developments, such as the Xilinx-FPGA-based PUF proposed by [94], have aimed to address this need. Despite this effort, vulnerabilities in hardware-based PUFs have been identified, as highlighted by [95]. We summarized the limitations of existing state-of-the-art authentication protocols, as shown in Table 2.1. These protocols utilize various cryptographic techniques such as identity-based schemes, ECC, lattice-based encryption, and PUF in the field of edge computing. The comparison reveals the trade-offs between computational costs, resilience to physical attacks, and the security features offered by each protocol.

Due to the capacity to handle large and distributed tasks, the swarm devices are on their way to becoming part and parcel of next-generation networks. However, owing to low latency and scalability issues, the cloud-based frameworks are not suitable for swarm devices. Edge computing can resolve these issues, but edge computing may also introduce new security threats, including vulnerability against physical attacks, weakness against forgery attacks, and tampering of the hardware.

Table 2.1: State-of-the-art Authentication Protocols Comparison

Protocol	Year	Methods	Limitations	Strengths
[88]	2020	Identity-based AKE scheme, one-way hash function	Lacks resilience to physical attacks	Ensures mutual authentication, user anonymity
[96]	2022	ECC, one-way hash	High computational cost	Resilient to physical attacks
[97]	2022	Lattice-based encryption and signature algorithms	High computational cost	Provides key agreement and location privacy
[94]	2023	PUF, XOR, hash	Vulnerable to impersonation and key-leakage attacks	Mutual authentication, FPGA-based PUF
[98]	2023	One-way hash, ECC	Lacks resilience to physical attacks at device level	Provides physical security and privacy
[99]	2023	PUF, ECC	High computation cost, lacks anonymity	Key agreement and privacy with physical security
[100]	2023	PUF, fuzzy extractor	Vulnerable to modeling attacks	Mutual authentication, silicon-based PUF

Recently, some authentication protocols based on PUFs for edge-computing-based swarm devices have been proposed. However, autonomous Intelligent Swarm System (ISS) require secure and efficient authentication schemes to ensure the safety and reliability of their operations. Existing authentication schemes for edge computing systems are often insufficient in terms of security, efficiency, and attack resistance. For example, they may need to verify the physical identity of swarm devices in different environments, such as smart cities, factories, and autonomous vehicle networks. This emphasizes the ongoing challenge of achieving robust physical security in authentication systems.

Various cryptographic authentication techniques have been designed for edge computing systems. However, these schemes indicate various limitations, such as a lack of security features and computational and communication overhead issues.

For example, Irshad et al. [101] proposed a key negotiation scheme that lacks a user revocation mechanism reported by [102].

Xiong et al. [102] proposed an enhanced key establishment scheme, which is vulnerable to impersonation and anonymity violation attacks. Jia et al. [85] designed a bilinear-based protocol susceptible to security attacks.

Agilandeeswari et al. [80] proposed a scheme that uses XOR operations for key agreement and authentication in the communication between vehicles and smart grids in a social IoT environment. They also presented formal and informal analyses to prove their scheme is secure from some well-known attacks.

Yang et al. [103] proposed an inefficient key agreement scheme for resource-

constrained swarm devices based on Elliptic Curve Cryptography (ECC). The protocol in question involves assigning a set of pseudo-IDs and a corresponding family of secret keys to each user. This is accomplished through an Access Service Network Gateway (ASN-GW), which executes a key pre-distribution process based on elliptic curve cryptography. Unlike other schemes, no bilinear pairing is required. However, the ASN-GW must generate many pseudo-IDs for each registered user, and any mobile user must store many pseudo-IDs and their corresponding secret keys. As a result, it is not practical for mobile devices with limited storage capacity.

Irshad et al. [101] proposed a protocol with a key negotiation for edge computing systems designed to be secure and efficient. The protocol uses a combination of symmetric and public key cryptography to negotiate a shared key between two devices. The protocol also includes a user revocation mechanism, allowing devices to revoke their key if it is compromised. However, the protocol lacks a user revocation mechanism. If a device's key is compromised, the attacker could still use it to impersonate the user and access their data.

Xiong et al. [102] reported that Irshad et al. protocol [101] is not secure. Specifically, they claim the protocol cannot accommodate a secure user revocation. In response to this challenge, they introduced an improved key establishment scheme for swarm-based edge computing systems. They claimed that their enhanced scheme offers efficiency and resilience against potential attacks.

Jia et al. [85] presented a protocol that utilizes bilinear pairing operations. Their claims regarding the scheme emphasized its robustness and efficiency, with purported attributes including perfect forward secrecy and user privacy. Le et al. [104] observed that Jia et al. protocols do not deliver the security features they initially claimed, particularly regarding resistance against impersonation and preserving user anonymity. They also suffer from ephemeral key disclosure attacks.

Li et al. [105] introduced a key agreement technique based on ECC for mobile devices in edge computing systems. The scheme's primary objectives include the provision of user anonymity and perfect forward secrecy.

However, vulnerabilities have been identified within the protocol by Jia et al. [85],

notably concerning susceptibility to impersonation and privacy breaches.

In another study, Castello et al. [106] proposed a novel authentication protocol for swarm-based intelligent systems using blockchain technology. While the paper highlights the potential benefits of blockchain in achieving secure and consensus-driven operations, it falls short in providing a critical evaluation of the limitations and challenges. The scalability issues of integrating blockchain into swarm robotics are not discussed. It should also address the computational requirements and potential performance trade-offs associated with blockchain implementation. Moreover, the paper did not explore the practical feasibility and cost-effectiveness of implementing blockchain in real-world swarm robotics applications.

In another study, Ogundoyin et al. [107] presented a certificate authentication scheme for edge computing systems. The scheme aims to provide secure communication while ensuring user anonymity and resistance against attacks. The authors claimed their proposed scheme is more efficient than other certificate authentication schemes. The scheme they proposed for secret key generation suffers from a flaw where the master secret key of the private key generator (PKG) can be leaked through their key generation method.

Nandy et al. [108] proposed a secure authentication protocol for vehicular ad hoc networks (VANETs) in edge computing systems. The protocol is based on the combination of symmetric and asymmetric key cryptography to provide mutual authentication, confidentiality, and integrity. The authors have compared and analyzed the performance of the proposed authentication scheme with the existing authentication protocols.

In another study, Lin et al. [109] proposed a lightweight authentication protocol for autonomous vehicles based on blockchain technology. The protocol provides secure authentication and efficient data exchange between autonomous vehicles and their surrounding environment. They also provide a detailed analysis of schemes based on blockchain-based authentication solutions.

In a different approach, Miao et al., [110] proposed an enhanced authentication protocol for autonomous vehicles based on ECC. They claim that protocol provides

secure and efficient authentication and key agreement between vehicles and their surrounding environment.

According to a comprehensive study by Kumar et al., [111], there are several types of security threats that these systems are vulnerable to, including replay attacks, session key security attacks, physical attacks, man-in-the-middle attacks, eavesdropping attacks, lack of forward and backward secrecy, impersonation attacks, non-synchronous attacks, and lack of anonymity.

Alladi et al. [112] proposed an authentication scheme that focuses on achieving lightweight and efficient authentication for resource-constrained vehicles. This is critical in vehicular networks where devices may have limited processing power and energy constraints. The scheme aims to minimize computational overhead and communication costs while emphasizing lightweight design, while maintaining high security.

Chaudhry et al. also proposed two solutions, including a bilinear pairing-based authentication protocol for devices in distributed IoT environments [113] and device-to-device access control using Elliptic Curve Cryptography (ECC). Due to the high computational overhead of the pairing-based solution and due to the storage of certificates on SmartNIC memory, both protocols were deemed impractical for securing the communication among Swarm devices.

Liu et al. [114] have introduced a new method called Privacy-Preserving Reputation Updating (PPRU) for cloud-assisted vehicular networks. This method ensures privacy by using the ECC and Paillier algorithms. However, the Cloud Service Provider (CSP), which is honest-but-curious (HBC), can still collect and pre-process the reputation feedback in this privacy-preserving system. Which can help significantly minimize overheads on the TA side.

Z. Liu et al. [115] proposed a Lightweight Trustworthy Message Exchange (LTME) scheme that offers rich functionality but faces computation limitations due to resource constraints on UAVs. To overcome computational issues, they also presented a simple scheme called the sLTME scheme that provides similar functionality to LTME but with lower communication overhead, potentially reducing the impact

on UAV network performance. The sLTME had lower TGEN values than LTME, indicating better performance, but lacked confidentiality when exchanging messages.

Liu et al. [116] proposed a Privacy-Preserving Trust Management (PPTM) scheme to address the challenges associated with distributing emergency messages in Space-Air-Ground-Integrated Vehicular Networks (SAGIVN) situations. This system is reliable in this use case, widely applicable, and has many positive qualities. The PPTM scheme enables robust conditional privacy preservation and accurate trust management with minimal communication overhead. To confirm the effectiveness of the PPTM system, the authors conducted extensive theoretical analysis and simulated tests.

In a recent study, Abdel et al. [117] proposed an efficient authentication protocol for the swarm-based intelligent system for Unmanned Ariel Vehicle (UAV). The paper presents a lightweight proxy signature-based authentication mechanism for 5G Drone to Drone (D2D) communication in drone swarms.

However, it lacks a detailed analysis of security vulnerabilities and does not compare its performance with existing mechanisms. The potential computational overhead and implications of leader transitions are not adequately addressed.

Further research is needed to evaluate the security robustness, conduct comparative assessments, and consider leader transition's computational impact and dynamics within the swarm. The proposed mechanism is implemented in an NS-3 simulation environment and tested on a Raspberry Pi 3 device; a more thorough evaluation against other authentication schemes would provide a better understanding of its performance, scalability, and reliability in realistic scenarios.

The literature review shows that various authentication protocols and schemes have been proposed for edge computing systems and ISS in recent years. These protocols and schemes utilize cryptographic methods and tools, such as biometric authentication, blockchain, and hardware-based authentication, to ensure secure communication between devices in the swarm system.

However, these existing techniques need to improve both security and efficiency. Hence, developing lightweight, secure key agreements and efficient techniques tai-

lored for edge computing systems is paramount. However, additional research is imperative to advance the development of lightweight, efficient, and secure authentication protocols that can effectively cater to the distinctive challenges and requisites of autonomous ISS.

The reviewed literature highlights that existing authentication protocols for edge and swarm systems often face trade-offs between security guarantees, computational cost, and resistance to physical attacks. While hardware-assisted techniques such as PUFs offer promising properties, their practical integration into programmable network devices remains an open research challenge.

2.7 SmartNIC for Cloud and HPC Workload

Remote Direct Memory Access (RDMA) has become a cornerstone in the design of HPC, Artificial Intelligence (AI), and modern storage architectures due to its ability to reduce latency and offload CPU overhead. SmartNICs and Data Processing Unit (DPU) such as Bluefield-2 (BF2) (200G) and Bluefield-3 (BF3) (200G/400G) [118] have emerged to accelerate data-intensive tasks. DPUs are capable of accelerating AI training [119], faster data transfer [120], enabling faster molecular dynamics simulations (MiniMD) [121], and offloading NVMe-over-Fabrics for cloud storage function [122], and having computation closer to the network [123]. DPUs have been used to offload MPI collective operations [124]–[126], delivering reduced latency and improved concurrency in distributed scientific applications [127].

With the recent release of BF3, substantial architectural advancements, such as dual-channel Double Data Rate 5 (DDR5)-5600 memory, increased Advanced RISC Machine (ARM) core count, and Peripheral Component Interconnect Express (PCIe) Gen5 support, present a compelling opportunity to reassess the RDMA capabilities previously established by BF2. While BF3 promises higher memory bandwidth, lower latency, and improved concurrency, the real-world impact of these enhancements remains underexplored in practical RDMA deployments, particularly in both DPU (ARM-based) and host Intel x86 Architecture (x86)-based Network Interface Card (NIC) modes. The BF3 SmartNIC introduces a major architectural leap over

its predecessor BF2 [119].

BF2's Arm CortexA72 cores and DDR4 subsystem often become bottlenecks under kernel-level packet processing, limiting sustainable link utilization to around half of the expected rate [128].

BF3 integrates up to 16 Arm CortexA78 cores and a many-core RISCV datapath accelerator with DDR5 memory, delivering vastly improved compute, network, and memory throughput [129]. Kashyap et al. evaluate that BF3 's ARM architecture achieves an IPC nearly twice that of BF2 and, combined with off-path datapath acceleration, delivers up to 3.9 X higher host bandwidth in comparable RDMA use cases [130].

2.7.1 NVMe Offload

NVIDIA's BF3 offloads NVMe storage processing by acting as an NVMe target to the host and an NVMe initiator to the actual storage device. Figure 2.2 presents a BF3 NVMe offload architecture. In this architecture, a PCIe NVMe controller interfaces with the host server internally, intercepts and processes NVMe commands using its on-board ARM cores and hardware accelerators.

The DOCA SDK's NVMe Emulation library (built on SPDK) is used to emulate a complete NVMe controller on BlueField. At the same time, BlueField's Data Path Accelerator (DPA) hardware handles low-level tasks like doorbell monitoring, DMA data movement, and interrupt signaling. NVMe Offload design enables the host to view what appears to be a local NVMe drive, while all the heavy lifting of NVMe command execution and data transfer is offloaded to the SmartNIC.

BF3 even includes dedicated NVMe-oF/NVMe-TCP offload engines for accelerated storage networking, meaning it's highly optimized for NVMe tasks in both direct-attached and networked storage scenarios.

2.7.2 DOCA RDMA Offload in BF2 and BF3

The BF2 and BF3 DPU s' architecture is depicted in Figure 2.3 as separate blocks to compare their internal components. The Host System interacts with the DPU

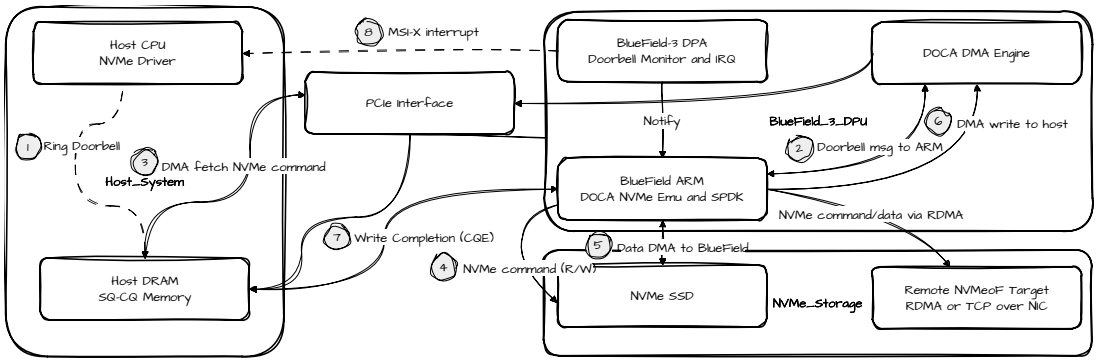


Figure 2.2: NVMe Offload Architecture on BF3 with RDMA target path. Solid arrows represent data flow, and dashed arrows indicate the control path.

(BF2/BF3) via PCIe (shown by solid black arrows). The Remote Endpoint receives data over the network. Both DPUs handle RDMA reads/writes similarly (as indicated by the parallel arrows for RDMA read/write paths), but BF3 has the extra DPA stage where inline processing can occur between the ARM and NIC. In both cases, the one-sided RDMA Write path goes from Host Memory to Remote Memory without involving the remote CPU.

In contrast, a Send/Recv involves the remote CPU posting a receive buffer (the diagram shows a send going to the Remote NIC, which then places data into Remote Memory and notifies the Remote CPU). The RDMA Read path is illustrated as originating from Remote Memory and returning to Host Memory. This side-by-side depiction highlights that BF3 retains all the capabilities of BF2 while offering higher performance and the DPA inline accelerator.

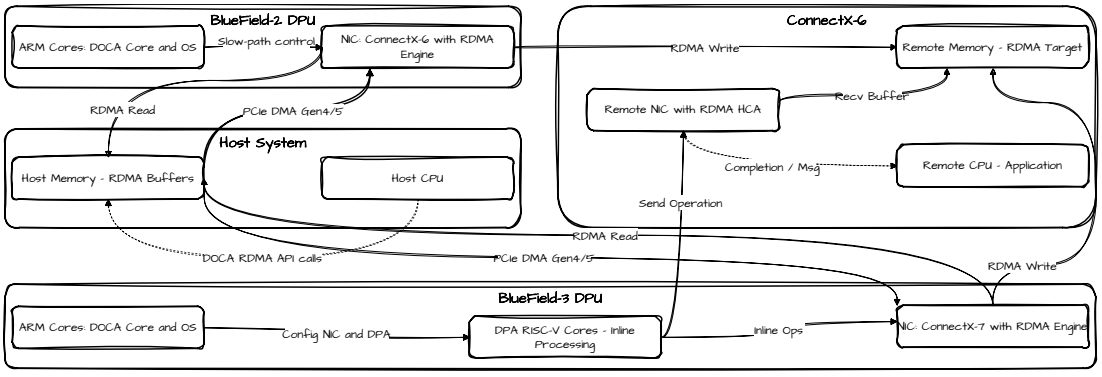


Figure 2.3: RDMA Offload Architecture on BF2 and BF3 with RDMA target path with host ConnectX-6

2.7.3 DOCA GPU Packet Processing Offload

Figure 2.4 presents a state-of-the-art offloading architecture in which GPU threads directly interface with RDMA queues and initiate memory operations via BlueField-based ConnectX-7 NIC, enabling a tightly coupled and efficient data pipeline. In this architecture, CUDA threads within GPU kernel blocks execute networking I/O functions `gpu_netio_post_write()`, `gpu_netio_commit()`, and `gpu_netio_flush()` directly on GPU-resident packet buffers.

RDMA operations start using ConnectX-7's hardware RDMA engine, and PCIe communicates these operations to the DPU. This GPU-to-DPU pipeline reduces reliance on host CPU intermediates, dramatically lowering latency and increasing throughput by enabling zero-copy and one-sided RDMA communication. The use of direct PCIe-connected GPU buffers and CUDA-aware NIC operations allows for low-overhead, fine-grained data transfer. The RDMA queue operates asynchronously, allowing multiple write operations to be posted and flushed in a pipelined fashion, further enhancing parallelism.

The BF3 acts not just as a data mover but also as a programmable accelerator. This enables offloading protocol processing, memory management, and transport operations from the host CPU, paving the way for software-defined, hardware-accelerated data centers optimized for AI, HPC, and edge workloads.

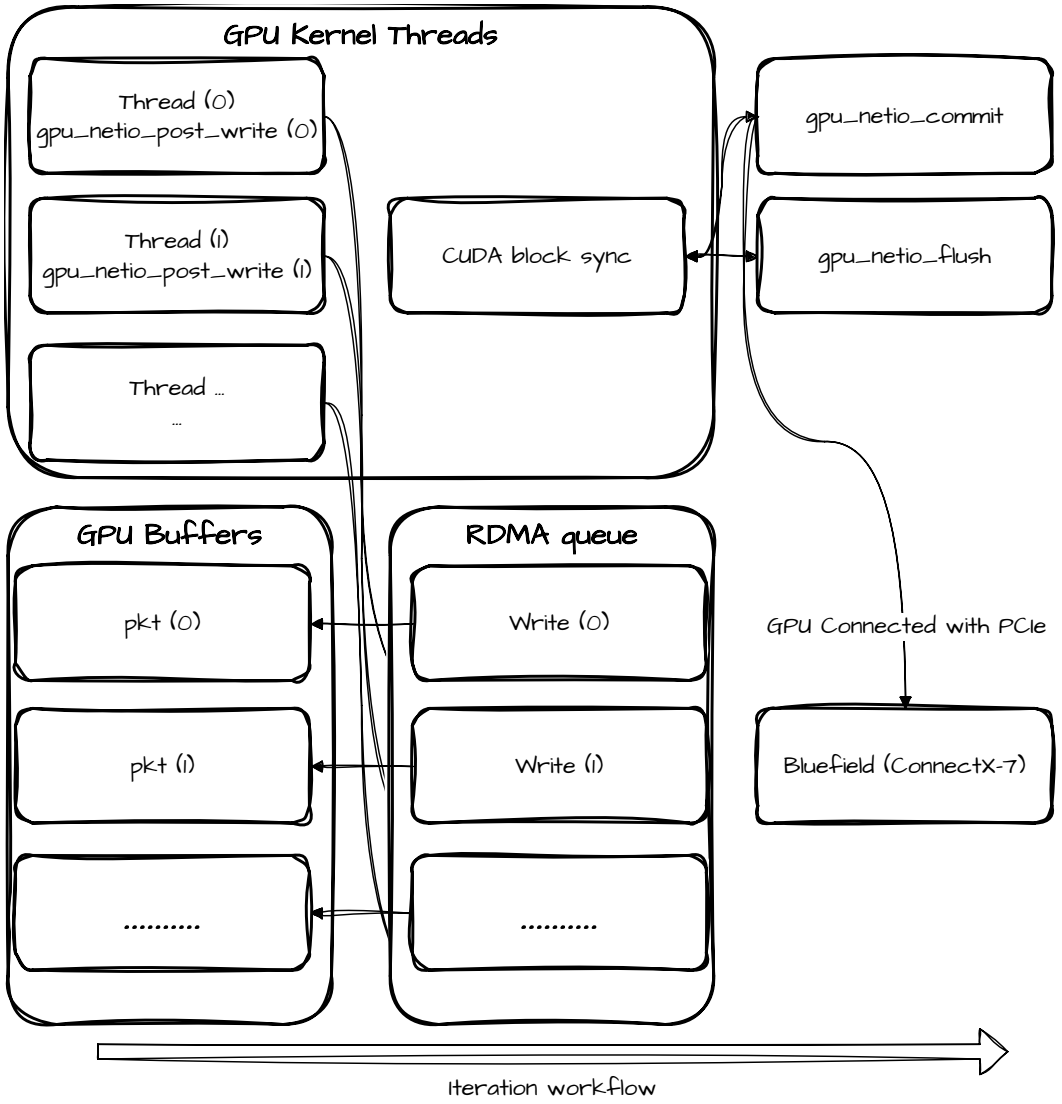


Figure 2.4: DOCA GPU Packet Processing Offload on BF2 and BF3

The growing demand for low-latency, high-bandwidth communication in modern data centers has spurred extensive research into Remote Direct Memory Access (RDMA) and SmartNIC architectures. This section highlights relevant studies that analyze the performance of RDMA, SmartNICs, and programmable DPUs, particularly NVIDIAs BlueField series.

Liu et al. [128] conducted one of the earliest evaluations of the NVIDIA BF2 platform, identifying constraints due to its limited CPU core count (Arm Cortex-A72) and DDR4 memory. Their work highlighted that BF2 struggles to sustain high-throughput RDMA workloads in kernel space due to CPU-bound bottlenecks and insufficient memory bandwidth.

Michalowicz et al. introduced *DPU-Bench*, a dedicated microbenchmark suite designed to quantify offload efficiency in SmartNICs [131]. Their framework enables detailed measurement of memory access, queue processing, and CPU utilization, providing critical insights into the effectiveness of RDMA offloading and queue handling strategies on SmartNICs.

In a follow-up study, the same authors presented a direct comparison of BF2 and BF3 performance under RDMA workloads [132]. Their results emphasized BF3s superior throughput and lower latency, attributing this to architectural changes such as the upgraded ARM cores, DDR5 memory, and PCIe Gen5 support. This complements our findings and further validates the architectural advantages of BF3.

Wei et al. [133] analyzed the behavior of off-path SmartNICs in distributed systems, highlighting the benefits of separating the datapath from control-plane processing. Their characterization of off-path acceleration aligns with BF3's use of a RISC-V-based datapath accelerator to improve isolation and reduce jitter under high message rates.

Tajbakhsh et al. [134] explored end-to-end network performance benchmarking techniques across heterogeneous SmartNICs. Their work underscores the importance of portable microbenchmarks and systematic testing methodology, which inspires our experimental design using both Perftest and DOCA RDMA tools.

Kashyap et al. [130] contributed a detailed microarchitectural model for BF3, reporting a 3.9Œ improvement in RDMA throughput compared to BF2. They attribute this to higher IPC, increased ARM core count, and BF3s datapath accelerator.

WekaIO [129] also highlights commercial claims of BF3s capabilities, including up to 16 ARM Cortex-A78 cores, DDR5-5600 memory, and PCIe Gen5, all of which align with observed performance boosts in host and DPU mode.

Usman et al. [125] enable OpenMP to support SmarNIC offloading. Their solution, ODOS-MPI, is based on DOCA [124] and it is evaluated against an extension of the OSU Microbenchmark with support for SmartNIC offloading. Results unveil that MPI kernels offloading increases the overlap and, in turn, reduces the latency

thanks to their proximity to the network.

These studies highlight the need for systematic benchmarking of RDMA performance across SmartNIC generations under realistic configuration and tuning conditions.

2.8 User Plane Function Offload

Mobile networks have significantly enhanced communication capabilities, enabling higher data rates, lower latency, and support for billions of connected devices. These developments increase the demand for highly efficient next-generation mobile networks, particularly for video streaming, online gaming, and the Internet of Things (IoT). The development of smarter cities and enhanced industrial automation is also driving the need for these networks. However, emerging use cases, such as autonomous driving, medical robotics, and real-time operational industrial control, demand ultra-reliable, low-latency, and deterministic communication, which are beyond the capabilities of current 5G infrastructures [135], [136].

To meet these evolving demands, 5G architectures have adopted technologies such as Network Function Virtualization (NFV) and Software-Defined Networking (SDN) [137]–[140]. NFV enables the virtualization of core network functions, including the User Plane Function (UPF), on general-purpose hardware, allowing for dynamic deployment and scalability [141]–[145]. SDN further decouples the control and data planes, allowing centralized traffic orchestration and flexible resource management [146].

Despite these advances, virtualized UPF implementations often face performance bottlenecks when deployed on commodity servers or traditional NICs. Kernel-based packet processing, high context-switch overheads, and inefficient handling of GTP tunnels limit their ability to meet the high-throughput and low-latency demands of modern mobile networks [147]. Current approaches [148]–[154] have introduced in-network acceleration using Data Plane Development Kit (DPDK) and P4 to improve performance, yet they fall short in supporting scalable, flow-aware UPFs with hardware offload. DPDK is a widely used software framework that accelerates packet pro-

cessing on CPUs by bypassing the kernel networking stack. While it reduces kernel overhead, DPDK remains CPU-bound, leading to scalability limitations under high traffic loads. High-performance user-space dataplanes, such as DPDK, eliminate kernel overheads while keeping per-packet work on general-purpose CPUs, which limits scalability at high packet-per-second (PPS) rates. In-kernel eBPF/XDP reduces I/O overhead and enables safe packet processing within the kernel's fast path, it still consumes host CPU cycles and is constrained by verifier/program complexity. In contrast, 6GUPF offloads encapsulation/decapsulation and flow lookup into the DPU hardware pipes, freeing the host CPU even under high PPS.

Recent work, such as HiP4-UPF [153], has shown performance improvements in UPF deployment using P4 programmable switches. By introducing a three-tiered optimization strategy, RED, SPT, and CAD, they addressed rule storage inefficiencies and operational latency. Their approach achieved a 2X increase in UE support and 84.6% lower reporting latency compared to baseline systems. However, HiP4-UPF remains tightly coupled to Intel's Tofino architecture and lacks full on-board buffering capabilities, which requires external servers for packet storage. P4 targets line-rate parsing/match-action in fixed pipelines. P4 switch-based UPFs achieve low latency but are limited by on-chip memory (TCAM/SRAM) and pipeline depth for stateful processing. Our approach differs by maintaining a fully programmable software control plane while offloading data-plane primitives (GTP-U encapsulation/decapsulation, flow steering) to the DPU. This avoids the table-size constraints and supports larger flow tracking states (e.g., TEID/5-tuple) without sacrificing the line-rate throughput of 400 Gbit/s.

In particular, existing UPF solutions struggle with limitations in the number of flow tracking, CPU-bound GTP processing, and a lack of granular QoS enforcement. These challenges are crucial for a transition to 6G, where networks must support billions of devices and various real-time services and applications.

These limitations motivate the exploration of programmable data-plane offload using DPUs, enabling flow-aware processing and stateful tracking closer to the network interface.

2.9 Chapter Summary

This chapter reviewed the state of the art in programmable networking, security, and high-performance communication to contextualize the research questions introduced in Chapter 1. The discussion began with the evolution of network interface technologies, highlighting the transition from conventional NICs to SmartNICs and DPUs, and emphasizing how programmability and hardware offload enable new execution models beyond CPU-centric designs.

The survey of network security literature showed that software-based IDS and DDoS mitigation systems struggle to sustain performance at high data rates, particularly when stateful inspection and deep packet analysis are required. While learning-based approaches improve detection accuracy, their deployment is often constrained by latency, scalability, and resource overhead. Graph-based methods address some of these limitations by capturing structural traffic relationships, yet introduce additional computational challenges that complicate real-time enforcement.

The review further examined hardware-assisted security mechanisms, including SmartNIC-based intrusion prevention and authentication schemes for edge and swarm systems. Existing solutions improve isolation and latency but frequently rely on specialized hardware or narrow assumptions that limit deployability and extensibility.

Finally, the chapter analyzed SmartNIC and DPU offloading across cloud, HPC, and mobile-core dataplanes, including NVMe and RDMA acceleration as well as UPF-specific constraints in GTP-U processing and flow-aware steering. Prior evaluations confirm SmartNICs' potential to reduce host overhead and improve performance, but also reveal gaps in systematic comparisons across generations and execution modes.

Overall, the literature highlights a fragmentation between detection, enforcement, and performance evaluation. This gap motivates the need for a unified, cross-layer investigation of DPUs as a common substrate for security, networking, and high-performance workloads. The following chapters build on this foundation by

addressing these challenges through empirical evaluation and system-level design.

Chapter 3

Graph Neural Networks for DDoS Detection

DDoS attacks are a major threat to computer networks. These attacks can be carried out by flooding a network with malicious traffic, overwhelming its resources, and/or making it unavailable to legitimate users. Existing machine learning methods for DDoS attack detection typically use statistical features of network traffic, such as packet sizes and inter-arrival times. However, these methods often fail to capture the complex relationships between different traffic flows. This thesis proposes a new DDoS attack detection approach that uses GNN ensemble learning. GNN ensemble learning is a type of machine learning that combines multiple GNN models to improve the detection accuracy. We evaluated our approach on the Canadian Institute for Cybersecurity Intrusion Detection Evaluation Dataset (CICIDS2018) and CICIDS2017 datasets, a benchmark dataset for DDoS attack detection. Our work provides two main contributions. First, we extend our DDoS attack detection approach using GNN ensemble learning. Second, we evaluate and fine-tune hyperparameter metrics using ensemble learning, significantly improving accuracy relative to a single GNN model and achieving an average 3.2% higher F1-score. Additionally, our approach reduces overfitting by incorporating regularization techniques, including dropout and early stopping. Specifically, we use a hierarchical ensemble of GNN, where each GNN learns the relationships between traffic flows at a different granularity level. We then use bagging and boosting to combine the predictions of

the individual GNN, further improving detection accuracy. Results are presented Section 3.3, where our system achieves 99.67% accuracy and an F1-score of 99.29%, outperforming state-of-the-art methods even with a single traffic architecture.

In our recent preliminary work [155], we introduced a hierarchical graph structure, Flow-to-Traffic Graph (FTG), that combines aggregated traffic-level and flow-level structures into a two-level representation. Additionally, we proposed the FTG-Net model, a GNN designed to process FTG graphs and effectively classify flows as legitimate or malicious. This approach adeptly captures and embeds the intricate flow structures between hosts and servers, combining this representation with the overall traffic structure. By doing so, it accommodates various DDoS attacks within the recognized patterns. Significantly, this solution relies solely on traffic topology, eliminating the need for computationally expensive stateful features in real-time scenarios. This divergence avoids potential overfitting to specific training dataset characteristics, as is commonly encountered in models that incorporate all stateful features [156]. When deploying FTG-Net in real-world scenarios, it is essential to integrate specific stateful features into the graph node. These features should be contingent on the network's requirements and functionalities, as they can significantly enrich the representation and ultimately improve performance. While our approach demonstrated impressive accuracy, we recognized that the battle against DDoS attacks is an ever-evolving challenge. These attacks continuously adapt, mimicking legitimate traffic patterns and evading detection mechanisms.

We extended our lab's previous work [155] by introducing a novel way to detect DDoS attacks using ensemble learning with GNN. Ensemble learning is a method that combines multiple machine learning models to improve the system's overall performance [157]. In our case, we use a combination of GNNs to detect DDoS attacks. By learning from network traffic structure, GNNs can identify patterns indicative of DDoS attacks. This happens because each GNN model learns different patterns from the data, and by combining the predictions of the other models, we can obtain a more accurate overall prediction.

Our approach employs a sub-sampling strategy to reduce the computational com-

plexity of training GNN models on large graphs. This involves dividing the graph into smaller subgraphs and training separate GNN models on each subgraph. This reduces the number of nodes and edges each GNN model needs to process, significantly improving training efficiency. Instead of using computationally expensive models, we use lightweight graph networks with simpler architectures and fewer parameters. This further reduces the computational requirements of training and inference. For instance, we employed GCN with fewer convolutional layers. The GCN can help build a model from network topologies represented as graphs. They enable the extraction of spatial features by learning from the relationships between nodes and their neighbors. They can extract temporal features, which allow them to capture changes in the network over time [158]. This lightweight GNN architecture aggregates information from neighboring nodes more efficiently. Ensemble learning combines predictions from multiple models to improve overall accuracy. Our approach trains multiple GNN models on different subgraphs and aggregates their predictions to produce the final output. This approach enhances accuracy and reduces the computational burden compared to training a single large GNN model on the entire graph.

3.1 FTG-Net-E Graph Structures and GNN Models

Network traffic resembles almost naturally message passing between nodes in an undirected graph. It can be modelled as a hierarchy of flow entries, both with fine-grained (pachets) and coarse-grained (endpoint communications). Our proposed basic structure, FTG, is constructed from traffic data divided into time slots of size t_s. The structure is composed of a high-level Traffic graph related to each time slot, with each node corresponding to a low-level Flow Graph. Every level has a GNN model that is processed in FTG-Net-E. The following subsections describe how the graphs are constructed, how the traffic is converted, and how the architecture of the traffic and flow GNNs.

3.1.1 Flow Graph

Flow Graphs draw inspiration from the graph structure introduced in GraphDDoS [56], but with a modification: the N nodes limit is replaced by the incorporation of time slots. These graphs are crafted by grouping all packets exchanged between two endpoints within a given time slot, even if they pertain to distinct networking flows. Within each group, packets are arranged in ascending order of time and transformed into nodes, featuring only the packet length as a characteristic. We chose the packet length as a key feature as it aids detecting malicious patterns in network traffic, such as large packet sizes used for DDoS attacks and small packets for port scanning attacks. Moreover, this feature is easily available from network traffic data, allowing for efficient real-time processing and model interpretability.

Assigning positive directional flow, whether upstream (client to server) or downstream (server to client), distinguishes positive values to upstream and negative values to downstream lengths.

Consecutive packets sent by one endpoint constitute a mini-group, with edges linking adjacent nodes representing packets from the same mini-group. Connections are established between the first packet of a mini-group and the first packets of the preceding and succeeding mini-groups, similar to the linkage between the last packets of mini-groups. Figure 3.1 shows an example of a Flow Graph.

Given this graph representation of a computer network, A GNNs can be used to classify nodes in the following way:

1. The GNN is initialized with random representations for each node in the network.

2. A message-passing function is applied for multiple time slots t_s: each node sends a message to its neighbors in each slot based on its current representation. The messages are then aggregated at each node, and the node's representation is updated using a function that combines its current representation with the aggregated messages from its neighbors.

3. The final representation of each node is used as the input to a classifier, which

predicts the node class.

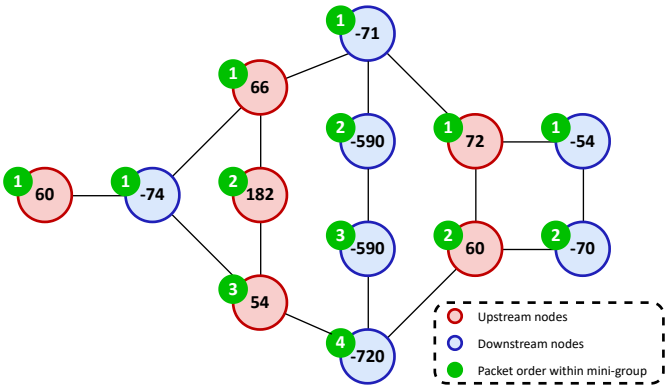


Figure 3.1: Flow Graph example: upstream packets are denoted by positive values, while downstream packets are represented by negative values. The sequence of mini-groups is organized from left to right.

The Flow Graph structure allows us to observe packet relationships within a single flow, which can be different from legitimate traffic in certain DDoS attacks, and relationships between multiple flows between the same endpoints, which can be used to detect burst information and periodic information.

Burst information corresponds to the scenario in which the attacker sends many packets to establish connections with its target on several ports. Periodic information corresponds to low-peace attacks in which the attacker periodically mimics a normal client to occupy the victim's resources as long as possible.

3.1.2 Traffic Graph

A Traffic Graph is a graphical representation of traffic flow over time. This graph is constructed for each time slot, which is typically a discrete interval of time, such as a minute, an hour, or a day. The Traffic Graph shows the amount of data that is transmitted across a network during each time slot, providing valuable insights into network performance, usage patterns, and potential issues that need to be addressed. For each time slot, any pair of endpoints that appears in at least one packet from the traffic data is transformed into a graph node. The features of this node are determined by the output vector of the Flow GNN, which includes a Flow Graph at a relatively low-level. If two nodes share the same source or destination IP address, an edge is added in between. According to the real-world implementation

requirements and capabilities, the set of features describing the communications between two endpoints can be modified and enriched by concatenating the available statistics.

We have developed a Traffic Graph that incorporates the interactions between different flows, displaying the distribution of devices that send requests to servers and flow-specific behaviours that may contain malicious patterns. The encoding of these flow-level patterns is closely linked to the traffic-level topology since the Flow GNN and the Traffic GNN weights are both optimized using the same gradients during the training phase.

3.1.3 Traffic Converting Stage

The traffic data is separated into time slots in this stage. For each time slot, the extraction process produces a Traffic Graph G_t and a set of Flow Graphs $F = G_{f_1}, G_{f_2}, \ldots, G_{f_N}$, where the node indexes of Flow Graphs align with those of the Traffic Graph. The number of Flow Graphs, denoted by N, depends on the time slot. During the supervised learning phase, each time slot data is associated with a corresponding list of labels $Y = y_1, y_2, \ldots, y_N$ consisting of one Flow Graph and one label.

The traffic data is analyzed sequentially in ascending order based on time to extract the graph. An empty Traffic Graph is initialized at the start of each time slot, and an empty sorted list of encountered pairs of endpoints is created. When a new endpoint combination is detected, a new node is added to the Traffic Graph, and a separate Flow Graph is created for the current packet. During this process, the packet direction is preserved as it will be instrumental in categorizing packets into mini-groups. If the combination of endpoints is already present in the list, any new packet that arrives is used to update the corresponding Flow Graph. If the direction of the packet differs from the previously added packet, the mini-group is concluded and interconnected through edges as explained in Section 3.1.1. As the time slot concludes, edges in the Traffic Graph are added following the guidelines outlined in Section 3.1.2.

3.1.4 Flow GNN

The Flow GNN plays a pivotal role in handling flow graphs and constructing embeddings, which will subsequently serve as node features in the traffic graph.

Initially, three GCN Layers are employed on the Flow Graph. These layers disseminate node information throughout of each node with the three-hop neighborhood. Following each GCN Layer, a ReLU activation function is applied.

First, a readout function is utilized to transform the graph into a singular vector. The selected readout function is global mean pooling, which calculates the average of node features across the dimension of a node. The definition of global mean pooling is as follows:

$$h_G = \frac{1}{N} \sum_{n=1}^{N} h_n^{(3)} \tag{3.1}$$

where in Flow Graph, the number of nodes is represented as N and the feature vector of the the n-th node after three iterations of GCN is represented by $h_n^{(3)}$. The output vector is obtained from readout and passed to a fully connected layer, producing the final output.

3.1.5 Traffic GNN

The traffic GNN takes a traffic graph as input and generates a prediction for each node, indicating whether it corresponds to legitimate or malicious traffic. The model consists of three GCN layers followed by a fully connected layer that has a dropout rate of 50%. The dropout is applied to the features of each node individually. The model produces single outputs that are passed through a sigmoid activation function, which provides the final scores ranging from 0 to 1.

3.1.6 Graph Features Engineering

Feature engineering is a critical process that helps extract relevant information from raw data. In the context of network traffic analysis, the construction of traffic graphs is a structured representation of network traffic patterns that enables graph-based

analysis. In our approach, we extracted critical attributes from network traffic data that serve as the basis for constructing the traffic graph. These attributes include: the URG Flag Count, SYN Flag Count, RST Flag Count, PSH Flag Count, Packet Size Average, Flow Packets Per Second, FIN Flag Count, ECE Flag Count, ACK Flag Count, Destination Port, and protocol information (IPv6 Hop-by-Hop 0, TCP 6, UDP 17) obtained from packet headers and traffic statistics. The selection of these features was not only made by relevance to intrusion detection, distinguishing between normal and malicious traffic, and their ability to capture diverse network behaviors; we have also experimented with different combinations of features to evaluate their impact on model performance. We used feature selection and ablation studies to identify the most relevant features and assess their individual contribution. We scaled the features to ensure they had a consistent range and distribution, which helped prevent features with larger magnitudes from dominating the learning process. We also addressed missing values in the dataset by employing imputation techniques to replace them with appropriate values, ensuring the dataset was complete and ready for analysis.

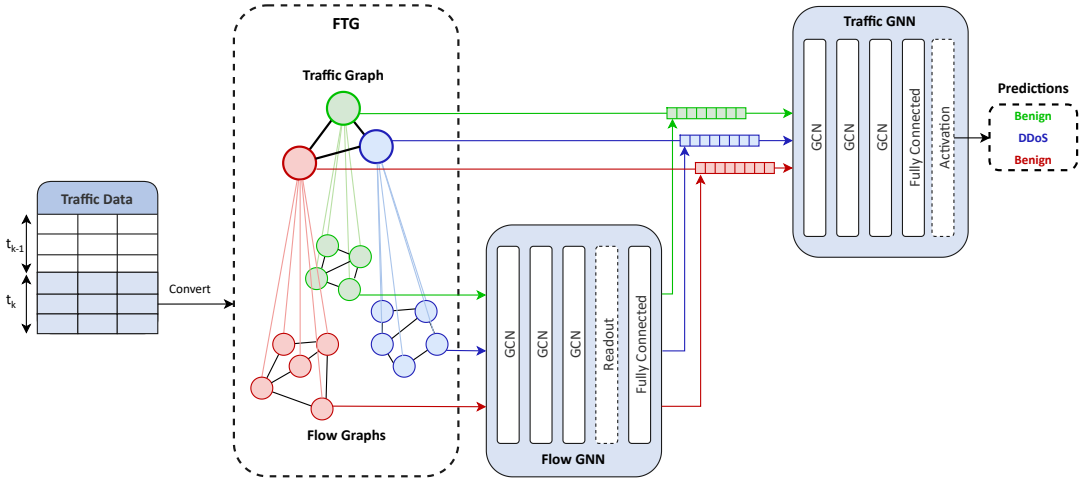


Figure 3.2: FTG-Net-E architecture. (1) "Traffic Data Preprocessing": Training on traffic data converted into flow graphs. (2) "Flow GNN" and "Traffic GNN": Learning representations of individual flow graphs and entire traffic graphs, respectively. (3) "Fully Connected Layer", gives the final predictions based on the Traffic graph network outputs as Benign and DDoS.

3.2 FTG-Net-E Architecture and Dataset

The architecture of FTG-Net-E is based on FTG-Net, which is a hierarchical GNN designed to process the multi-level graph representation FTG. FTG-Net works in three main steps, as shown in Figure 3.2, which are described as follows:

1. **FTG construction**: The first step is to convert the traffic data into FTG structures. This is accomplished by grouping all packets exchanged between two endpoints within a time slot to create a flow graph. The Flow Graphs are then sorted in chronological order and converted to nodes in the Traffic Graph. The features of each node in the Traffic Graph are given by the Flow GNN output vector, which embeds the corresponding Flow Graph.

2. **Flow GNN Inference**: The second step involves utilizing the Flow GNN to analyze the Flow-level Graphs. This module leverages three GCN layers to extract node representations from individual flow graphs. These layers capture local dependencies and interactions within each flow. A readout layer aggregates information from the entire graph, followed by a fully connected layer for final representation learning. Dropout regularization with a rate of 0.5 is applied after each GCN layer to prevent overfitting. The output of the last GCN layer is passed through a fully connected layer with a sigmoid activation function to generate flow-level predictions. The output vector of the Flow GNN for each Flow Graph is used as the features of the corresponding node in the Traffic Graph.

3. **Traffic GNN Inference**: The third step is to process the traffic-level graphs using the Traffic GNN. This module, consisting of three GCN layers, batch normalization, and dropout regularization with a rate of 0.5 after each GCN layer, captures high-level patterns across the network traffic graph. By processing the aggregated flow representations obtained from the Flow GNN, this module gives a global understanding of the overall network traffic behavior. A final fully connected layer with a ReLU activation generates the final predictions, differentiating between benign and DDoS traffic based on the learned

representation. The final prediction is the output of the Traffic GNN for each flow.

FTG-Net-E is designed to exploit the hierarchical structure of FTG to learn more effective representations of the traffic data and to improve the performance of DDoS attack detection. This ensemble model is developed in three main steps: (i) the graph structure is subsampled into subgraphs, and node features into subfeatures, (ii) multiple base GNN models are trained on different subgraphs and subfeatures, such as GCN by [159] and GraphSage by [61]. Each GNN model learns a representation of the nodes in its subgraph, which can then be used to make predictions about the node labels, and (iii) finally the predictions of multiple GNN models are aggregated using a ranking scheme, so to combine the information from different models and make a more accurate prediction.

3.2.1 DDoS Detection Datasets

In 2016, a thesis listed 11 criteria that must be fulfilled for a dataset to be considered suitable for intrusion detection purposes [160]. Among the various publicly available datasets, we chose the Canadian Institute of Cybersecurity Intrusion Detection System Datasets (CIC-IDS2017, CSE-CIC-IDS-2018) as these are modern flow-based datasets that fulfill all these criteria. The CIC-IDS2017 was developed by Sharafaldin et al. [161] and consists of normal and malicious network traffic. The normal traffic is emulated by utilizing B-profiles, which are obtained from the benign (normal) conduct of 25 individuals (human). The malicious traffic consists of common attacks (DoS, DDoS, Brute Force, XSS, SQL Injection, Infiltration, Port Scan, and Botnet), and is generated by executing existing DDoS attacking tools at specific time windows.

Aside to the CIC-IDS-2017, we have trained FTG-Net-E also on its successor, The CSE-CIC-IDS-2018 dataset. The 2018 dataset was generated using the same tools of the 2017 one, but deployed in AWS rather than on a local university network. The CIC-IDS2018 dataset is labeled with more than 83 traffic features collected using CICFlowMeter [162]. The capturing period for this dataset started at 09:00

on Monday and ended at 17:00 on Friday. The DDoS attacks were performed on Friday afternoon and are the only significant for this work. Traffic data have been converted considering only time slots with at least 20% of labeled data. GNN based preprocessed datasets representing without prediction and with prediction labels. From the 2018 dataset, 127,844 additional infiltration samples were obtained, these samples were kept separately to test the robustness of the zero-day capability of our novel proposed approach.

3.2.1.1 Ensemble Learning Model

The ensemble learning model consists of multiple GNN models, each trained with distinct hyperparameters. We created three instances of the GNN model with parameter variations such as learning rate, hidden size, and dropout rate. Despite variations in hyperparameters, the training process remains consistent across all models. After independent training on the same dataset, the predictions from individual GNN models are combined using a soft voting approach. This method involves averaging the predicted probabilities from each model to make final predictions, allowing for more nuanced outcomes compared to simple majority voting.

3.2.1.2 Dataset Preparation

We first loaded the raw dataset to ensure data quality and selected only the relevant attributes for our analysis. After collecting the data, we performed data cleaning procedures that involved eliminating redundant columns, removing rows with missing or infinite values and filtering out any duplicated records. We also handled missing values by forcing the Flow Pkts/s feature to be numeric and dropping rows with missing entries. After preprocessing the dataset, to clean and standardize the features we scaled the data using the StandardScaler implementation from the scikit-learn Python library. The data has been then split into training and test sets using a stratified split to ensure that the distribution of classes in each set is representative of the overall dataset. Using one-hot encoding, we encoded categorical variables like the protocol to facilitate classification tasks. The preprocessed data, which included features such as packet size, duration, and source and destination

IP addresses, was then saved as CSV files for easy integration into our GNN-based DDoS attack detection model.

The resulting dataset, which consisted of 80% benign and 20% malicious samples, was split into training, validation, and test sets. The down-sampled malicious dataset was divided into 70% for training 15% testing, and 15% for validation. The 20 zero-day samples, representing novel malware attacks that were not previously known to the system, were added to the test set along with the benign samples so to assess the ability of our ensemble graph network to detect novel attacks. The validation set was created by randomly selecting 700 samples from each of the five attack categories, namely UDP flood, TCP SYN attack, and HTTP attack, from the malicious training set. As a result, the validation set consisted of a total of 2,100 samples. The benign samples were split into the train, validation, and test sets to maintain the overall distribution of 80% benign and 20% malicious samples in the final datasets.

3.2.1.3 Data for the Anomaly Detector

In the first stage, the anomaly detector was trained using a training set composed of mostly benign traffic. The validation set is instead composed of 80% benign and 20% malicious traffic. This approach was chosen because the anomaly detector primary function is to filter malicious traffic from a stream of predominantly benign traffic. However, it is essential to acknowledge that this method produces a model that is prone to give false positives. One strategy to address this potential issue involves employing a more balanced dataset for training and validation. This entails incorporating a higher proportion of malicious traffic into the validation set, enabling the detector to gain proficiency in distinguishing between benign and malicious traffic. Still, implementing this approach is challenging, as acquiring substantial amounts of malicious traffic data is often tricky. Another approach to enhance the anomaly detector performance involves utilizing a more sophisticated training algorithm. Specific algorithms excel at identifying anomalies and understanding the patterns of normal traffic. Additionally, we fine-tune the parameters of the training

algorithm to improve its effectiveness further. Monitoring the anomaly detector's performance is crucial as new forms of malicious traffic emerge. The detector needs to be re-trained or updated periodically to maintain its efficacy.

3.2.1.4 Classifier Dataset

The GNN classifier learns to classify malicious traffic into attack categories. While its validation set is equally distributed of 50% benign and 50% malicious samples, the attack kinds are balanced within the malicious traffic samples. In order to avoid bias towards a specific type of attack, it's crucial to consider the possibility of unknown attacks. Instead, the classifier should be rewarded for outputting a vector with low probabilities for each known attack that it was trained on, and this will result in a higher validation score for benign traffic.

3.3 FTG-NET-E Results

This section presents the experimental results to assess the performance of the proposed FTG-Net-E model. The experiments were run on a Ubuntu 20.04 LTS workstation with Dell PowerEdge R760Intelő Xeonő Gold 6438M processor and NVIDIA A30X graphics card. The main libraries for implementing FTG-Net-E are TensorFlow, NumPy, PyTorch, Scapy, PyTorch Geometric, and Pandas.

The proposed structure deals with a binary classification task, in which indicators are established through a confusion matrix and in which it is possible to formulate the following metrics:

$$ACC = \frac{x_{correct}}{x_{total}} = \frac{TP + TN}{TP + TN + FP + FN} \tag{3.2}$$

$$recall = \frac{TP}{TP + FN} \tag{3.3}$$

$$precision = \frac{TP}{TP + FP} \tag{3.4}$$

$$F1 = 2 \times \frac{precision \times recall}{precision + recall} \tag{3.5}$$

where ACC stands for accuracy and TP, TN, FP, FN stand for true positive, true

negative, false positive, and false negative, respectively.

In classification problems, the malignant class is usually more important, and recall becomes the main metric. But in certain cases, like in DDoS Detection, it is not possible to state that one class is more significant than the other. In such scenarios, combining metrics computed considering each class as positive and averaging the results is a common approach. Weighted F1-Score metric is also used, where F1-Scores are combined by weighted averaging using the number of samples of the positive classes as weights. This helps in achieving a balanced classification that doesn't block legitimate traffic while detecting malicious packets.

We investigated the impact of two hyperparameters, learning rate and hidden size, on the performance of our ensemble model. We have trained FTG-Net-E with a learning rate of either 0.001 or 0.0001, while we have changed the hidden size of our models between 128, 256, and 512 neurons.

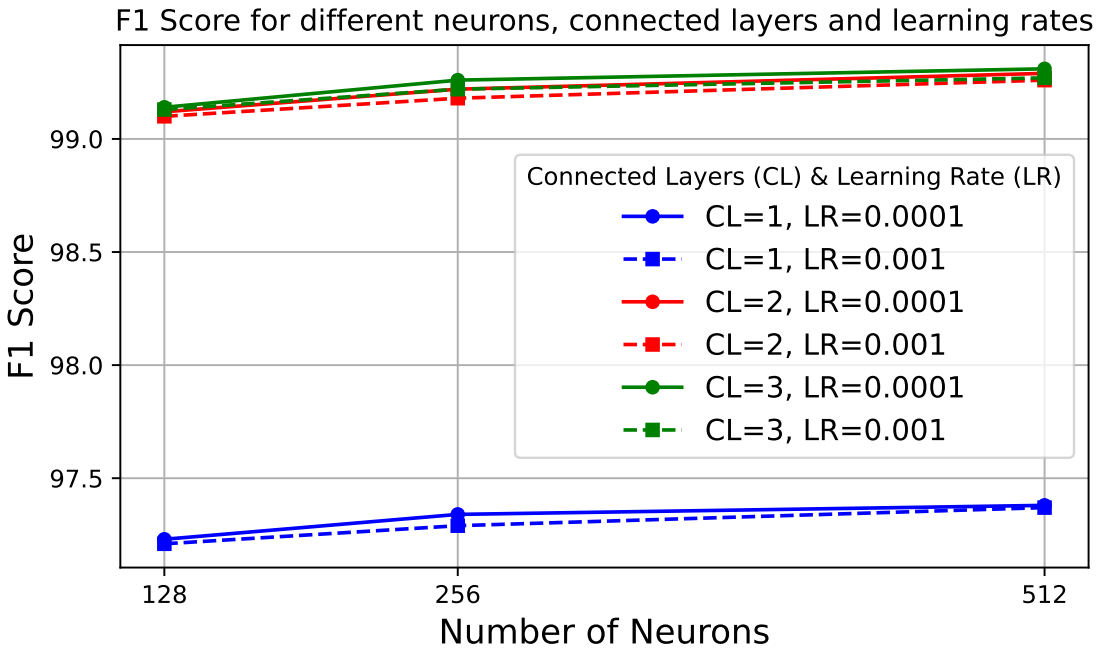


Figure 3.3: FTG-Net-E mean F1 scores achieved by FTG-Net-E for all 6 possible combinations of learning rate and hidden size

Figure 3.3 shows the mean F1 scores achieved by FTG-Net-E on both datasets (CIC-IDS-2017, and CSE-CIC-IDS-2018)) when trained with all six possible combinations of learning rate and hidden size. As shown in the table, the ensemble model achieves the highest F1 of 99.29% with a learning rate of 0.0001 and a hidden size of

512, making these hyperparameters the best for our ensemble model. Nonetheless, in all cases the F1 score is above 99%, making it possible to use models that are lighter (lower hidden size) and able to be trained in lower time (higher leraning rate), without a significant loss of accuracy. As observed, the model achieves the highest F1 score of 99.29% with a learning rate of 0.0001 and a hidden size of 512 neurons. However, it's important to note that all configurations achieved above 99% F1 scores, indicating the model's robustness to these specific hyperparameter choices. This allows for flexibility in selecting models based on specific deployment requirements, such as prioritizing faster training times (using a higher learning rate) or lower computational resources (using a smaller hidden size) with minimal impact on accuracy.

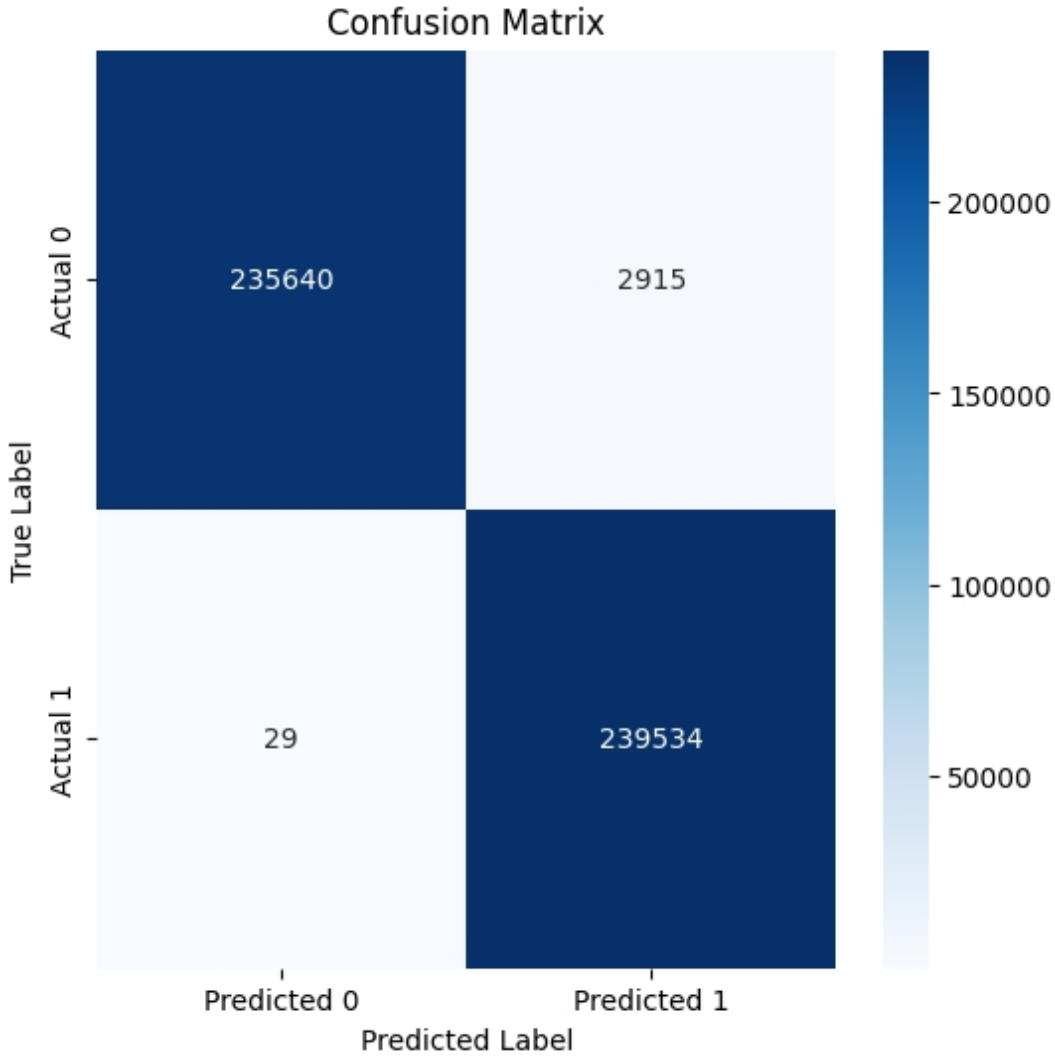


Figure 3.4: FTG-Net-E confusion matrix

Figure 3.4 shows the confusion matrix when FTG-Net-E is trained with the

best hyperparameter. The ensemble model very high F1 score testifies its ability to accurately classify both positive and negative cases. The model correctly predicted 235640 positives and 239534 negative cases. There were 2915 false positives and 29 false negatives. Our ensemble model correctly predicted 235640 positives and 239534 negative cases. There were 2915 false positives and 29 false negatives. The high F1 score and low false positives and negatives indicate that the ensemble model is a very effective classifier for this task.

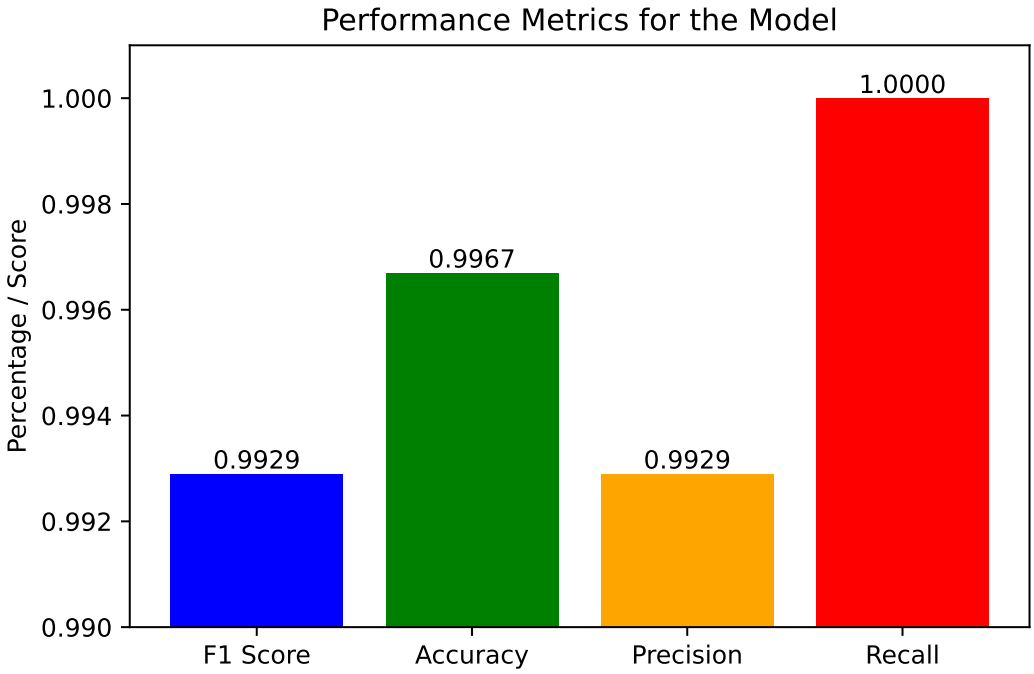


Figure 3.5: Performance metrics for the best FTG-Net-E model

Figure 3.5 presents all the performance metrics for the best FTG-Net-E. The ensemble model achieved an F1 score of 0.9929, accuracy of 0.9967, precision of 0.9929, and recall of 1.0. These results indicate that the model can accurately predict both positive and negative instances with a high degree of accuracy. The model can also minimize false positives and negatives, which is important for many real-world applications. This high performance is because it combines the predictions of multiple GNN models with different hyperparameters. This averaging process helps reduce the prediction variance and improve the model overall performance.

Figure 3.6 presents the Receiver Operating Characteristic (ROC) curve analysis for our GNN models with different hyperparameter settings. The single ROC curve

plot showcases the performance of each GNN model across these parameter configurations. Each curve represents the ROC curve for a specific parameter setting, and the area under the curve Accuracy (AUC) is used as a metric to quantify the model's discriminative power. As seen in the plot, the ROC curves demonstrate the trade-off between the true positive rate (sensitivity) and the false positive rate (1-specificity) for various threshold values. The dashed line represents the baseline scenario where the model's predictions are no better than random chance. Our analysis reveals that these ensemble GNN models consistently achieve high AUC values, even with distinct hyperparameter settings, indicating their effectiveness in distinguishing malicious and legitimate traffic. The consistent performance of our models across different hyperparameter configurations underscores the reproducibility of our approach and suggests that our models can generalize well to new data.

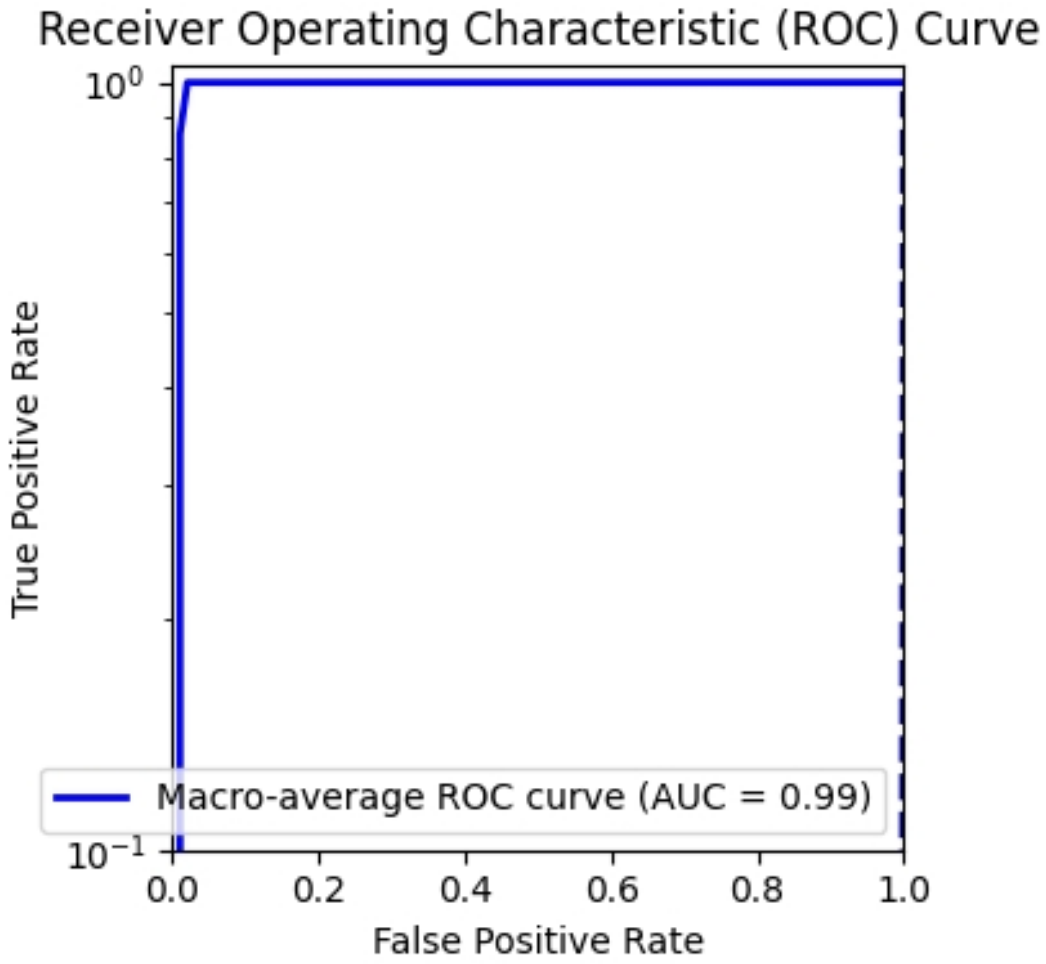


Figure 3.6: ROC Curve with different parameters

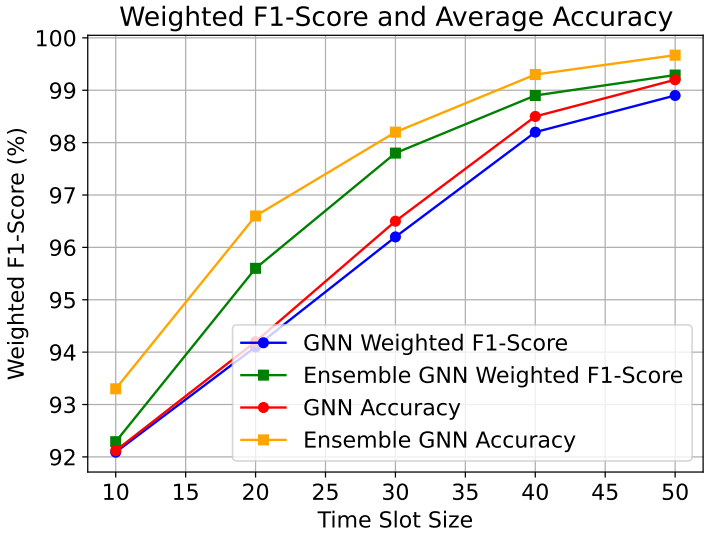
3.3.1 Inference Time and Model Comparison

We evaluated the performance of FTG-Net-E using a 5-fold cross-validation method to determine its effectiveness. The dataset was randomly partitioned into five segments for each fold, with a model trained on four segments and the remaining one serving as the validation set. The configuration details include a 5-second time slot size, an 8-dimensional vector as the output of the Flow GNN, and a 64-hidden channel output for the GCN layers. Initially, the Flow Graph nodes are made up of only one element which corresponds to the length of the packet.

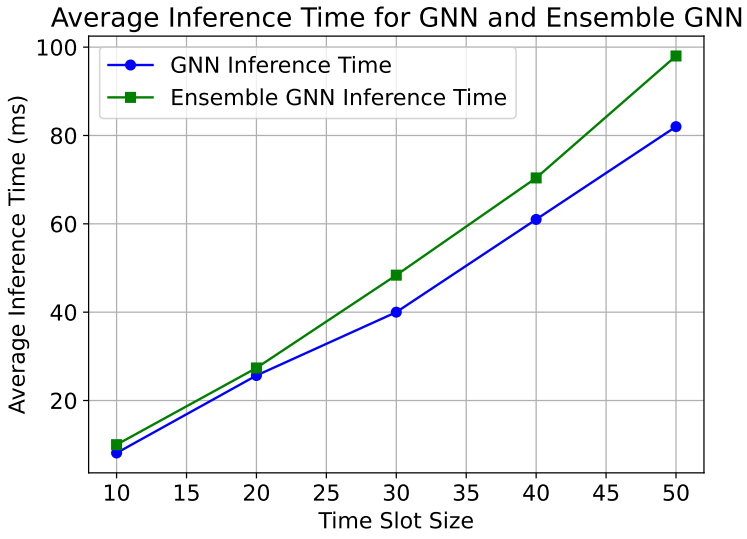
In our approach, the size of the time slot needs to be configured carefully. A larger time slot provides a broader view of the traffic topology, which contains more information to detect potential DDoS attacks. However, the size of the time slot is inversely proportional to the system's responsiveness. A larger time slot may not allow the system to detect attacks in a timely manner, which could result in significant damage. Moreover, with large Traffic Graphs, the inference time increases. It's possible that a short time period may not provide sufficient data to accurately classify traffic. To address this, we tested the performance of four different time slot sizes by evaluating their average inference time and weighted F1-Score.

Figure 3.7 (a) compares the accuracy and weighted F1-score of the simple GNN and Ensemble GNN models as a function of the time slot size. The simple GNN model achieves an accuracy of 93.5% weighted F1-score of 92.9% for a time slot size of 10 and increases accuracy to 99.14%, the same increase seen in weighted F1-score 99.13% for a time slot size of 50. The Ensemble GNN model achieves a weighted F1-score and accuracy of 93.9% and 92.29%, respectively, for a time slot size of 10 and increases to 99.10% and 99.29% for a time slot size of 50. FTG-Net-E consistently outperforms FTG-Net in the weighted F1-score as expected.

Figure 3.7 (b) shows the average inference time of the GNN and Ensemble GNN models as a function of the time slot size. The GNN model has an average inference time of 8.12 ms for a time slot size of 10 and increases to 82 ms for a time slot size of 50. The Ensemble GNN model has an average inference time of 10 ms for a time slot size of 10 and increases to 98 ms for a time slot size of 50. Noteworthy, up to a



a)



b)

Figure 3.7: (a) Weighted F1-Score and accuracy for GNN and Ensemble GNN (b) Average inference time for GNN and Ensemble GNN models results using different time slot sizes

time slot size of 20, FTG-Net-E has an almost negligible increase of inference time, while having a higher F1 score, making it the best choice in this range. In particular, for a time slot of 20, it increases the F1 score of more than 1%.

The results from the best models for each training phase are summarized in Table 3.1, with a comparison with three models from the literature.

FTG-Net-E shows a higher accuracy in respect to all considered models, and an F1 score that is surpassed by less than 0.5%. Noteworthy, FTG-Net-E achieves this by solely considering the network structure and without relying on numerous stateful features. Hence, our approach significantly reduces computational complexity and system latency, demonstrating the advantage of focusing solely on the network

Table 3.1: Comparison of Results

Method	Accuracy	F1-score	Precision
LUCID [47]	0.9967	0.9966	
GLD-Net [58]	0.9940	0.9920	
GraphDDoS [56]	0.9959	0.9959	
N-STGAT [71]	0.9788	0.9869	
Anomal-E [66]	0.9412	0.9630	
E-GraphSAG [61]	0.95472	0.9719	
ST-GCN [158]	0.9110	NA	
FTG-Net [155]	0.9914	0.9913	0.9908
FTG-Net-E	0.99675	0.9929	0.99256

structure rather than incorporating numerous stateful features.

To testify how our approach is significantly more computationally efficient than previous methods consider for example the CICIDS2018 dataset. FTG-Net-E takes 10 minutes to train a model with 8.3 million parameters, while LUCID [47] takes 1 hour to train a model with the same number of parameters. This computational efficiency makes our approach much more practical for real-world applications.

3.3.2 Limitations and Potential Challenges

The experimental results showcased the efficacy of FTG-Net-E in mitigating DDoS attacks. However, deploying such a system in real-world scenarios may encounter several challenges and limitations. Scaling FTG-Net-E to large-scale network environments might pose computational and memory challenges. Evaluating scalability issues depends on hardware dependent and requires optimizing model architecture, training procedures, and inference strategies. Real-world deployment relies on the availability and quality of network traffic data. Strategies for collecting and augmenting of network data and representative datasets need also a solid consideration. Network traffic patterns evolve due to changes in network configurations, user behaviors, and emerging threats. We can also consider a factor of implementing FTG-Net-E in resource-constrained environments, such as edge devices or IoT networks, which may necessitate model compression, optimization, or distributed learning approaches to reduce computational and memory overhead. We used to minimize the

effect of adversarial attacks but ensuring the robustness and security of FTG-Net-E against adversarial attacks, model poisoning, and data tampering is important to evaluate. Robust training techniques, model validation procedures, and adversarial defense mechanisms need more exploration. Future research endeavors should focus on mitigating these challenges to facilitate the effective deployment of FTG-Net-E in real-world cybersecurity ecosystems.

3.4 A proposed Nested GNN for IDS

The growing complexity of network attacks has outpaced the capabilities of traditional IDS, which often rely on flat data structures that fail to capture complex relationships within networks. To address this limitation, we propose IDS-NGNN, a novel IDS that integrates hardware-offload SmartNIC preprocessing with a nested graph neural network (NGNN) architecture. Unlike standard GNN, IDS-NGNN jointly captures local and global dependencies using a three-layer design: an internal GNN for host-level activity, a nested graph module for hierarchical aggregation, and an external GNN for inter-host communication. SmartNIC acceleration enables efficient real-time processing of large-scale graph-structured network data at the edge. We evaluate IDS-NGNN at section 3.6.1 on six public IDS datasets, including CIC-IDS-2017, CSE-CIC-IDS-2018, and ToN-IoT. Experimental results demonstrate that IDS-NGNN achieves up to **95% accuracy and 92% F1-score**, while maintaining efficiency suitable for real-time 100 Gbps deployments.

3.5 IDS-NGNN Architecture

In this section, we introduce the **IDS-NGNN** framework to address the limitations of traditional GNNs in network intrusion detection. IDS-NGNN captures both local and global relational patterns within network traffic. The core idea of IDS-Nested Graph Neural Networks (NGNN) is constructing a two-tier nested graph structure as an **internal graph** for each host, capturing its localized communication behavior and internal activity. As an **external graph**, constructed over hosts, capturing

inter-host communication patterns.

Algorithm 1 IDS-NGNN Forward Pass

Require: $Graphbatch\mathrm{D}with nodes\mathrm{x}, edgeindices\mathrm{E}, labels\mathrm{y}, and batchvector\mathrm{b}$
Ensure: $Nodepredictions\mathrm{O}, internalembeddings\mathrm{I}, externalembeddings\mathrm{Ext}, graphlevelembe$

1: $I \leftarrow InternalGATLayer(x, E)$ ▷ Compute internal node embeddings using GAT
2: $N \leftarrow NestedGraphLayer(I, E)$ ▷ Aggregate internal embeddings (map-reduce step)
3: $E_{\text{ext}} \leftarrow \text{ExternalGATLayer}(N, E)$ ▷ Compute external node embeddings using GAT
4: $G \leftarrow \text{GlobalMeanPool}(E_{\text{ext}}, b)$ ▷ Aggregate node embeddings to obtain a graph-level embedding
5: $O \leftarrow \text{Classifier}(E_{\text{ext}})$ ▷ Predict node labels (normal vs. compromised)
return O, I, E_{ext}, G

This hierarchical design enables the model to continuously analyze fine-grained subgraph features and higher-level inter-host interactions. The three layers of IDS-NGNN are described in Figure 3.8. Internal GNN applies a GAT-based message passing operation over each internal graph to learn node embeddings reflecting intra-host behavior. The Nested Graph Layer aggregates internal graph embeddings and integrates them into the corresponding nodes of the external graph, providing node-level representations. An external GNN learns host-level embeddings based on the external graph topology and the aggregated internal representations, enabling anomaly detection at the network level.

The overall forward pass of our IDS-NGNN framework is described in Algorithm 1, which explains how internal node embeddings are computed using a GAT layer, aggregated via the nested graph layer, and subsequently processed by an external GAT layer to provide node-level predictions. By integrating these components, IDS-NGNN can detect unexpected anomalies within the local activity of hosts and their interactions with other hosts in the network.

The system is end-to-end optimized, allowing the model to learn from internal and external graphs concurrently, thus improving detection accuracy in complex network environments.

3.5.1 IDS-NGNN Hierarchical Representation

IDS-NGNN employs a hierarchical GNN architecture at the base level, a GNN operates over rooted subgraphs (e.g., 1-hop ego-networks) centered around each host node w. These subgraphs are denoted as Gh_w^v, where v is a node in the subgraph rooted at w. The representation of node v at time step $t+1$ is updated as follows:

$$h_{t+1}^{v,Gh_w^v} = U_t(h_t^{v,Gh_w^v}, m_{t+1}^{v,Gh_w^v}), \tag{3.6}$$

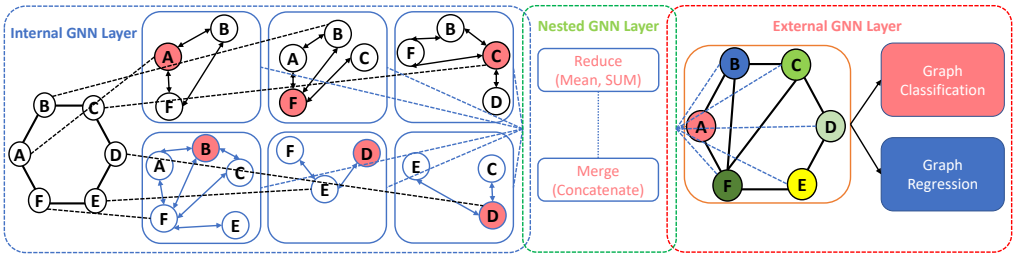


Figure 3.8: Architecture of the IDS-NGNN model, composed of three layers. The blue rectangle (*left*) represents the internal layer, where node-level ego-nets are extracted. The green rectangle (*center*) shows the nested and merge layers, where local subgraphs are processed and embeddings are aggregated. The red rectangle (*right*) denotes the external GNN layer, which learns a global representation from the merged embeddings.

where h_{t+1}^{v,Gh_w^v} is the updated embedding of node v in subgraph Gh_w^v, $U_t(\cdot)$ is the node update function at iteration t, m_{t+1}^{v,Gh_w^v} is the aggregated message from the neighbors of v in Gh_w^v. The message aggregation is defined as:

$$m_{t+1}^{v,Gh_w^v} = \sum_{u \in N(v|Gh_w^v)} M_t(h_t^{v,Gh_w^v}, h_t^{u,Gh_w^v}, e_{vu}), \tag{3.7}$$

where $N(v|Gh_w^v)$ denotes the neighbors of node v within subgraph Gh_w^v, $M_t(\cdot)$ is the message function that aggregates features from v and its neighbor u, e_{vu} is the edge feature (e.g., communication type or frequency) between v and u.

Once node embeddings are obtained, a local pooling function $R_0(\cdot)$ aggregates them to form a single embedding for the subgraph (host-level representation):

$$h_G^w = R_0(\{h_T^{v,Gh_w^v} \mid v \in Gh_w^v\}), \tag{3.8}$$

where T is the total number of message-passing iterations, h_G^w is the pooled embed-

ding of the host subgraph rooted at w.

At the global level, another GNN treats each h_G^w as a node in a higher-level graph G, representing the full network. A global pooling function $R_1(\cdot)$ computes the final graph representation:

$$h_G = R_1(\{h_G^w \mid w \in G\}), \tag{3.9}$$

where h_G encodes both intra-host activity and inter-host relationships.

This nested GNN design enables IDS-NGNN to hierarchically learn from fine-grained host-level behavior and high-level network interactions, improving robustness and expressivity in complex intrusion scenarios.

3.5.2 Internal Layer

Let $A_{in} \in \mathbb{R}^{N\times n\times n}$ and $H_{in} \in \mathbb{R}^{N\times n\times m}$ represent the adjacency matrices and node features for N internal graphs, each with at most n nodes and m-dimensional features.

The node features for the $(i-th)$ internal graph are:

$$h_i = \left\{\overrightarrow{h_i^a}, \overrightarrow{h_i^b}, \ldots, \overrightarrow{h_i^{n_i}}\right\}, \quad \overrightarrow{h_j^i} \in \mathbb{R}^m, \tag{3.10}$$

where $n_i \leq n$. The internal GNN module computes updated embeddings for the i-th graph:

$$\left\{\overrightarrow{h_i^{a'}}, \ldots, \overrightarrow{h_i^{n_i'}}\right\}' = GNN_{in}\left(h_i\right). \tag{3.11}$$

Assuming graph convolutional layers, a typical layer operation is:

$$H^{l+1} = \sigma\left(\tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}H^lW^l\right), \tag{3.12}$$

where $\tilde{A} = A + I$ is the adjacency matrix with self-loops, $\tilde{D}$ is the degree matrix, W^l are learnable weights, and σ is an activation function like ReLU.

The internal GNN outputs $H'_{in} \in \mathbb{R}^{N\times n\times m'}$, where m' is the new feature dimen-

sion. All internal graphs share the same GNN architecture and parameters, enabling efficient and consistent learning.

3.5.3 Nested Graph Layer

An intermediate graph layer is placed between the internal and external graph layers, where the newly generated internal node embedding H'_{in} is used to update the external graph with the original external node feature vector H_{out}, as seen in Figure 3.8. $H_{out} = \left\{\overrightarrow{H_1}, \overrightarrow{H_2}, \overrightarrow{H_3}, \ldots, \overrightarrow{H_N}\right\}, \overrightarrow{H_l} \in \mathbb{R}^m$ where M denotes the number of features of each external node. The *reduce* and *merge* operations are carried out within the stacked graph layer. Here, *reduce*() denotes the reduction function (e.g., mean, sum), and Equation 3.13 illustrates how the *reduce* operation generates an embedding for the $(i-th)$ internal network.

$$\overrightarrow{h_l''} = reduce(\overrightarrow{h_l{}^{a'}}, \overrightarrow{h_l{}^{b'}}, \ldots, \overrightarrow{h_l{}^{n_{i'}}}) \tag{3.13}$$

The merge operation combines the original features of the external nodes with the newly created features from the internal network, as produced by the stacked graph layer.

To do this, Equation 3.14 shows how to define the updated external node feature $\overrightarrow{H'_l}$, where *merge*() stands for the merge function, which in our current design is the concatenation operation.

$$\overrightarrow{H'_l} = merge(\overrightarrow{H_l}, \overrightarrow{h_l''}) \tag{3.14}$$

Equation 3.15 shows the construction of the $N(.)$ layer of the layered graph at the $(i-th)$ external node.

$$\begin{aligned} N(\overrightarrow{H'_1}, \overrightarrow{h_l{}^{a'}}, \ldots, \overrightarrow{h_l{}^{n_{i'}}}) = \text{merge}(&\overrightarrow{H_l}, \\ &\text{reduce}(\overrightarrow{h_l{}^{a'}}, \overrightarrow{h_l{}^{b'}}, \ldots, \overrightarrow{h_l{}^{n_{i'}}})) \end{aligned} \tag{3.15}$$

3.5.4 External GNN Layer

At this stage, the nested graph has been fully constructed with attributed nodes. The next step is to apply the external GNN module, as illustrated in Figure 3.8. The final external node embedding set H''_{out} is produced by the external GNN module, which is used as input for the updated external graph feature matrix.

Such as $H''_i = \left\{\overrightarrow{H_1'}, \overrightarrow{H_2'}, \ldots, \overrightarrow{H_N'}\right\}$ and the original external graph adjacency matrix A_{out}.

For all external nodes, the output node embedding set H''_{out} is determined by Equation 3.16 using the $GNNout(.)$ layer as the externalmost layer of the network. The output node embedding size M' is represented by the dimensions of the vector H''_{out}, which are $[N, M']$.

$$\left\{\overrightarrow{H_1''}, \ldots, \overrightarrow{H_N''}\right\} = GNN_{out}\left\{\overrightarrow{H_1'}, \ldots, \overrightarrow{H_N'}\right\} \tag{3.16}$$

Mathematically, this is expressed as Equation 3.12 if we use graph convolution for the external layers as well.

We perform an embedding analysis, as described in Algorithm 2, by partitioning internal node embeddings according to their labels and computing class-wise mean vectors to distinguish benign from compromised hosts.

Algorithm 2 Embedding Analysis for Benign vs. Compromised

Require: Internal embeddings $I = \{\mathbf{h}_i\}_{i=1}^{n}$, node labels $y \in \{0,1\}^n$

1: $M_{\text{benign}} \leftarrow \{i \mid y_i = 0\}$
2: $M_{\text{compromised}} \leftarrow \{i \mid y_i = 1\}$
3: $\boldsymbol{\mu}_{\text{benign}} \leftarrow \frac{1}{|M_{\text{benign}}|} \sum_{i \in M_{\text{benign}}} \mathbf{h}_i$
4: $\boldsymbol{\mu}_{\text{comp}} \leftarrow \frac{1}{|M_{\text{compromised}}|} \sum_{i \in M_{\text{compromised}}} \mathbf{h}_i$ **return** $\boldsymbol{\mu}_{\text{benign}}, \boldsymbol{\mu}_{\text{comp}}$

3.6 IDS-NGNN Experiments and Results Analysis

For the experiments on the NGNN-based IDS, we used two Dell R760 servers, each equipped with an Intelő Xeonő Gold 5418Y processor and a SmartNIC Converged Accelerated card with an NVIDIA A30x GPU. The servers were connected via high-speed 100 Gbps Ethernet and Ubuntu 22.04 as the operating system. The multi-GPU training was performed using PyTorchs DDP, which synchronized the model's weights and gradients across the two GPUs using NVIDIA flare library. Each GPU processed a distinct subset of the data in parallel, reducing training time and enabling faster convergence on large-scale datasets. Training, validation, and testing are performed at a 1:1:3 ratio. Each experiment is repeated 10 times, and the mean is reported. We classify models according to their precision, macro F1 score, and weighted F1 score. The percentage of occurrences that are appropriately labeled is used to calculate accuracy.

3.6.1 Performance Analysis

A summary of the dataset statistics is presented in Table 3.2, which includes results from five benchmark IDS datasets. On CIC-IDS-2017, Base-GCN achieves 92% accuracy, while IDS-NGNN(GAT) increases this to 95%. Similarly, Base-GraphSAGE

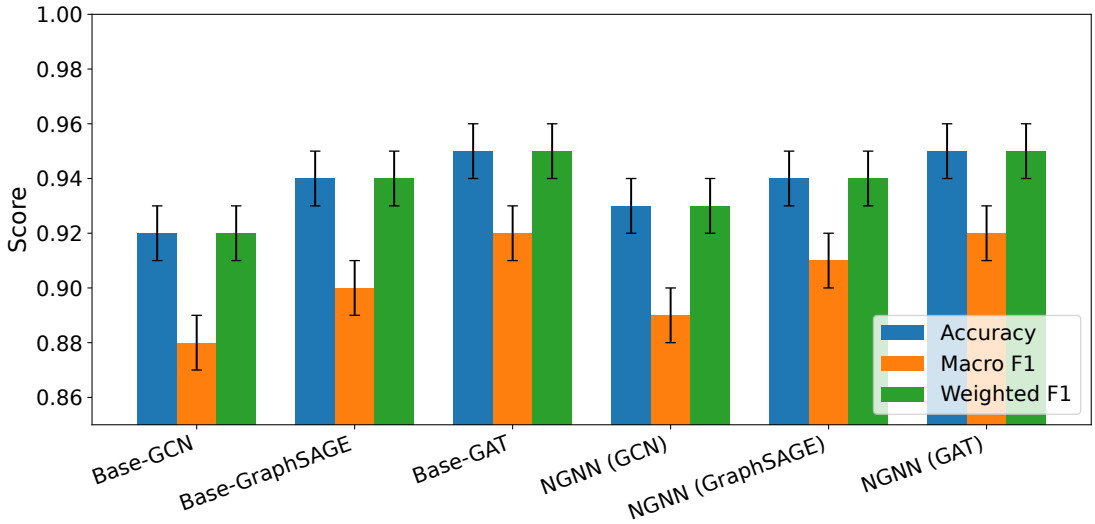


Figure 3.9: Aggregated performance results of each model.

reaches 94%, and Base-GAT 95%, with NGNN variants sustaining consistent improvements. On newer IoT-oriented datasets, IDS-NGNN continues to provide small but stable gains, confirming that nested reasoning generalizes across domains. We contrast Base-GCN, Base-GraphSAGE, and Base-GAT, three widely used GNN techniques. On the nested graph, we evaluated three different IDS-NGNN variants by employing GCN, GraphSAGE, or GAT as the internal and external layers within the architecture.

Table 3.2: Accuracy (%) across five benchmark IDS datasets. Values are mean±std.

Dataset	#Graphs	Avg. #nodes	GCN		GraphSAGE [60]	
			Base	Nested	Base	Nested
CIC-IDS-2017	1113	39.1	92.0±0.1	93.0±0.1	94.0±0.1	94.0±0.1
CSE-CIC-IDS-2018	344	14.3	56.4±7.1	57.0±7.3	57.0±5.5	56.7±8.1
ToN-IoT	500	25.0	60.0±4.0	61.5±5.0	58.0±3.5	59.0±4.0
CICIoT2023	750	30.0	65.0±3.5	66.0±3.0	64.5±4.0	65.5±3.5
CICIoV2024	1000	50.0	70.0±4.5	71.5±4.0	69.0±5.0	70.5±4.5

IDS-NGNN improves the aggregated accuracy to 95%, the macro F1 score to 92%, and the weighted F1 score to 92%, as shown in Figure 3.9. In terms of accuracy and F1 metrics, this corresponds to a ∼3–4% gain over the strongest baseline. All IDS-NGNN configurations also exhibit consistent macro F1 improvements over their base models, demonstrating that the learned hierarchical embeddings effectively capture both host-level and network-level dependencies. The performance difference between

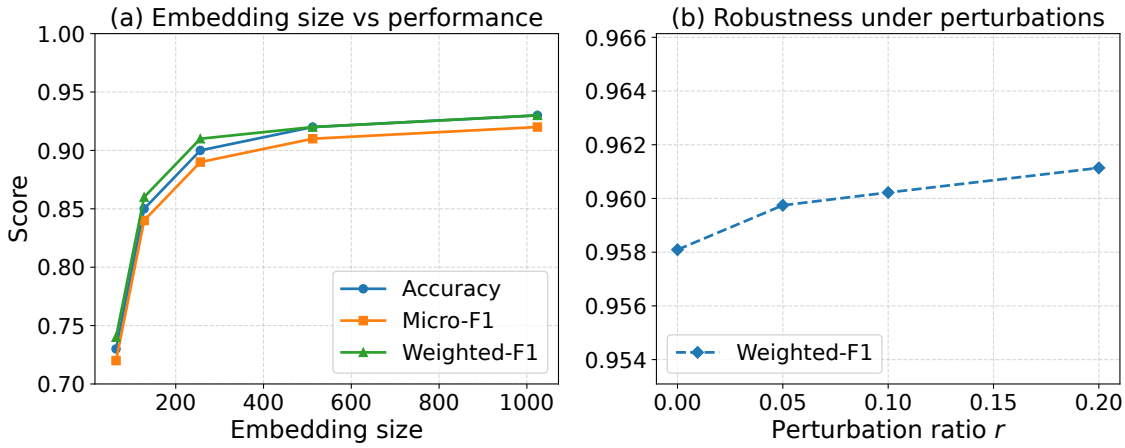


Figure 3.10: IDS-NGNN performance analysis. (a) Accuracy, Micro-F1, and Weighted-F1 across embedding sizes. (b) Structural robustness under edge perturbations r, showing less than 2% degradation up to $r = 0.20$.

the NGNN variants is modest, with the best results obtained using GAT for both the internal and external layers. Leveraging attention enables GAT to focus on the most critical nodes and edges, resulting in superior detection performance.

For comparison, the manually generated feature-based baseline (Base-GCN) achieves a weighted F1 of only 74%. In contrast, IDS-NGNN without explicit host attributes achieves 83% weighted F1, suggesting that the process-alive graph (internal graph) already encodes sufficient host behavioral characteristics for intrusion detection.

The performance of $N(\text{GCN}, \text{GCN})$ across various embedding sizes ranging from 64 to 1024 is illustrated in Figure 3.10(a). IDS-NGNN maintains consistent performance across all three metrics despite changes in embedding size. Accuracy improves from 73% at size 64 to 93% at size 1024, confirming the effectiveness of the learned embeddings. Even small embeddings deliver competitive performance, highlighting scalability to resource-constrained settings. Figure 3.10(b) assesses robustness against structural perturbations: The Weighted-F1 score decreases slightly, from 0.958 at $r = 0.00$ to approximately 0.954 at $r = 0.20$, representing a relative reduction of less than 2%. This demonstrates resilience to adversarial edge insertions and deletions, as well as stability under graph manipulations.

3.6.2 Complexity Analysis

We next analyze computational complexity and runtime efficiency. Table 3.3 reports FLOPs per forward pass, where IDS-NGNN adds only +50–80% overhead compared to flat GNNs, keeping all models under 10 MFLOPs per graph. This modest increase justifies the three-part design, which expands receptive fields without heavy cost.

To complement FLOPs, Table 3.4 reports wall-clock preprocessing, training, and inference times on Xeon CPU, NVIDIA A30 GPU, and SmartNIC+GPU (A100) pipelines. On the 6k-graph workload, GPU training is $\sim$2.4$\times$ faster than CPU (124.9s $\rightarrow$ 51.7s), while SmartNIC+GPU reduces training further to 48.6s. Inference latency remains sub-millisecond, dropping from 0.402 ms/graph on GPU to 0.352 ms/graph with SmartNIC integration. FLOPs scale with N, E, and H: larger/denser 6k graphs explain higher preprocessing time and inference latency. GPU acceleration is most evident in models with deeper and attention-heavy architectures (e.g., GAT); short, shallow runs underutilize the GPU. Despite scale-up, IDS-NGNN maintains sub-ms inference per graph, confirming suitability for real-time 100 Gbps intrusion detection.

Table 3.3: Per-forward FLOPs (MFLOPs) on CICIDS2017 (avg. graph $N = 20$, $E = 82$, $F = 84$, $H = 128$, $A = 4$ heads).

Model	2-Layer	+Nested (SUM/CONCAT)
GCN	1.24 M	1.94 M / 2.25 M
GraphSAGE	1.22 M	1.91 M / 2.22 M
GAT (4 heads)	4.96 M	7.69 M / 8.98 M

Table 3.4: Runtime benchmarks on CICIDS2017 (6,000 graphs, batch size 512, GAT, 15 epochs).

Setting	Preprocess	Training	Inference
CPU-only (Xeon)	320.8 ms	124.9 s	—
GPU (A30)	318.3 ms	51.7 s	0.402 ms
SmartNIC+GPU (A100) *	$\ll$325.6 ms	48.6 s	0.352 ms

3.6.3 On the Feasibility of ML and GNN Offloading to SmartNICs

The proposed architecture deliberately separates learning-based detection from data-plane enforcement. While DPU-based mitigation focuses on deterministic, wire-speed packet handling using stateful tracking and signature-based mechanisms, IDS-NGNN targets a different operating regime: cross-flow correlation and structural traffic analysis.

In principle, the SmartNIC provides architectural features (e.g., DPA engines, programmable accelerators, and high-bandwidth memory interfaces) that could support limited forms of machine learning inference. We therefore considered offloading GNN-based detection to the SmartNIC. However, several practical constraints motivated the chosen design.

First, graph neural networks exhibit irregular memory access patterns and require frequent aggregation across variable-size neighborhoods. This leads to memory-bound execution and limited data reuse, which is poorly matched to the execution model and memory hierarchy of current SmartNICs. Second, GNN inference requires maintaining graph state and adjacency information for large numbers of concurrent flows, which quickly exceeds the on-board memory capacity available on DPUs under realistic deployment constraints. Third, inference latency for non-trivial GNN models is incompatible with strict wire-speed processing requirements, particularly at 100–400 Gb/s.

For these reasons, IDS-NGNN is executed on GPU-accelerated/host-side compute resources, where parallelism, memory capacity, and programming flexibility are sufficient to support graph-based learning at scale. The SmartNIC instead acts as an enforcement and filtering layer, executing lightweight, deterministic operations such as flow tracking, thresholding, and selective packet forwarding. This division of workload ensures that complex detection logic does not compromise data-plane latency or throughput guarantees.

From a system perspective, this design reflects a conscious trade-off between detection expressiveness and enforcement determinism. While future SmartNIC

generations may support more capable ML offloading, the architecture proposed in this thesis prioritizes deployability and predictable performance in real high-speed networks.

3.7 Chapter Summary

This chapter investigated graph neural network paradigms for network security analytics, with a focus on DDoS detection and high-data-rate intrusion detection. First, we presented FTG-Net-E, an ensemble-based extension of the FTG-Net[155] framework for DDoS detection. FTG-Net-E exploits a hierarchical traffic representation in which packet-level interactions are encoded in Flow Graphs and then propagated to a Traffic Graph to capture cross-flow dependencies. Unlike methods that depend heavily on high-dimensional stateful features, FTG-Net-E is primarily structure-driven and is trained end-to-end so that Flow GNN embeddings directly parameterize the Traffic GNN input space. By aggregating multiple lightweight GNN models via ensemble learning, the proposed design improves robustness and reduces variance relative to a single model. On CIC-IDS2017 and CICIDS2018, FTG-Net-E achieves state-of-the-art performance, reaching an F1-score of 0.9929 and an accuracy of 0.9967 while maintaining low false-positive and false-negative rates.

Second, we introduced IDS-NGNN, a nested GNN-based IDS that jointly models intra-host activity and inter-host communication through a three-stage hierarchy: an internal graph module, a nested aggregation layer, and an external graph module. This design addresses the limitations of flat GNN formulations by explicitly capturing local host behavior and global network interactions within a unified architecture. Experimental results across multiple public IDS datasets show consistent, albeit dataset-dependent, improvements over strong baselines (e.g., GCN, GraphSAGE, and GAT), with the best configuration achieving up to 95% accuracy and 92% F1-score on CIC-IDS-2017. Complexity and runtime analyses further indicate that the nested design introduces only modest computational overhead while preserving sub-millisecond inference latency per graph under GPU acceleration, supporting real-time deployments.

Finally, we discussed the feasibility of offloading learning-based inference to SmartNIC-class devices. While SmartNICs are well-suited for deterministic, wire-speed enforcement and selective filtering, the irregular memory access patterns and state requirements of GNN inference make full offload challenging under current hardware constraints. Accordingly, the chapter motivates a practical split architecture in which the SmartNIC performs lightweight preprocessing and enforcement, while GPU/host-side compute executes graph-based inference when expressiveness and cross-flow correlation are required.

Chapter 4

Hardware-Accelerated DDoS Mitigation Using Programmable DPUs

In this chapter, we investigate a hardware-accelerated approach for mitigating DDoS attacks using programmable DPUs. As cloud and web service providers continue to face evolving DDoS threats, traditional mitigation methods struggle with high latency, inefficient traffic filtering, and excessive resource overhead. The solution proposed in this chapter leverages the power of DPUs to address these challenges by performing attack detection and mitigation directly within the DPU itself. This approach eliminates the need for expensive additional network hardware and reduces the strain on the host system, making it a robust and scalable solution for high-speed networks. The experimental evaluation of this approach is presented in Section 4.2.

4.1 Proposed Methodology of Hardware-Accelerated Framework

This section describes the system design choices required to execute DDoS detection and mitigation at wire speed on a programmable DPU, focusing on stateful processing, queue management, and deep packet inspection. This thesis presents a

novel hybrid framework that performs live traffic analysis and intrusion detection on the NVIDIA BlueField-2 DPU by integrating DPDK libraries, regular expression (RegEx), and the Suricata intrusion detection system. This framework leverages hardware acceleration to perform DPI for intrusion detection and efficient blocking of malicious traffic. The system is designed to offload the DPI processing tasks to the DPU/SmartNIC, freeing up the host CPU and ensuring scalability for high-throughput networks. The detailed flow of DPI processing, which utilizes the integration of the DOCA QoS engine and DPU acceleration, is outlined in Algorithm 3, and the algorithm's functions and variables are explained in section 4.1.2.

Algorithm 3 DPI Processing with DOCA QoS Integration

Packet Ingress:
Let $P = \{p_1, p_2, ..., p_n\}$ be the set of incoming packets
Classify and prioritize each packet: $f_{\text{QoS}}(p_i)$
Assign packet p_i to queue Q_j based on QoS function: $Q_j = f_{\text{QoS}}(p_i)$
Stateful Tracking (CT):
Track flow f_i for each packet p_i: $f_i = f_{\text{track}}(p_i)$
Use multithreading for parallel flow tracking: $f_i^T = f_{\text{track}}^T(p_i)$
Update flow state counters: $C_{f_i} = C_{f_i} + 1$ for each packet in flow f_i
Monitor flow rate using hardware meters: $M_{f_i} = f_{\text{meter}}(f_i)$
DPI Processing (Queue-Based):
for each packet p_i in queue Q_j **do**
    Perform packet parsing: $h_i, p_i = f_{\text{parse}}(p_i)$
    Signature matching and anomaly detection: $f_{\text{sig}}(p_i), f_{\text{anomaly}}(p_i)$
    Perform DPI using hardware acceleration: $f_{\text{DPU}}(p_i)$
    if $f_{\text{sig}}(p_i) = 1$ **or** $f_{\text{anomaly}}(p_i) = 1$ **then**
        Drop packet p_i: $p_i \notin P_{\text{valid}}$
    else
        Forward packet p_i: $p_i \in P_{\text{valid}}$
    end if
end for
Egress:
Forward valid packets P_{valid} to TX interface
Update hardware egress meter: $M_{\text{egress}} = f_{\text{meter}}(P_{\text{valid}})$

Algorithm 3 provides a conceptual view of the packet processing pipeline rather than a literal API implementation, highlighting how state tracking, DPI, and mitigation actions are combined on the DPU datapath.

4.1.1 DDoS Detection without and with Offload

Figure 4.1 shows how DDoS attack detection with hardware offload functionality of DPU and without a hardware offload. In Figure 4.1(a), a packet is received from the external source router at point 1 (NIC), then forwarded to the CPU. At point 3, the RegEx operation is performed, followed by sending the result back to the CPU at point 4. At point 5, the GPU is involved in parallel processing of the RegEx operation, and then the packet is forwarded to the second NIC toward the router. In Figure 4.1(b), the packet is received from the external source router at point 1 (DPU) and then forwarded to the ARM for the RegEx operation. The processed packet is then sent to the router, resulting in a significant reduction in edge load.

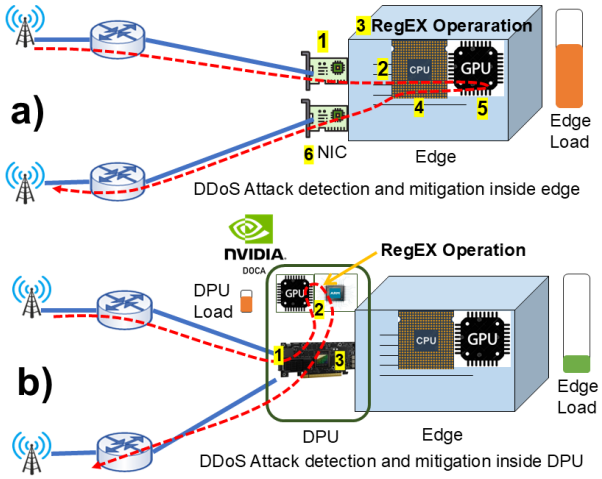


Figure 4.1: (a) Without offload: DDoS attack detection and mitigation inside edge and (b) With offload: DDoS attack detection and mitigation inside DPU

4.1.2 Stateful Connection Tracking and DPI Process

We present an enhanced rule-based stateful connection tracking system with hardware offloading to prevent DDoS attacks by efficiently managing and tracking each flow, ensuring successful TCP handshakes, and leveraging the processing power of a DPU. Our DOCA(Data Center-on-a-Chip Architecture) IPS API with custom rule integrates rate-limiting and connection tracking for real-time attack mitigation, such as SYN flood and other volumetric DDoS attacks. Below is an example of the IPS custom rule for connection tracking with hardware offload.

```
alert tcp any any -> $HOME_NET 80
```

```
(msg:"TCP Handshake Detection - Stateful";
flags:S; sid:1001; rev:1; track_by_src;
state_new, established; con_tracking_hwd;
flow_timeout 300; rate_limit 1000;
new_action drop; timeout 30;)
```

When a SYN packet is received, the SmartNIC/DPU begins tracking the connection according to the defined rule. This rule, implemented through the $f_{QoS}(p_i)$ function, incorporates both rate-limiting and connection thresholds to prevent DDoS attacks. If a TCP handshake (SYN, SYN-ACK, ACK) is completed, the connection is marked as established, and the state is managed by the $f_{track}(p_i)$ function, which tracks the flow for each packet and updates connection states accordingly. The DPU continues to track the connection through the hardware meter, ensuring the flow is monitored at line rate.

Rate-limiting is ensured by the $f_{QoS}(p_i)$ function, which slows the rate of SYN packets when the SYN packet rate exceeds the threshold (e.g., `rate_limit 1000`), thus preventing SYN floods attack.

The DPU offloads the tasks of connection tracking and flow management using the $f_{track}(p_i)$ and $f_{meter}(f_i)$ functions from the DOCA API Flow Connection Tracking (AF_CT) module. Hardware acceleration enables stateful tracking and flow management at line rate. Packets are classified using the 5-tuple (source IP, destination IP, source port, destination port, and protocol) through the $f_{track}(p_i)$ function. The packet is checked in the flow table, and it is forwarded if a match is found. Otherwise, it is sent to the missing pipe for further inspection. The $f_{track}(p_i)$ function also manages the connection states by updating the status of each flow (NEW, ESTABLISHED, CLOSED) as packets arrive. The DPU tracks connection directions to maintain session awareness and handle packets bidirectionally (incoming and outgoing traffic).

To ensure efficient flow management, the DPU performs connection aging through the $f_{meter}(f_i)$ function. Each connection in the table is periodically checked for inactivity. If a connection remains idle beyond a predefined threshold (e.g., 300 seconds),

it is removed from the table, freeing up resources for new connections. This process of aging, controlled by the $f_{track}(p_i)$ and $f_{meter}(f_i)$ functions, prevents the table from becoming overloaded with stale flow information. With multiple $f^T_{track}(p_i)$ workers running in parallel, the DPU can scale to handle a high number of connections per second (CPS) and maintain high throughput during DPI processing.

Operating in autonomous mode, the DPU manages the entire connection lifecycle from creation to deletion, using the $f_{track}(p_i)$ and $f_{meter}(f_i)$ functions without host CPU intervention. This offloads the CPU and ensures the DPU handles most tasks autonomously, optimizing performance.

By combining rate-limiting through $f_{QoS}(p_i)$ and connection tracking via $f_{track}(p_i)$, the system mitigates SYN flood attacks by slowing down SYN packets when necessary. Functions $f_{sig}(p_i)$ and $f_{anomaly}(p_i)$ help to detect and prevent spoofing by blocking packets from invalid sources or uninitiated connections.

4.1.3 Multi-Processing and Multithreading DPI with DOCA QoS Engine

The Enqueue DPI (DocaDpiEnqueue) core function is used to enqueue packets with thread-safe operations per queue. Multiple processes/threads can utilize enqueue when flows are linked to different queues. However, they cannot use enqueue simultaneously when flows are linked to the same queue. To address this limitation, integrating the DOCA QoS engine ensures efficient queue management and prioritization.

The DOCA QoS engine enables advanced traffic classification and resource allocation by assigning flows to appropriate DPDK queues based on traffic type and priority. These queues are then linked to the stateful connection tracking (CT) engine and DPI processing. This hierarchical queue management reduces contention and ensures seamless multithreading. Packets entering the QoS engine are classified, balanced, and forwarded to DPI multiqueues for deep inspection.

The Matching DPI signature (DocaDpiDequeue and DocaDpiFlowMatchGet) matches and dequeue packets. The same DPI signature can be utilized by multiple

processes/threads through two options:

- Using DocaDpiFlowMatchGet instead of DocaDpiDequeue, while still needing dequeuing to free up the DPI queue.
- Assigning a different uint16_dpi_q value for each process/thread to ensure packet processing isolation.

Packet processing isolation is further enhanced by integrating the DOCA QoS engine by dynamically allocating uint16_dpi_q values based on QoS policies. This ensures that high-priority traffic is processed with minimal latency while low-priority or malicious traffic is throttled.

This design demonstrates that queue-aware scheduling and isolation are essential to sustain DPI throughput under concurrent attack and benign traffic conditions.

4.1.4 DOCA-based Hardware offload IPS Architecture

We employed a stateful tracking module and multithreading DPI. In DPU with DOCA, multithreading DPI integration with hardware offload mechanisms ensures efficient and high-speed packet processing. These DPUs have dedicated processing units optimized for packet inspection and analysis, allowing for parallelized and accelerated DPI operations. Our proposed hardware offload DPI packet processing is described in Figure 4.2. When an incoming packet arrives at the DPU interface, it undergoes deep inspection to extract key attributes such as source and destination addresses and ports, protocol, and payload content. After identifying the flow and ensuring it meets specific criteria, such as not exceeding predefined rate thresholds, packets are enqueued for DPI processing. The result is dequeued for further analysis upon successful processing by the DPI engine. If a match is found during DPI processing, it counts match statistics related to the detected threat. The packet is dropped if the match corresponds to a known DDoS signature. Otherwise, the flow is hardware offloaded to the physical function. Once the packet processing is completed, the flow is terminated.

The DPI engine compares the attributes of the extracted packets in predefined signatures based on rules that represent known threats and attack patterns. These

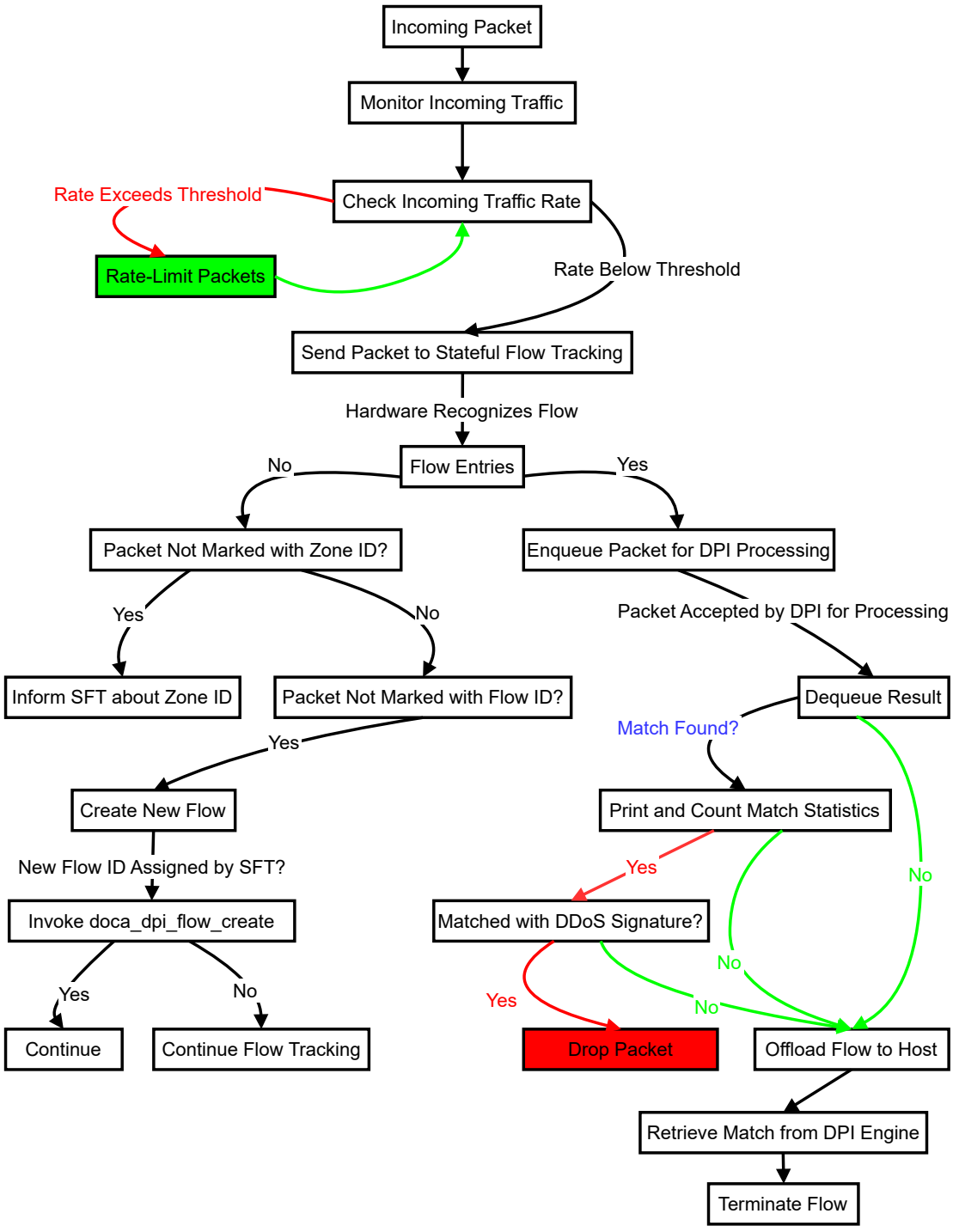


Figure 4.2: DOCA-based DPI Packet Processing Flow with Stateful Flow Tracking

rules are generated based on common DDoS attack patterns and can detect other malicious activities. DPI maintains stateful flow tracking information to contextualize packet inspection results. By associating packets with specific network connections and flows, DPI can detect anomalous behavior and identify patterns that indicate threats. Upon detecting a match between packet attributes and a signature in the database, DPI can trigger drop actions to mitigate the threat. These actions include packet drop, rate limitation, and redirection to another port, depending on the severity and nature of the detected threat. DPI operations are also integrated into the broader DOCA framework, allowing for centralized management and orchestration of network security policies. DOCA provides a unified interface for configuring DPI settings, monitoring traffic patterns, and responding to security events in real-time. DPI signature creation for our IPS involves manual analysis, threat intelligence gathering, and automated tooling provided by DOCA API. While the DOCA framework provides the underlying DPI and flow-tracking primitives, this work focuses on their integration, scheduling, and configuration to enable scalable DDoS mitigation at wire speed. Our implementation testbed will be discussed in the next section.

4.1.5 Testbed

The DDoS detection and mitigation testbed consists of two Dell R760 servers connected using a SONiC-based [163] white box network switch shown in Figure 4.3. Each server has a NVIDIA DPU. One server runs the DDoS detection system, which uses a DOCA IDS mode to classify traffic as normal or anomalous. The other server runs the DDoS mitigation system, which uses an IPS to block malicious traffic. The two servers are connected to a local data center using VLAN 2. The DPU admin is connected to the testbed using the management (Mgmt) console. The testbed is used to simulate a DDoS attack and evaluate the performance of the hardware-accelerated IPS framework. Evaluate the performance of our framework through extensive testing on wire speed in real-time traffic and using the malicious traffic

generated tools HPING3[1], LOIC[2], HOIC[3], and TRex (Cisco TRex traffic generator)[4].

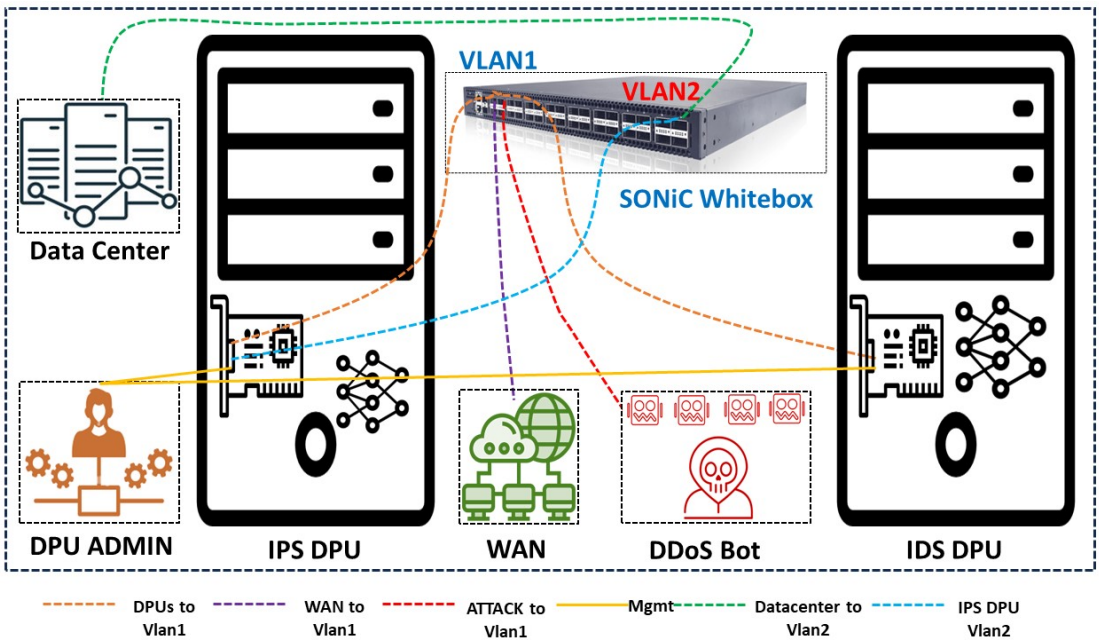


Figure 4.3: Evaluation Testbed For DDoS Attacks Detection

The testbed is designed to evaluate throughput, latency, and scalability under realistic attack scenarios, enabling direct comparison between CPU-only and DPU-offloaded deployments.

4.2 Results Analysis

Figure 4.6 evaluates the efficiency of connection tracking under varying rules counts, comparing hardware offload with CPU-only scenarios. Concurrent connections decrease rapidly Without offload, starting at 50K for 10 rules and dropping to below 10K as the rules increase to 100. The hardware offload maintains significantly higher efficiency, with 500K concurrent connections for 10 rules and a gradual decline to 200K for 100 rules. This improvement highlights the scalability and robustness of DPU-based connection tracking in complex networks.

Figure 4.7 compares the time required to process 500K connections as the number of rules increases. With hardware offload, the processing time increases at a decreas-

[1]https://www.hping.org/
[2]https://sourceforge.net/projects/loic/
[3]https://sourceforge.net/projects/hoic/
[4]https://trex-tgn.cisco.com/

ing rate, starting at 3 seconds and reaching 15 seconds for 50 rules. The CPU-only scenario shows a linear increase, starting at 20 seconds and reaching nearly 170 seconds for the same number of rules. This difference highlights the effectiveness of hardware offload in minimizing processing delays.

Figure 4.8 shows latency under different attack loads, ranging from 10^5 to 10^8 pps. With hardware offload, latency remains low and stable, increasing minimally from 1 ms to approximately 101 ms under extreme attack loads. On the other hand, without offloading, latency increases significantly from 10 ms to the timeout, underscoring the shortcomings of CPU-only processing during extreme attacks. It shows that hardware offloading effectively maintains low latency performance in the face of network assaults.

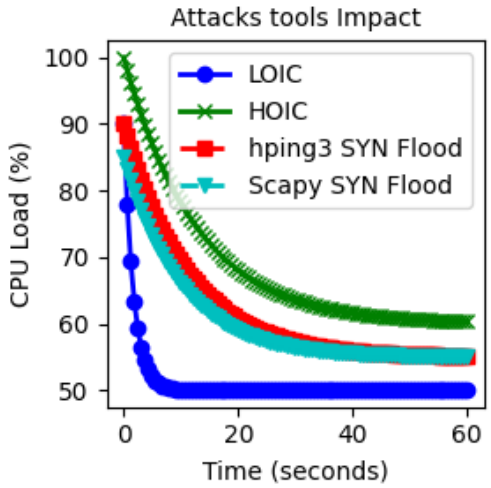


Figure 4.4: Impact of Attack Tools on CPU Load

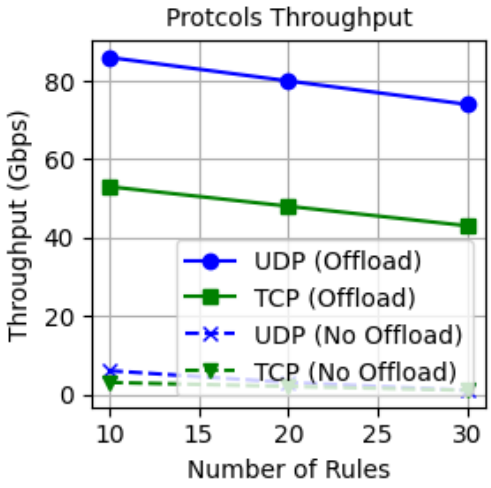


Figure 4.5: UDP and TCP Throughputs

These results directly address Research Question RQ2 by demonstrating that DPU-based offload enables deterministic, low-latency mitigation under extreme DDoS conditions.

4.2.1 Hardware-Software Synergy: Trade-offs and Bottlenecks

Offloading tasks to a DPU can reduce host CPU utilization, such as offloading deep packet inspection, which saves CPU cycles. However, latency in DPU-host data transfers can be constrained by the PCIe bottleneck, reducing overall throughput.

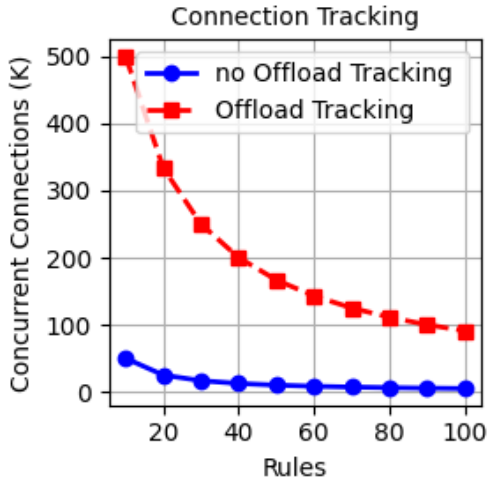


Figure 4.6: Connection tracking performance with and without hardware offload

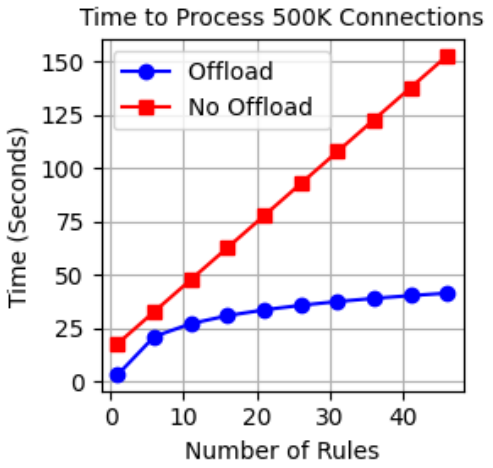


Figure 4.7: Processing time for 500K connections with varying rules

Some functions, such as dynamic flow-state management, depend on the host's memory management. Fully offloading these to the DPU may not be feasible, as it may lack effective mechanisms for managing dynamic updates. These trade-offs highlight the need for a balanced approach to optimize system performance. These trade-offs motivate the broader evaluation of SmartNIC resource allocation and architectural limits discussed in Chapters 6 and 7.

4.3 Chapter Summary

This chapter investigated how programmable DPUs can be used as an effective and practical platform for mitigating large-scale DDoS attacks in high-speed networks. The motivation stems from the limitations of traditional software-based defenses, which rely heavily on host CPUs and external appliances. Under high attack volumes, these approaches suffer from increased latency, limited scalability, and excessive resource consumption, often leading to service degradation or failure. The core idea explored in this chapter is to move detection and mitigation as close as possible to the data path by executing them directly inside the DPU.

The proposed solution introduced a hardware-accelerated intrusion prevention framework built on NVIDIA BlueField-2. By integrating DPDK, DOCA libraries, regular-expression matching, and Suricata-based inspection, the framework performs

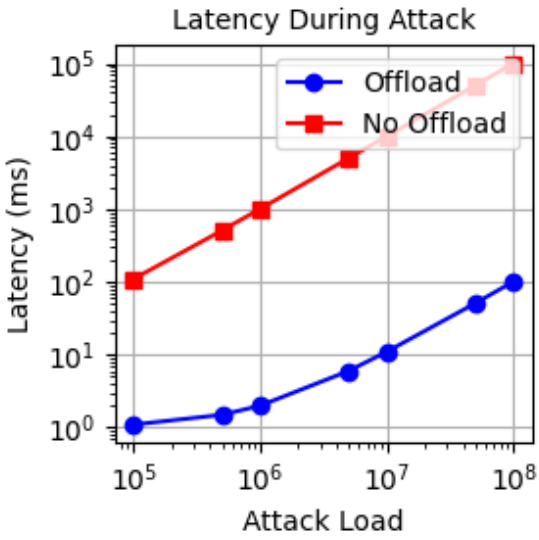


Figure 4.8: Latency during the attack with and without offload

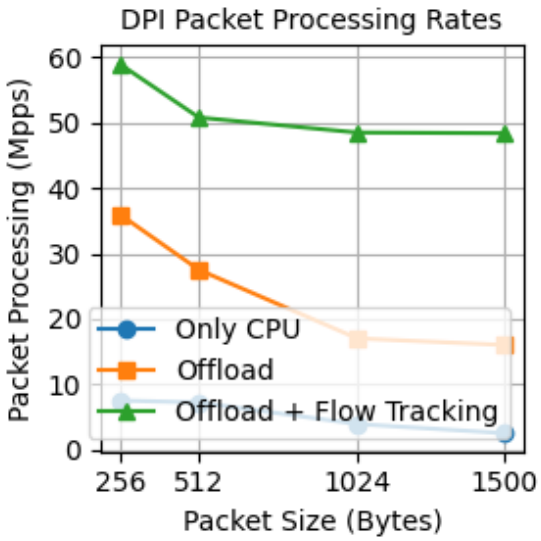


Figure 4.9: The packet processing rates for varying packet sizes

deep packet inspection and attack mitigation at line rate. Offloading DPI and stateful processing to the DPU reduces dependency on the host CPU and removes the need for additional middleboxes, making the design both scalable and cost-efficient for modern data centers and edge deployments.

A key contribution of this chapter is the detailed design of a stateful, rule-based connection tracking and DPI pipeline that operates entirely within the DPU. Using DOCA Flow connection tracking and hardware meters, the system maintains per-flow state, validates TCP handshakes, enforces rate limits, and identifies abnormal traffic patterns such as SYN floods. Connection aging and flow cleanup are handled autonomously by the DPU, preventing flow-table exhaustion and ensuring sustained operation under attack. The combination of rate-limiting, state tracking, and signature-based detection enables early filtering of malicious traffic before it reaches host resources.

The chapter also addressed scalability through multi-processing and multithreading. By integrating the DOCA QoS engine, traffic is classified and distributed across multiple DPI queues, reducing contention and enabling parallel inspection. This design allows high-priority or benign traffic to be processed with minimal latency, while suspicious or low-priority flows are throttled or dropped. The architecture demonstrates how careful queue management and hardware-assisted scheduling are

essential for sustaining performance in multi-tenant and high-throughput environments.

Experimental evaluation on a realistic testbed showed clear advantages of DPU-based offload compared to CPU-only processing. Hardware offload sustained a much higher number of concurrent connections as rule complexity increased, significantly reduced processing time for large connection sets, and maintained low and stable latency even under extreme packet rates. In contrast, CPU-based approaches degraded rapidly, exhibiting sharp drops in throughput and latency spikes under heavy attack loads. These results confirm that DPU offload is not only an optimization, but a necessity for defending against modern volumetric DDoS attacks.

Finally, the chapter discussed the practical trade-offs of hardware offload. While DPUs greatly reduce CPU load and enable line-rate mitigation, certain limitations remain, such as PCIe transfer overheads and the challenges of fully offloading dynamic state management. These observations highlight the importance of hardware-software co-design, where responsibilities are carefully split between the host and the DPU to achieve optimal performance.

Overall, this chapter demonstrated that programmable DPUs provide a strong foundation for next-generation DDoS mitigation. By combining stateful flow tracking, deep packet inspection, and hardware acceleration, the proposed framework achieves robust, scalable, and low-latency defense against DDoS attacks, while preserving host resources and maintaining service availability in high-speed networks.

Chapter 5

DPUAUT: Secure Authentication Protocol for Intelligent Swarm Systems

Intelligent Swarm Systems (ISS) comprise large numbers of autonomous devices that cooperate to sense, compute, and act in dynamic environments. Unlike cloud-centric systems, swarm deployments are latency- and scale-sensitive: devices may join or leave frequently, communicate over heterogeneous and intermittently connected wireless links, and require near-real-time responses for safe and coordinated operation. In such settings, authentication is a critical control-plane primitive that governs admission, trust establishment, and secure data exchange.

Conventional authentication and key-management solutions are often designed around remote cloud servers, where identities and long-term secrets are stored and verified centrally. This architecture introduces practical limitations for swarms. First, end-to-end authentication delay increases because of long round-trip times, congestion, or intermittent connectivity. Second, centralized verification becomes a scalability bottleneck as the swarm grows. Third, cloud-centric authentication expands the attack surface when protocol messages traverse public networks, enabling eavesdropping, message modification, and replay. Even when authentication services are moved to the edge, security remains non-trivial: many schemes store long-term cryptographic keys in device memory, making physical capture a serious

threat. Once secrets are extracted from a captured device, adversaries can impersonate legitimate swarm nodes, inject forged data, or disrupt coordination.

To address these limitations, this chapter proposes *DPUAUT*, a secure and lightweight authentication protocol tailored to ISS. The central design objective is to achieve low-latency mutual authentication while resisting both network-level and physical attacks. DPUAUT anchors device trust in a PUF-based identity primitive, avoiding the storage of static long-term secrets that are vulnerable to physical compromise. In addition, DPUAUT accelerates the verifier-side workflow by offloading authentication verification to a Smart Network Interface Card (SmartNiC)-based DPU, specifically targeting NVIDIA BlueField-class hardware. This architectural choice reduces end-to-end authentication delay, improves scalability for large swarms, and strengthens isolation by executing security-critical operations inside a dedicated, hardware-assisted subsystem.

Contributions. The main contributions of this chapter are:

- Design of *DPUAUT*, a PUF-anchored authentication and key-agreement protocol for ISS that supports mutual authentication, device anonymity, and session key establishment under realistic adversarial conditions.
- Design of a SmartNiC/DPU-assisted verification architecture that offloads verifier-side authentication logic to BlueField hardware to reduce latency and improve scalability.
- Formalization of the system and threat models, definition of notation, and specification of the protocol in structured phases (initialization, device registration, Autonomous Swarm Devices Computing Server (ASDCS) registration, and authentication).
- Security evaluation using automated verification (AVISPA backends), a proof sketch in the Real-or-Random (RoR) model, and an informal analysis against practical threats.

- Quantitative analysis of computation and communication overheads, and comparison of DPUAUT with representative state-of-the-art authentication schemes for similar deployments.

The remainder of this chapter is structured as follows. Section 5.1 introduces the system model, preliminaries, and threat assumptions. Section 5.2 presents the proposed DPUAUT protocol phases and message flows. Section 5.3 provides security validation and analysis. Finally, reports performance and overhead results, followed by a chapter summary.

5.1 System Model and Preliminaries

This section defines the system entities, trust assumptions, cryptographic primitives, and notation used throughout the proposed protocol. We also present the adversarial capabilities considered in the security analysis.

5.1.1 System Entities and Trust Assumptions

We consider an ISS composed of the following entities (Figures 5.1–5.2):

- **Autonomous Swarm Device ($\mathcal{ASD}_i$):** a resource-constrained swarm node equipped with sensing and wireless communication capabilities. Each device embeds a PUF instance used to derive device-bound responses from challenges.
- **Autonomous Swarm Device Computing Server ($\mathcal{ASDCS}_j$):** an edge-side computing server deployed near the swarm operation area. It supports low-latency services for swarm nodes and participates in the authentication workflow.
- **Private Cloud Server ($\mathcal{PCS}$):** a trusted authority responsible for bootstrapping system parameters, registering entities, and assisting in credential issuance and verification.

We assume that $\mathcal{PCS}$ is trustworthy and maintains long-term master secrets in a protected environment. We assume that wireless/public channels among $\mathcal{ASD}_i$

and $\mathcal{ASDCS}_j$ are insecure and may be fully controlled by an attacker. Registration steps that require secure enrollment (e.g., PUF challenge–response association) are assumed to occur over a protected channel or in a controlled environment.

5.1.2 Physical Unclonable Function (PUF)

A PUF is a hardware primitive that exploits manufacturing variability to produce instance-specific responses. Given a challenge $\mathcal{CH}_i \in \{0,1\}^k$, a device-specific PUF instance $\mathcal{P}_i(\cdot)$ generates a response $\mathcal{R}_i \in \{0,1\}^\lambda$:

$$\mathcal{R}_i \leftarrow \mathcal{P}_i(\mathcal{CH}_i). \tag{5.1}$$

A key security property is that two distinct devices produce sufficiently different responses for the same challenge. Let $\mathcal{H}_\mathcal{D}(\cdot,\cdot)$ denote the Hamming distance. For two devices $i_1 \neq i_2$, the responses should satisfy:

$$\mathcal{H}_\mathcal{D}\Big(\mathcal{P}_{i_1}(\mathcal{CH}), \mathcal{P}_{i_2}(\mathcal{CH})\Big) \geq b, \tag{5.2}$$

for a system threshold b, ensuring inter-device uniqueness. In swarm deployments, PUFs are attractive because they reduce reliance on stored long-term secrets and increase robustness against physical extraction attacks. However, PUF deployments must also consider resistance to invasive and non-invasive physical attacks, including side-channel leakage and machine-learning-based modeling attempts [164].

5.1.3 SmartNIC/DPU-Assisted Verification

To reduce authentication latency and improve scalability, DPUAUT offloads verifier-side logic to a SmartNiC-based DPU. The DPU provides an isolated execution environment, enabling:

- **Lower end-to-end delay:** verifier operations execute closer to the data path and avoid overload on general-purpose host CPUs.
- **Higher concurrency:** authentication requests from multiple swarm devices

can be handled in parallel under constrained edge resources.

- **Improved isolation:** sensitive verification steps execute inside the DPU subsystem rather than in a shared host environment.

Accordingly, the protocol description in subsequent sections assumes that $\mathcal{ASDCS}_j$ can leverage DPU acceleration for cryptographic verification and message processing.

PUF deployment considerations. In this work, the PUF is integrated on the SmartNIC platform and follows implementation guidelines for TrustZone-aligned designs [165]. The design goals are: (i) resistance to physical and modeling attacks, (ii) practical implementation feasibility on BlueField-class SmartNICs, and (iii) latency reduction by enabling local, verifier-side acceleration. In particular, the verifier-side PUF validation and associated hash-based computations are placed in the SmartNIC/DPU execution domain to improve isolation and reduce control-plane contention on the host CPU.

Table 5.1 provides a simplified illustration of the challenge–response mechanism used by the enrollment process.

Table 5.1: Challenge–response example for SmartNiC-based PUF authentication

Challenge	**Response**
$\mathcal{CH}_1$	$\mathcal{R}_1 = \mathcal{P}(\mathcal{CH}_1)$
$\mathcal{CH}_2$	$\mathcal{R}_2 = \mathcal{P}(\mathcal{CH}_2)$
$\mathcal{CH}_3$	$\mathcal{R}_3 = \mathcal{P}(\mathcal{CH}_3)$
$\vdots$	$\vdots$

where $\mathcal{CH}_i$ is a challenge generated by $\mathcal{PCS}$, $\mathcal{P}$ is the PUF function, and $\mathcal{R}_i$ is the response produced by the device-specific PUF instance.

5.1.4 Notation

Throughout the thesis, a specific notation represents the entities, algorithms, and cryptographic primitives used in the authentication protocol; these are summarized in Table 5.2.

Table 5.2: Notations

Notation	Description
$\mathcal{ASDCS}$	Autonomous Swarm Devices Computing Server
$\mathcal{MSK}$	Master secret key of $\mathcal{PCS}$
$\mathcal{PCS}$	Private Cloud Server (trusted authority)
SID_j	Pseudonym of $\mathcal{ASDCS}_j$ (server-side identity token)
$\mathcal{K}_j$	Secret key associated with SID_j
$\mathcal{ASD}_i$	ith Autonomous Swarm Device
ID_i	Long-term identity of $\mathcal{ASD}_i$ (enrollment-time)
σ_i	Digital signature / verifier tag for $\mathcal{ASD}_i$ (as defined by the scheme)
TID_i	Temporary identity of $\mathcal{ASD}_i$
$\mathcal{CH}_i, \mathcal{R}_i$	Challenge–response pair of $\mathcal{ASD}_i$
$\mathcal{P}_i$	PUF instance for device i
$\mathcal{A}$	Adversary
$\mathcal{TR}_i$	Tamper-resilient storage (device-side)
$\mathcal{ASD}\int\|\|$	Authenticated swarm device
G_0, G_1, G_2, G_3	Security games (used in the proof sketch)

5.1.5 System Setup

We consider an ISS that consists of $\mathcal{ASD}$ devices, a Private Cloud Server (PCS), and an ASDCS as depicted in Fig. 5.1. Figure 5.1 and Figure 5.2 illustrate the architecture and design of the authentication-based ISS. Each $\mathcal{ASD}_i$ is equipped with a sensor node that collects data. The ISS design builds upon our previous work [166].

In the proposed swarm authentication system, a trustworthy $\mathcal{PCS}$ performs enrollment and registration and provides the necessary information for authentication. An $\mathcal{ASDCS}$ positioned near the data source assists $\mathcal{ASD}$ nodes by providing low-latency access to compute and coordination services in the swarm area. This approach offloads computation tasks, reduces backhaul bandwidth consumption, and alleviates pressure on centralized authentication services.

During the authentication process, mutual authentication occurs between swarm devices and $\mathcal{ASDCS}$, establishing a session key for secure communication over a public channel. We focus on the common scenario where a new device attempts to join the swarm. In this context, the authentication procedure must prioritize latency while maintaining strong resistance to impersonation and physical compromise. After successful authentication, the established session key is used to protect

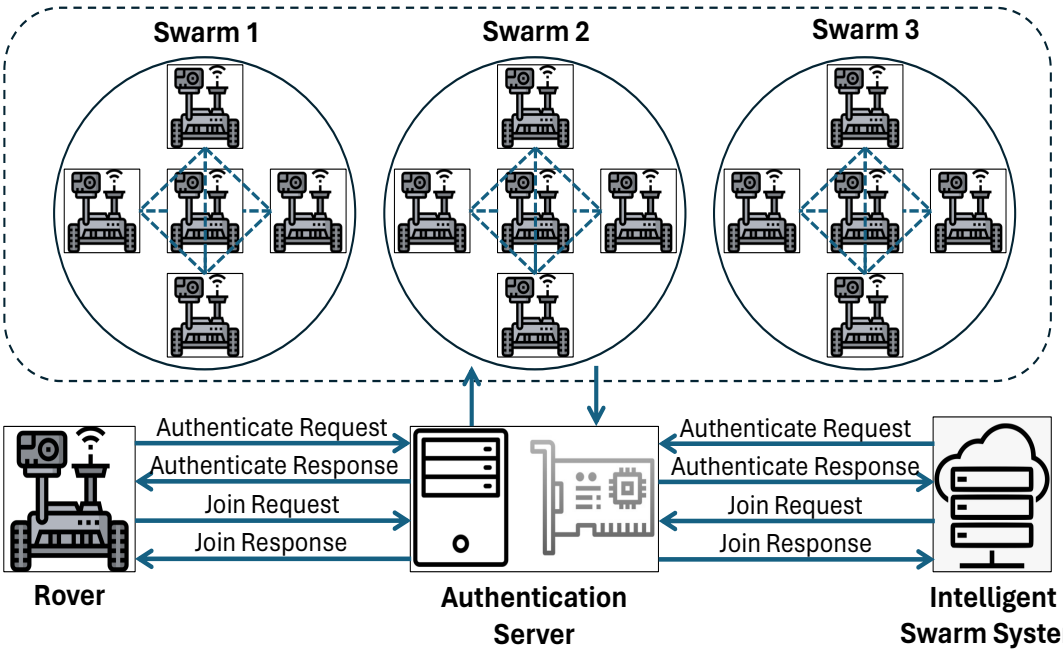


Figure 5.1: Autonomous Swarm Intelligence System

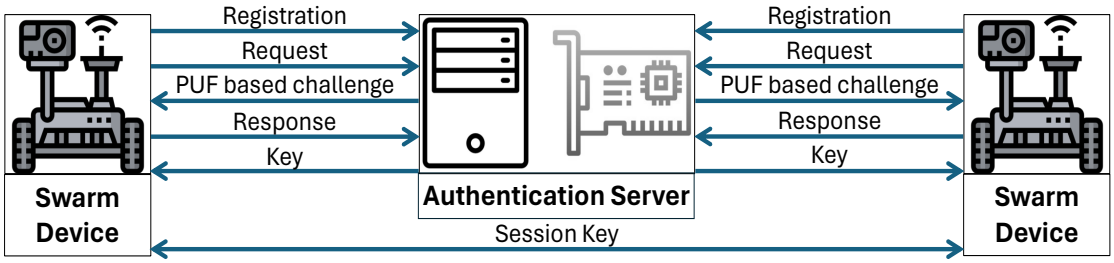


Figure 5.2: SmartNIC-based PUF authentication protocol

subsequent swarm communications.

5.1.6 Acceleration of Authentication Protocol

To improve performance and efficiency, we propose offloading verifier-side authentication operations from a general-purpose CPU to a SmartNiC/DPU. This design targets three objectives:

- **Latency:** reduce the time required to authenticate swarm devices before initiating protected service access or data transfer.
- **Scalability:** support authentication for large swarms (up to tens of thousands of devices) under SmartNIC memory and processing constraints.
- **Security and isolation:** execute verification logic in a dedicated hardware-

assisted execution domain, reducing exposure to host-level compromise and limiting the attack surface of the verifier.

A key practical constraint is that SmartNIC/DPU environments may restrict available primitives and execution models compared to fully featured CPUs. Therefore, DPUAUT emphasizes lightweight, hash-based computations and careful state management to remain feasible under SmartNIC constraints while still providing robust security guarantees.

5.1.7 A One-Way Hash Function

DPUAUT employs a one-way hash function $h(\cdot)$ that maps an input of arbitrary length to a fixed-length digest. The function is assumed to satisfy standard security properties:

- **Preimage resistance:** given y, it is computationally infeasible to find x such that $h(x) = y$.
- **Second-preimage resistance:** given x, it is computationally infeasible to find $x' \neq x$ such that $h(x') = h(x)$.
- **Collision resistance:** it is computationally infeasible to find distinct $x_1 \neq x_2$ such that $h(x_1) = h(x_2)$.

These properties enable compact message authentication tags, integrity verification, and key derivation throughout the protocol.

5.1.8 Threat Model

Following prior authentication literature, we adopt the Canetti–Krawczyk adversary model [167] to assess the security of the proposed scheme for ASDs. In particular, the attacker $\mathcal{A}$ is assumed to have the following capabilities:

- **Full control of the public channel:** $\mathcal{A}$ can eavesdrop, replay, delay, drop, and modify any protocol message exchanged over the public channel between $\mathcal{ASD}_i$ and $\mathcal{ASDCS}_j$.

- **Man-in-the-middle behavior:** $\mathcal{A}$ can mount Man-in-the-Middle (MiTM) attacks by altering, injecting, or reordering authentication messages to attempt impersonation or session disruption.
- **Device capture and memory extraction:** $\mathcal{A}$ may physically capture an $\mathcal{ASD}_i$ and apply explicit extraction techniques to read non-tamper-resilient memory contents.
- **Public identifiers are observable:** system-wide identifiers required for interoperability (e.g., server pseudonyms broadcast to registered devices) are assumed to be observable by $\mathcal{A}$.

We assume that the $\mathcal{PCS}$ is trustworthy and not compromised. Enrollment/registration steps that associate PUF challenge–response pairs with a device identity are assumed to occur over a protected channel or in a controlled environment, consistent with standard PUF enrollment assumptions.

5.2 Proposed DPUAUT Protocol

This section presents the proposed *DPUAUT* scheme (Secure Authentication Protocol with SmartNIC Integration for Trustworthy Communications in ISS). DPUAUT is designed to provide low-latency mutual authentication and session key establishment between a swarm device and an edge-side computing server, while leveraging PUF-anchored identity primitives and SmartNIC/DPU-assisted verifier acceleration. The protocol uses the notation summarized in Table 5.2 and consists of four phases: (i) ISS initialization, (ii) device registration, (iii) ASDCS registration, and (iv) online authentication.

5.2.1 ISS Initialization

During initialization, the trusted authority $\mathcal{PCS}$ selects global system parameters. Specifically, $\mathcal{PCS}$ samples a master secret key $\mathcal{MSK} \in \mathbb{Z}_p^*$ uniformly at random, where p is a large prime. The value $\mathcal{MSK}$ is kept secret and never disclosed. The

authority also selects a cryptographic one-way hash function $h(\cdot)$ and publishes its description as a system parameter. All entities use $h(\cdot)$ for message authentication tags, key derivation, and integrity checks, while $\mathcal{PCS}$ securely stores $\mathcal{MSK}$.

5.2.2 Device Registration Phase

The device registration phase enrolls each swarm device $\mathcal{ASD}_i$ with the trusted authority $\mathcal{PCS}$, binds the device to a PUF challenge–response record, and provisions device-side credentials for later online authentication. This phase is performed over a protected enrollment channel (cf. Section 5.1).

1. **Registration request.** The device $\mathcal{ASD}_i$ sends its identity ID_i to $\mathcal{PCS}$ as a registration request. The authority checks whether ID_i is already registered. If ID_i exists, $\mathcal{PCS}$ rejects the request and asks the device to choose a new identity; otherwise, it accepts ID_i.

2. **Challenge issuance.** $\mathcal{PCS}$ generates a fresh challenge $\mathcal{CH}_i$ and securely transmits it to $\mathcal{ASD}_i$.

3. **PUF response generation.** Upon receiving $\mathcal{CH}_i$, the device evaluates its PUF instance to obtain

$$\mathcal{R}_i \leftarrow \mathcal{P}_i(\mathcal{CH}_i). \tag{5.3}$$

   The device sends $\mathcal{R}_i$ back to $\mathcal{PCS}$.

4. **Enrollment verification.** $\mathcal{PCS}$ verifies the device enrollment record and the associated authenticator σ_i as defined by the scheme (e.g., a signature/-tag covering the registration transcript). If verification fails, the request is rejected.

5. **Credential provisioning.** If verification succeeds, $\mathcal{PCS}$ assigns a temporary identity TID_i to $\mathcal{ASD}_i$ and derives a device-specific secret

$$\mathcal{K}_i = h(ID_i \| \mathcal{MSK}). \tag{5.4}$$

$\mathcal{PCS}$ stores $\{ID_i, \mathcal{CH}_i, \mathcal{R}_i, \sigma_i\}$ indexed by TID_i in its protected repository. The authority then encrypts and transmits the credential pair $\langle TID_i, \mathcal{K}_i \rangle$ to $\mathcal{ASD}_i$ over the protected channel.

6. **Secure storage.** The device stores $\langle TID_i, \mathcal{K}_i \rangle$ in its tamper-resilient storage $\mathcal{TR}_i$ (or Trusted Personal Device (TPD)) for use during the online authentication phase.

Figure 5.3 illustrates the device registration message flow between $\mathcal{ASD}_i$ and $\mathcal{PCS}$.

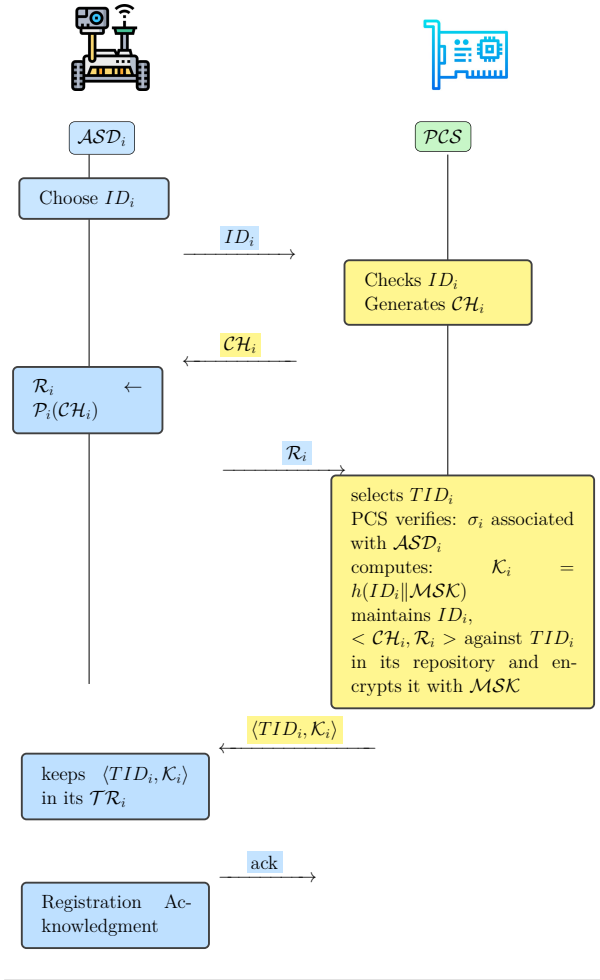


Figure 5.3: $\mathcal{ASD}_i$ registration phase with $\mathcal{PCS}$

5.2.3 Swarm Devices Registration Phase

The $\mathcal{ASDCS}_j$ registers with $\mathcal{PCS}$ to obtain server-side credentials used during online authentication. This phase is executed offline and incorporates a digital signature

mechanism to ensure authenticity, integrity, and non-repudiation. Figure 5.4 illustrates the registration message flow.

1. **Registration request.** $\mathcal{ASDCS}_j$ sends a registration request to $\mathcal{PCS}$ to join the system.

2. **Identity assignment and challenge.** $\mathcal{PCS}$ assigns a unique identity ID_j and a pseudo-identity SID_j to $\mathcal{ASDCS}_j$, generates a fresh challenge C_j, and sends C_j to $\mathcal{ASDCS}_j$ for processing.

3. **Signature creation.** $\mathcal{PCS}$ computes a digital signature σ_j over the registration transcript (e.g., $ID_j \| SID_j \| C_j$). Let $Sign_{SK}(\cdot)$ denote the signing operation under $\mathcal{PCS}$'s signing key SK. The authority sends $\langle ID_j, SID_j, C_j, \sigma_j \rangle$ to $\mathcal{ASDCS}_j$.

4. **Signature verification.** $\mathcal{ASDCS}_j$ verifies σ_j using the corresponding verification key PK and a verification function $Verify_{PK}(\cdot)$. If verification fails, the registration is aborted.

5. **Server secret derivation.** Upon successful verification, $\mathcal{PCS}$ derives the server secret

$$\mathcal{K}_j = h(ID_j \| \mathcal{MSK}), \tag{5.5}$$

   stores the mapping between $\mathcal{K}_j$ and SID_j in its database, encrypts $\langle \mathcal{K}_j, SID_j \rangle$ under $\mathcal{MSK}$, and sends it to $\mathcal{ASDCS}_j$.

6. **Secure storage.** $\mathcal{ASDCS}_j$ securely stores $\langle \mathcal{K}_j, SID_j \rangle$ for later use during the authentication phase.

5.2.4 Authentication Phase

In this phase, the swarm device $\mathcal{ASD}_i$ and $\mathcal{ASDCS}_j$ mutually authenticate and establish a shared session key SK for subsequent secure communication. The core idea is that $\mathcal{ASD}_i$ proves freshness and possession of device credentials, while $\mathcal{ASDCS}_j$ (with assistance from $\mathcal{PCS}$) validates the device and returns values that allow both

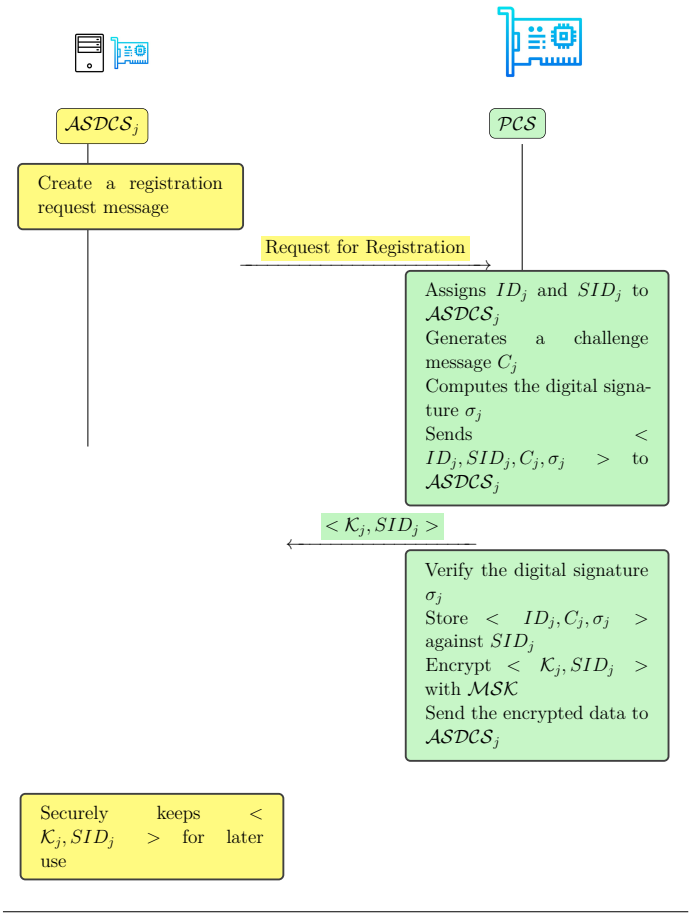


Figure 5.4: $\mathcal{ASDCS}_j$ registration phase

endpoints to derive the same SK. Only authenticated devices ($\mathcal{ASD}\int\|$) are permitted to participate in the swarm.

Figure 5.5 summarizes the complete message flow and computations.

5.2.4.1 Authentication Initiation

- **Freshness and tag computation.** $\mathcal{ASD}_i$ samples a fresh nonce r_i and computes

$$T_i = r_i \oplus h(ID_i), \qquad Verify_i = h(ID_i \| \mathcal{K}_i \| SID_j \| r_i). \tag{5.6}$$

- **Transmission to $\mathcal{ASDCS}_j$.** $\mathcal{ASD}_i$ sends $(TID_i, T_i, Verify_i)$ to $\mathcal{ASDCS}_j$ over the public channel.

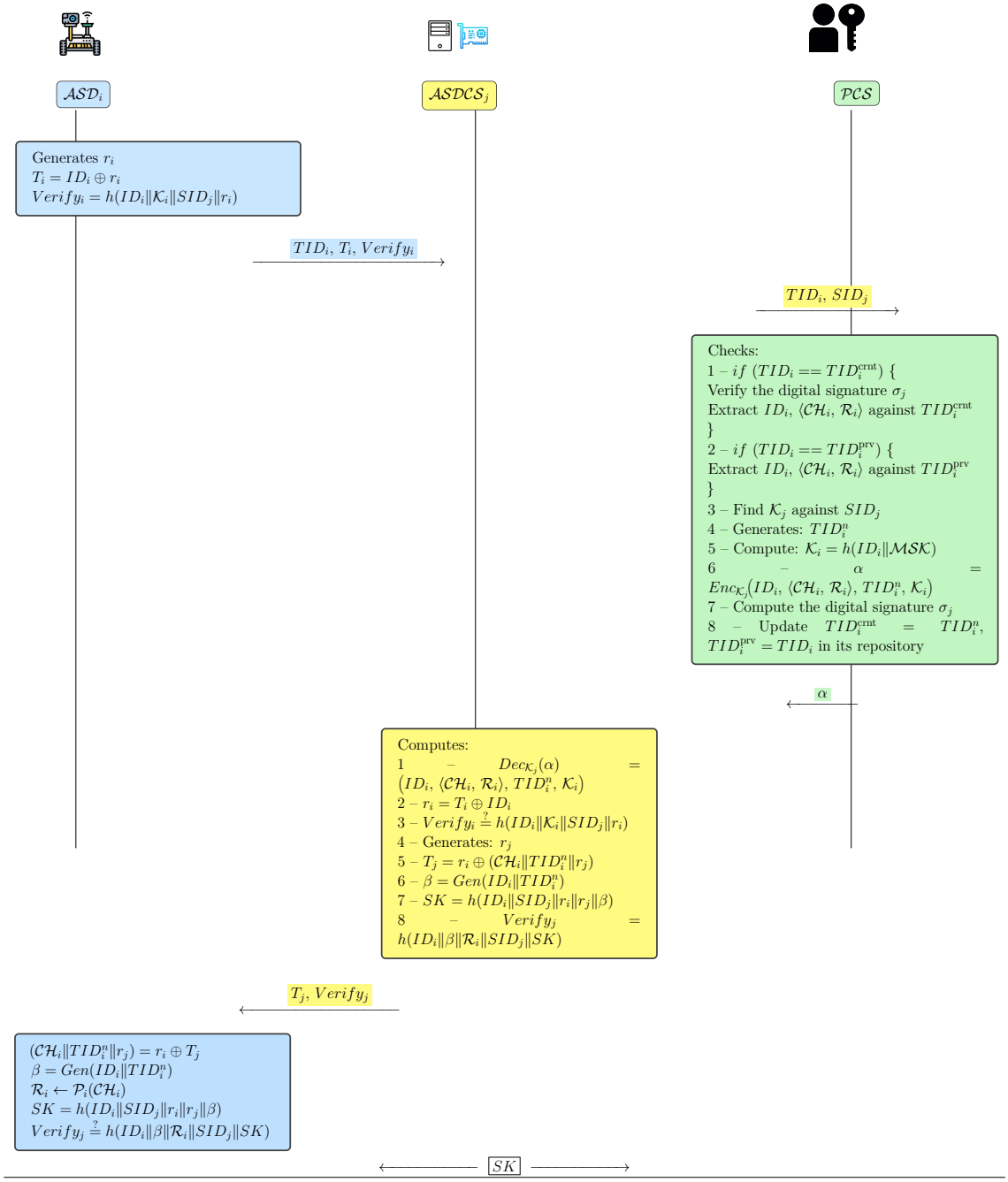


Figure 5.5: Authentication phase computations and message flow

5.2.4.2 Credential Verification (DPU-assisted at $\mathcal{ASDCS}_j$)

- **Reception.** $\mathcal{ASDCS}_j$ receives $(TID_i, T_i, Verify_i)$ and stores T_i and $Verify_i$.
- **Delegation to $\mathcal{PCS}$.** $\mathcal{ASDCS}_j$ forwards (TID_i, SID_j) to $\mathcal{PCS}$ for credential lookup and validation. In DPUAUT, this verifier-side processing is executed in the SmartNIC/DPU domain to reduce end-to-end delay and improve concurrency.

5.2.4.3 Credential Extraction and Token Generation at $\mathcal{PCS}$

- **Lookup.** $\mathcal{PCS}$ checks whether TID_i matches the current or previous temporary identity record (supporting controlled identity update).
- **Key retrieval and updates.** $\mathcal{PCS}$ retrieves $\mathcal{K}_j$ for SID_j, generates the next temporary identity TID_i^n, and (re-)derives $\mathcal{K}_i = h(ID_i \| \mathcal{MSK})$.
- **Server-to-edge token.** $\mathcal{PCS}$ constructs

$$\alpha = Enc_{\mathcal{K}_j}\Big(ID_i, \langle \mathcal{CH}_i, \mathcal{R}_i \rangle, TID_i^n, \mathcal{K}_i\Big), \tag{5.7}$$

updates $TID_i^{crnt} \leftarrow TID_i^n$ and $TID_i^{prv} \leftarrow TID_i$ in its repository, and sends α to $\mathcal{ASDCS}_j$.

5.2.4.4 Verification and Key Derivation at $\mathcal{ASDCS}_j$ and $\mathcal{ASD}_i$

- **Edge-side checks.** $\mathcal{ASDCS}_j$ decrypts α using $\mathcal{K}_j$ to obtain $(ID_i, \langle \mathcal{CH}_i, \mathcal{R}_i \rangle, TID_i^n, \mathcal{K}_i)$. It recovers $r_i = T_i \oplus h(ID_i)$ and verifies:

$$Verify_i \stackrel{?}{=} h(ID_i \| \mathcal{K}_i \| SID_j \| r_i). \tag{5.8}$$

- **Session key material.** If valid, $\mathcal{ASDCS}_j$ samples a fresh nonce r_j and computes

$$M = (\mathcal{CH}_i \| TID_i^n \| r_j), \quad T_j = M \oplus H_{|M|}(r_i), \quad \beta = h(ID_i \| TID_i^n), \tag{5.9}$$

derives

$$SK = h(ID_i \| SID_j \| r_i \| r_j \| \beta), \tag{5.10}$$

and computes

$$Verify_j = h(ID_i \| \beta \| \mathcal{R}_i \| SID_j \| SK). \tag{5.11}$$

It sends $(T_j, Verify_j)$ to $\mathcal{ASD}_i$.

- **Device-side validation.** $\mathcal{ASD}_i$ recovers $M = T_j \oplus H_{|M|}(r_i)$, parses M as $(\mathcal{CH}_i, TID_i^n, r_j)$, recomputes $\beta = h(ID_i \| TID_i^n)$, evaluates its PUF to obtain $\mathcal{R}_i \leftarrow \mathcal{P}_i(\mathcal{CH}_i)$, derives

$$SK = h(ID_i \| SID_j \| r_i \| r_j \| \beta), \tag{5.12}$$

and verifies

$$Verify_j \stackrel{?}{=} h(ID_i \| \beta \| \mathcal{R}_i \| SID_j \| SK). \tag{5.13}$$

Algorithm 4 Mutual Authentication and Session Key Establishment in DPUAUT

Entities: $\mathcal{ASD}_i$, $\mathcal{ASDCS}_j$, $\mathcal{PCS}$
Output: shared session key SK between $\mathcal{ASD}_i$ and $\mathcal{ASDCS}_j$
Notation: $h(\cdot)$ cryptographic hash function; $Enc_K(\cdot), Dec_K(\cdot)$ authenticated encryption/decryption;
    $H_\ell(x)$ expands x to ℓ bits using a KDF; $\|$ concatenation; $\oplus$ XOR on equal-length bitstrings.
    $|M|$ denotes bit-length of M.
Step 1 (Device → Edge)
$\mathcal{ASD}_i$ samples fresh nonce r_i.
$T_i \leftarrow r_i \oplus h(ID_i)$
$Verify_i \leftarrow h(ID_i \| \mathcal{K}_i \| SID_j \| r_i)$
$\mathcal{ASD}_i \rightarrow \mathcal{ASDCS}_j : (TID_i, T_i, Verify_i)$
Step 2 (Edge → PCS)
$\mathcal{ASDCS}_j \rightarrow \mathcal{PCS} : (TID_i, SID_j)$
Step 3 (PCS processing)
$\mathcal{PCS}$ validates TID_i and retrieves $(ID_i, \mathcal{CH}_i, \mathcal{R}_i)$.
$\mathcal{PCS}$ retrieves $\mathcal{K}_j$ associated with SID_j.
$\mathcal{PCS}$ generates fresh TID_i^n and computes $\mathcal{K}_i \leftarrow h(ID_i \| \mathcal{MSK})$.
$\alpha \leftarrow Enc_{\mathcal{K}_j}(ID_i, \langle \mathcal{CH}_i, \mathcal{R}_i \rangle, TID_i^n, \mathcal{K}_i)$
$\mathcal{PCS} \rightarrow \mathcal{ASDCS}_j : \alpha$
Step 4 (Edge processing)
$(ID_i, \langle \mathcal{CH}_i, \mathcal{R}_i \rangle, TID_i^n, \mathcal{K}_i) \leftarrow Dec_{\mathcal{K}_j}(\alpha)$
$r_i \leftarrow T_i \oplus h(ID_i)$
Abort if $Verify_i \neq h(ID_i \| \mathcal{K}_i \| SID_j \| r_i)$
Sample fresh nonce r_j
$\beta \leftarrow h(ID_i \| TID_i^n)$
$SK \leftarrow h(ID_i \| SID_j \| r_i \| r_j \| \beta)$
$M \leftarrow (\mathcal{CH}_i \| TID_i^n \| r_j)$
$T_j \leftarrow M \oplus H_{|M|}(r_i)$
$Verify_j \leftarrow h(ID_i \| \beta \| \mathcal{R}_i \| SID_j \| SK)$
$\mathcal{ASDCS}_j \rightarrow \mathcal{ASD}_i : (T_j, Verify_j)$
Step 5 (Device verification)
$M \leftarrow T_j \oplus H_{|M|}(r_i)$ ▷ $M = (\mathcal{CH}_i \| TID_i^n \| r_j)$
Parse M as $(\mathcal{CH}_i, TID_i^n, r_j)$
$\beta \leftarrow h(ID_i \| TID_i^n)$
$\mathcal{R}_i \leftarrow \mathcal{P}_i(\mathcal{CH}_i)$
$SK \leftarrow h(ID_i \| SID_j \| r_i \| r_j \| \beta)$
Accept iff $Verify_j = h(ID_i \| \beta \| \mathcal{R}_i \| SID_j \| SK)$; else **Abort**

5.3 SmartNIC-based Authentication Security Analysis

This section comprehensively evaluates the proposed protocol, examining its security and performance properties under predefined experimental and adversarial assump-

tions. The analysis combines automated verification, formal reasoning, and informal threat assessment to provide a comprehensive view of the protocol's security posture.

5.3.1 AVISPA Simulation Verification-based Analysis

We used the AVISPA (Automated Validation of Internet Security Protocols and Applications) tool [168] to model and analyze the protocol. AVISPA integrates four backends (OFMC, CL-AtSe, SATMC, and TA4SP) into a single platform [80].

- **OFMC (On-the-Fly Model Checker).** OFMC symbolically explores protocol executions using state-space search and abstract interpretation. It supports secrecy and authentication properties but can become computationally expensive for large specifications.
- **CL-AtSe (Constraint-Logic-based Attack Searcher).** CL-AtSe reduces protocol analysis to constraint solving (SAT-based search) and is effective at finding subtle attacks.
- **SATMC (SAT-based Model Checker).** SATMC encodes protocol behaviors as propositional formulas and uses SAT solvers to search for attacks; it handles large instances but may be resource-intensive.
- **TA4SP.** TA4SP approximates intruder knowledge using tree automata and rewriting. It can analyze complex protocols but may be less precise for some properties.

Our protocol includes three participant types: ASD, ASDCS, and PCS. We define the following roles in AVISPA to represent these participants:

- role_ASD
- role_ASDCS
- role_PCS
- role_SN (session)
- role_environment

Each role is defined using the HLPSL specification, which includes parameters, states, and corresponding operations.

```
% OFMC
% Version of 2006/02/13
SUMMARY
  SAFE
DETAILS
  BOUNDED_NUMBER_OF_SESSIONS
PROTOCOL
  /home/span/span/testsuite/results/ranabest2.if
GOAL
  as_specified
BACKEND
  OFMC
COMMENTS
STATISTICS
  parseTime: 0.00s
  searchTime: 0.01s
  visitedNodes: 15 nodes
  depth: 5 plies
```

Figure 5.6: AVISPA OFMC Result

```
SUMMARY
  SAFE

DETAILS
  BOUNDED_NUMBER_OF_SESSIONS
  TYPED_MODEL

PROTOCOL
  /home/span/span/testsuite/results/ranabest2.if

GOAL
  As Specified

BACKEND
  CL-AtSe

STATISTICS

  Analysed   : 8 states
  Reachable  : 4 states
  Translation: 0.00 seconds
  Computation: 0.00 seconds
```

Figure 5.7: AVISPA CL-ATSE Result

We simulated our protocol using the OFMC and CL-AtSe backends; the results are shown in Figure 5.6 and Figure 5.7, respectively. The simulation results indicate that our protocol satisfies the security requirements and is safe from active and passive attacks.

However, it is important to note that SATMC and TA4SP do not support the bitwise XOR operation. This is a limitation of these backends and prevents us from using them to simulate our protocol.

5.3.2 Formal Analysis

A rigorous security analysis is conducted using the RoR model [169]. The RoR model allows us to prove the protocol's security under various attack scenarios, including impersonation, session key compromise, and replay attacks. In the RoR model, an adversary $\mathcal{A}$ interacts with the t-th instances of the involved entities: $\mathcal{ASD}_i$, $\mathcal{ASDCS}_j$, and $\mathcal{PCS}$. Thus, the model considers three instances: $P^{t_1}_{\mathcal{ASD}_i}$, $P^{t_2}_{\mathcal{ASDCS}_j}$, and $P^{t_3}_{\mathcal{PCS}}$, representing the t_1-th, t_2-th, and t_3-th instances of $\mathcal{ASD}_i$, $\mathcal{ASDCS}_j$, and $\mathcal{PCS}$, respectively. The RoR model includes several queries that simulate different types of attacks, such as $Test$, $Execute$, $Send$, $Reveal$, and $CorruptMD$. These queries allow analysis of the protocol's security under various scenarios. Table 5.3 presents these queries and their descriptions.

Table 5.3: Queries with Descriptions

Query	**Description**
$Send(P_t, \text{msg})$	This query models an active attack, where $\mathcal{A}$ can transmit a message msg to an instance P_t, and P_t responds accordingly.
$Reveal(P_t)$	This query allows the revelation of the shared session key SK between P_t and its counterpart to $\mathcal{A}$.
$CorruptMD(P^{t_1}_{\mathcal{ASD}_i})$	$\mathcal{A}$ can obtain the values $\{TID_i, T_i\}$ stored in the device $\mathcal{TR}_i$ of $\mathcal{ASD}_i$.
$Test(P_t)$	$\mathcal{A}$ requests P_t to determine the correct value of the session key SK, and P_t responds with a probabilistic outcome of an unbiased coin flip.
$Execute(P^{t_1}_{\mathcal{ASD}_i}, P^{t_2}_{\mathcal{ASDCS}_j}, P^{t_3}_{\mathcal{PCS}})$	This query allows $\mathcal{A}$ to eavesdrop on the communication messages between $\mathcal{ASD}_i$, $\mathcal{ASDCS}_j$, and $\mathcal{PCS}$.

We introduce Theorem 1, which states that DPUAUT is secure against session-key (SK) attacks under the stated assumptions. All participants, including $\mathcal{A}$, have access to the one-way hash function $h(\cdot)$, modeled as a random oracle [170]. The proof uses the RoR model and the oracle queries in Table 5.3.

Informally, Theorem 1 bounds the adversary's advantage in winning the RoR game in terms of the number of oracle queries (e.g., qr_h for hash queries, qr_s for send queries), the hash range 2^l, and the size of the hash space $|Hash|$.

The advantage of an adversary $\mathcal{A}$ in obtaining the session key is bounded by:

$$Adv^{\mathcal{A}}_{DPUAUT}(t) \leq \frac{qr_h^2}{|Hash|} + 2\max\left\{C' \cdot qr_s^{s'}, \frac{qr_s}{2^l}\right\}. \tag{5.14}$$

Here C' is a constant and s' is a parameter related to certain oracle simulations in the RoR model. The bound follows from a sequence of game-hopping reductions described next.

Proof sketch: We follow standard game-hopping techniques (e.g., [75], [171], [172]) and define games $G0$–$G3$. Let $Succ_j$ denote the event that an adversary correctly guesses the challenge bit in game G_j. We write $Adv^{\mathcal{A},G_j}_{DPUAUT} = \Pr[Succ_j]$.

- Game $G0$ corresponds to the real attack against DPUAUT in the RoR model. The adversary interacts with instances $P^{t_1}_{\mathcal{ASD}_i}$, $P^{t_2}_{\mathcal{ASDCS}_j}$, and $P^{t_3}_{\mathcal{PCS}}$ via the oracle queries in Table 5.3. We relate the adversary's advantage in $G0$ to the real-world advantage:

$$Adv^{\mathcal{A}}_{DPUAUT} = \left|2 \cdot Adv^{\mathcal{A}}_{G0} - 1\right|. \tag{5.15}$$

- Game $G1$ considers attacks that include device corruption (modeled by the $CorruptMD$ oracle) and message interception ($Execute$). The adversary may intercept (TID_i, T_i), (TID_i, SID_j), α, and $(T_j, Verify_j)$.

  The session key is computed as $SK = h(ID_i \| SID_j \| r_i \| r_j \| \beta)$, where r_i and r_j are nonces and β is derived material. Recovering SK requires both short-term values (nonces and updated TID) and long-term secrets (e.g., ID_i, SID_j). Because short-term values are refreshed and long-term secrets are protected, the adversary's ability to win $G1$ remains limited even with $CorruptMD$.

- Game $G2$ examines the effect of $Hash$ and $Send$ queries. The protocol uses the one-way hash function $h(\cdot)$ to secure computations; assuming $h(\cdot)$ is collision resistant, collisions during the simulated queries are unlikely. Games $G1$ and $G2$ differ only in how these oracles are simulated. We bound their advantage difference by:

$$|Adv^{\mathcal{A},G1}_{DPUAUT} - Adv^{\mathcal{A},G2}_{DPUAUT}| \leq \frac{qr_h^2}{2|Hash|}. \tag{5.16}$$

- Game $G3$ extends $G2$ with a simulated $CorruptMD$ query. If the adversary executes $CorruptMD$, it may obtain stored values such as $\{TID_i, \mathcal{K}_i\}$. However, reconstructing the PUF's internal challenge–response behavior or deriving the session key remains computationally infeasible without access to correct PUF responses. Therefore, the change in advantage from simulating $CorruptMD$ is bounded. Formally:

$$|Adv^{\mathcal{A},G2}_{DPUAUT} - Adv^{\mathcal{A},G3}_{DPUAUT}| \leq \max\left\{C' \cdot qr_s^{s'}, \frac{qr_s}{2^l}\right\}, \tag{5.17}$$

  where C' depends on the hardness of the $CorruptMD$ simulation and the PUF's security.

If the adversary cannot do better than guessing the test bit in $G3$, its advantage there is $1/2$. Combining the bounds from the games and applying the triangle inequality yields the claimed bound. In particular,

$$Adv^{\mathcal{A}}_{DPUAUT}(t) \leq \frac{qr_h^2}{2|Hash|} + \max\left\{C' \cdot qr_s^{s'}, \frac{qr_s}{2^l}\right\}. \tag{5.18}$$

5.3.3 Informal Security Analysis

This section gives an informal security assessment of the proposed protocol and its resilience against common threats: physical tampering, cloud-side attacks, masquerading/impersonation, anonymity and privacy breaches, untraceability, and desynchronization.

5.3.3.1 Resilience against Physical Attacks

If an adversary tampers with device storage $\mathcal{TR}_i$ or attempts invasive modification of the PUF, the PUF response will typically change and verification will fail. Because

authentication depends on reproducing the correct PUF response $\mathcal{R}_i$, tampering leads to authentication rejection. While physical extraction remains a practical concern, using a PUF together with tamper-resistant storage raises the attacker's cost and reduces the utility of any extracted state.

5.3.3.2 Masquerading and Impersonation

Messages in the registration and authentication phases travel over public channels and can be eavesdropped. However, forging valid protocol messages requires knowledge of secret values (identities, keys, or nonces) that the adversary does not possess under our assumptions.

Device registration messages contain $Verify_i = h(ID_i \| \mathcal{K}_i \| SID_j \| r_i)$ and $T_i = ID_i \oplus r_i$. An adversary who lacks ID_i, $\mathcal{K}_i$, or r_i cannot compute $Verify_i$ or T_i, so device impersonation is infeasible under the assumed cryptographic primitives.

5.3.3.3 Masquerading of *PCS*

In this context, impersonation of $\mathcal{PCS}$ refers to an adversary attempting to forge protocol messages that appear to originate from the trusted authority. To impersonate $\mathcal{PCS}$ an adversary must produce a valid pair $(T_j, Verify_j)$ destined for $\mathcal{ASD}_i$. Computing $Verify_j$ requires values such as ID_i, the PUF response $\mathcal{R}_i$, SID_j, and the session key SK. These values are either protected by encryption or derived from secrets held by legitimate servers. In particular, ID_i and other sensitive fields are carried inside the encrypted blob α, which is protected by $\mathcal{K}_j$. Since the adversary is assumed not to know $\mathcal{K}_j$, it cannot recover the necessary ingredients and therefore cannot forge a valid $(T_j, Verify_j)$ originating from $\mathcal{PCS}$.

5.3.3.4 Masquerading of *ASDCS*

Impersonation of $\mathcal{ASDCS}_j$ refers to forging edge-side responses intended for swarm devices during the authentication phase. The message α sent from $\mathcal{PCS}$ to $\mathcal{ASDCS}_j$ contains encrypted parameters $(ID_i, \langle \mathcal{C}_j, \mathcal{R}_j \rangle, TID_i^n, \mathcal{K}_i)$ under $\mathcal{K}_j$. Without knowledge of $\mathcal{K}_j$ it is computationally infeasible for an adversary to construct or modify α to impersonate $\mathcal{ASDCS}_j$.

5.3.3.5 Device Anonymity and Privacy

Device anonymity is preserved by masking ID_i in transit: $T_i = ID_i \oplus r_i$, where r_i is a fresh session nonce. An adversary that does not know r_i cannot recover ID_i from T_i, preventing straightforward identity disclosure.

5.3.3.6 Untraceability

Because each session uses fresh nonces and temporary identities, registration messages $(TID_i, T_i, Verify_i)$ vary across sessions. This prevents an adversary from linking messages from different sessions to the same device, providing untraceability.

5.3.3.7 Resilience against Desynchronization

Desynchronization attacks attempt to disrupt state updates between parties. Our protocol requires successful verification of

$$Verify_j \stackrel{?}{=} h(ID_i \| \beta \| \mathcal{R}_i \| SID_j \| SK)$$

before any state (for example $\mathcal{TID}_i^{new}$) is updated. If verification fails, the device does not update state and the session is aborted. This conservative behaviour limits the impact of message loss or injection attacks and helps maintain protocol synchronization.

5.3.4 Comprehensive Performance and Security Features Analysis

In this section, we provide a comprehensive performance and security features analysis of our proposed protocol with several state-of-the-art protocols[85], [88], [91], [92], [94], [96], [98]–[101], [173]. Our scheme achieves all the security features outlined in Table 5.4, including mutual authentication, device anonymity, perfect forward secrecy, resistance to replay attacks, key agreement, defense against device impersonation attacks, resilience to de-synchronization attacks, formal security analysis,

and informal security analysis. In our related work section, we provide a detailed analysis of why existing schemes fall short of delivering comprehensive security coverage. The performance comparison is based on key performance metrics, including computation overhead, communication overhead, and each protocol's security features. We analyze the computational requirements of each protocol to assess the efficiency of our proposed solution. Also, we look over the communication overhead incurred during the execution of the protocols, considering the number and size of messages sent between the involved entities.

Table 5.4: Evaluation of authentication schemes based on these security properties A1: Mutual Authentication A2: Device Anonymity, A3: Provide Perfect Forward Secrecy, A4: Replay Attack, A5: Key Agreement, A6: Device Impersonation Attack, A7: Withstand De-synchronization attack, A8: Formal Security Analysis, A9: Informal Security Analysis

Schemes	**Security Properties**								
	A1	**A2**	**A3**	**A4**	**A5**	**A6**	**A7**	**A8**	**A9**
[88]	✓	✓	✓	✓	✓	✓	×	✓	×
[91]	✓	✓	×	✓	✓	✓	×	×	×
[92]	✓	✓	×	✓	✓	×	✓	✓	×
[96]	✓	✓	✓	✓	✓	✓	×	✓	✓
[94]	✓	×	×	×	×	✓	×	✓	×
[98]	✓	✓	✓	×	✓	✓	×	✓	×
[99]	✓	✓	✓	✓	✓	✓	×	✓	✓
[100]	✓	✓	×	✓	✓	✓	×	✓	✓
DPUAUT	✓	✓	✓	✓	✓	✓	✓	✓	✓

Furthermore, we evaluate security features and performance across comparable protocols. By conducting this comprehensive performance comparison, we demonstrate the advantages of the proposed protocol in terms of efficiency and security.

Our proposed protocol comprises three entities, namely $\mathcal{ASD}_i$, $\mathcal{ASDCS}_j$, and $\mathcal{PCS}$.

Similarly, the state-of-the-art protocols [85], [88], [92], [93], [101] consider either two ($\mathcal{ASD}_i, \mathcal{ASDCS}_j$) or three entities ($\mathcal{ASD}_i, \mathcal{ASDCS}_j, \mathcal{PCS}$). In our implementation we focus on the cryptographic operations executed at the swarm devices and server-side. The swarm devices are equipped with NVIDIA BlueField-2 DPUs, which provide hardware acceleration for cryptographic operations required by the protocol. Server-side cryptographic operations run on Dell PowerEdge R760 machines

(128-core Intel Xeon, 256 GB RAM) in our private cloud.

Table 5.5: Execution Time of Primitive Operations

Operation	**Description**	**Execution Time**	
		Swarm Device	Dell Server
T_{hf}	Hash Function	0.096 ms	0.001 ms
T_m	Multiplication	0.084 ms	0.01 ms
T_AESe-d	Encryption-Decryption	0.091 ms	0.01 ms
T_b	Pairing Operation	1.6 ms	0.06 ms

To evaluate the performance of the proposed protocol, we conducted a detailed analysis of the computation overhead. We compared our protocol with several related state-of-the-art protocols, including those referenced in [88], [91]–[93], [96], [101], [173]. The results, summarized in Table 5.6, indicate that our proposed protocol exhibits lower computation overhead compared to the state-of-the-art protocols. The computation overhead of our proposed protocol, as compared to various state-of-the-art protocols, is visually compared in Figure 5.8. The protocol listing is presented horizontally, with the computation time associated with each protocol displayed on the vertical axis. Analysis of Figure 5.8 reveals a lower computation overhead at both ends, namely $\mathcal{ASD}i$ and $\mathcal{PCS}j$, in our proposed protocol compared to the corresponding ends in the state-of-the-art protocols. Additionally, the overall computation overhead of our protocol stands out as significantly less than that of the state-of-the-art protocols.

Table 5.6: Analysis of Computation and Communication Aggregated Overheads

Protocols	$\mathcal{ASD}_i$ Side	$\mathcal{ASDCS}_j$ Side	Computation Overhead	Communication Overhead
DPUAUT	$3T_h \approx 6.06$ ms	$3T_h + 1T_{e/d} \approx 0.01$ ms	6.161 ms	1052 bits
[96]	$3T_h + 1T_{e/d} \approx 8.888$ ms	$3T_h + 1T_{e/d} \approx 0.373$ ms	9.261 ms	2145 bits
[92]	$5T_h \approx 11.755$ ms	$4T_h \approx 0.392$ ms	12.147 ms	1568 bits
[91]	$6T_h \approx 14.106$ ms	$6T_h \approx 0.588$ ms	14.694 ms	1600 bits
[88]	$9T_h + 2T_{pm} \approx 24.405$ ms	$6T_h + 1T_b + 1T_{pm} \approx 1.756$ ms	26.161 ms	1664 bits

These results emphasize our proposed authentication protocol's efficiency and lightweight nature, making it a promising solution for secure authentication in autonomous vehicle swarm systems.

We have analyzed the computation overhead of our proposed protocol and other state-of-the-art protocols. To do this, we have used the computation times of primitive operations listed in Table 5.5. As the registration process is a singular event, our evaluation solely concentrates on cryptographic operations during authentication.

This approach enables us to determine the computation overhead with precision. This analysis includes our protocol as well as those presented in previous works [88], [91]–[93], [96], [101]. [174].

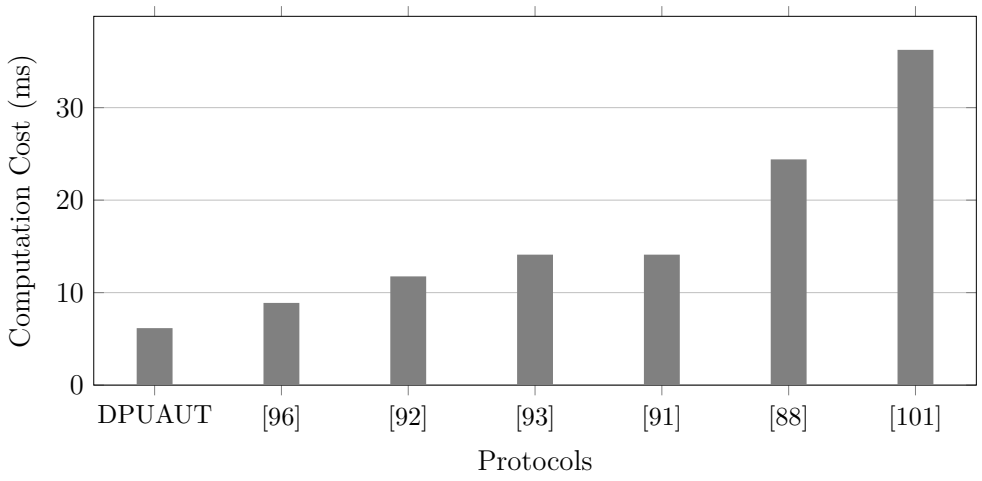


Figure 5.8: Computation Cost Comparison

In our proposed protocol, $\mathcal{ASD}_i$ performs $3|h|$ cryptographic operations during the registration and authentication phases, resulting in a computation time of $(3 \times 2.02) \approx 6.06$ ms. Similarly, $\mathcal{ASDCS}_j$ performs $3|h| + 1|e/d|$ cryptographic operations during the registration and authentication phases, resulting in a computation time of $(3 \times 0.091) + (1 \times 0.01) \approx 0.101$ ms. Therefore, our proposed protocol's total computation cost for the registration and authentication phases is $(6.06 + 0.101) \approx 6.161$ ms.

In secure communication protocols, communication overhead refers to the number of bits required for message exchange among the entities involved. This section thoroughly compares communication overhead between our proposed protocol and the current state-of-the-art protocols [88], [92], [93], [101]. Our analysis focuses on the messages depicted in Figure 5.5, and our assumptions for computing the communication overhead are presented in Table 5.6.

Figure 5.9 compares the communication overhead of our proposed and state-of-the-art protocols. The comparison chart showcases the protocols being analyzed, with the X-axis indicating the protocols and the Y-axis representing the communication overhead linked to each. Figure 5.9 shows that the proposed protocol outperforms all related state-of-the-art protocols regarding communication overhead.

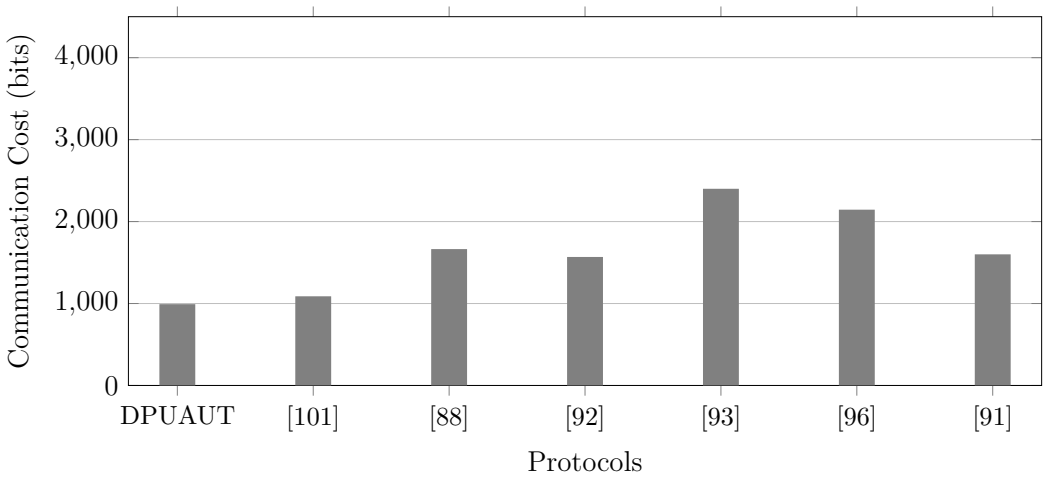


Figure 5.9: Communication Cost Comparison

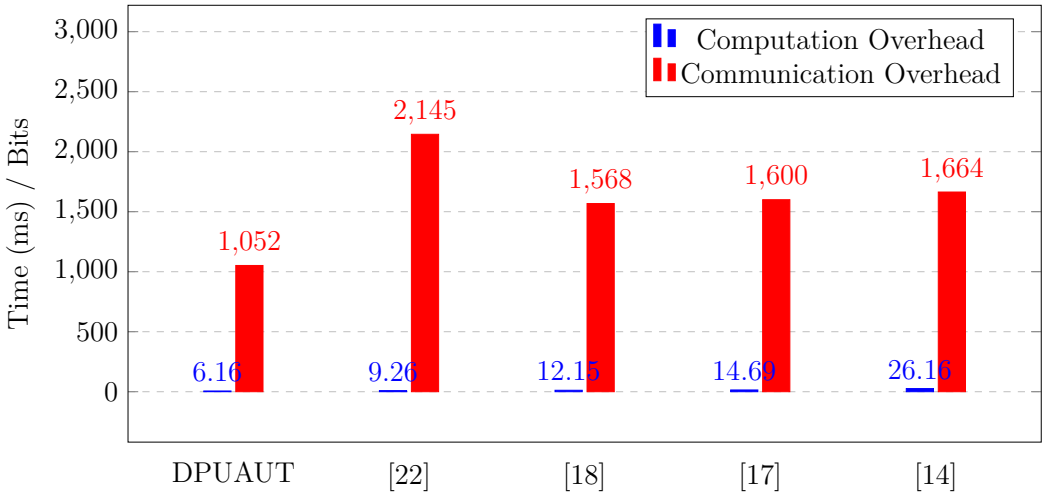


Figure 5.10: Analysis of Aggregated Overheads

Figure 5.10 shows that the proposed protocol outperforms all related state-of-the-art protocols regarding aggregated computational overhead.

5.4 Chapter Summary

This chapter introduced *DPUAUT*, a secure authentication protocol tailored to Intelligent Swarm Systems (ISS), where large numbers of autonomous devices must join and communicate under tight latency constraints and realistic adversarial conditions. The starting point was the mismatch between swarm requirements and cloud-centric security: remote authentication and key management can be slow to react, difficult to scale, and exposed to interception when traffic crosses public or heterogeneous links. Even when edge computing is adopted, the core security prob-

lems remain non-trivial: devices still need strong identity verification, controlled access, and privacy protection. A recurring weakness in many existing schemes is that long-term secrets are stored in device memory, so a physically captured device can leak credentials and enable impersonation.

To address this, the chapter adopted PUF-anchored identity as a hardware-rooted trust mechanism. Since a PUF response depends on intrinsic manufacturing variations, it becomes difficult to replicate or clone the device identity in a practical setting. We described the SmartNiC-based PUF implementation within NVIDIA BlueField hardware, and stated the main requirements that matter for swarm deployments: resistance to physical and learning-based attacks, small footprint, and low overhead. We also formalized the system model and notation used throughout the chapter, to keep the protocol description and analysis consistent.

The protocol design assumes an ISS composed of three entities: the swarm device $\mathcal{ASD}_i$, $\mathcal{ASDCS}_j$, and a trusted private cloud server $\mathcal{PCS}$. DPUAUT is organized into four phases: initialization, device registration, $\mathcal{ASDCS}$ registration, and authentication. During initialization, $\mathcal{PCS}$ selects $\mathcal{MSK}$ and publishes the hash function. During registration, $\mathcal{PCS}$ enrolls a device using PUF challenge–response pairs and issues a temporary identity and derived secret key material. $\mathcal{ASDCS}$ registration is performed offline and relies on signature-based integrity checks before the server receives its credentials. In the authentication phase, $\mathcal{ASD}_i$ and $\mathcal{ASDCS}_j$ achieve mutual authentication and derive a session key using fresh nonces, hash-based verifiers, temporary identifiers, and the PUF response. The design goal is explicit: reduce round complexity and limit what is exposed on the public channel while still supporting anonymity and untraceability.

A core architectural decision in this chapter is the offload of PUF verification logic to a SmartNiC-based DPU. The rationale is not only performance. Offload reduces authentication latency and improves scalability because the DPU can serve many swarm devices under constrained CPU budgets. It also helps security isolation because verifier logic and sensitive verification steps are moved away from general-purpose hosts. The chapter framed this as a practical step toward deploying

authentication at scale in an edge-heavy swarm environment.

Security evaluation was presented in three layers. First, we verified the protocol using AVISPA (OFMC and CL-AtSe backends) and obtained *SAFE* results under the supported modeling features; we also noted the backend limitation regarding XOR for SATMC and TA4SP. Second, we provided a formal proof sketch under the RoR model using a game-hopping argument, bounding the adversary's advantage in deriving the session key in terms of oracle queries and hash security. Third, we discussed an informal security analysis that targets the practical threats expected in swarm systems: physical tampering, masquerading/impersonation, MiTM modification, replay, device traceability, and de-syn.

Chapter 6

RDMA Offload Using SmartNIC

Remote Direct Memory Access (RDMA) is a cornerstone of high-throughput, low-latency communication for HPC, AI, and disaggregated storage. While RDMA performance in commodity servers using RDMA Network Interface Cards (RNICs) is well-documented, ARM-based SmartNICs and Data Processing Units (DPUs) remain underexplored. In this thesis, we benchmark RDMA performance on BlueField-2 (BF2) and BlueField-3 (BF3) using Perftest and DOCA RDMA APIs. We evaluate bandwidth throughput, latency, and message rate under varying queue depths, postlist sizes, inline thresholds, and completion queue moderation. Bluefield-3 (BF3) achieves up to 3.18 times higher bandwidth and 50% lower latency compared to Bluefield-2 (BF2), reaching 48.4 GB/s and 63 million packets per second (Mpps) under optimal settings. Beyond empirical evaluation, we recommend SmartNIC-aware workload mappings and API strategies for RDMA-enabled cloud services. These experimental results demonstrate BF3s architectural advantages, including more ARM cores, increased DDR5 memory, PCIe Gen5, and guidance for RDMA deployment in performance-critical, cloud-native environments. In contrast to these studies, our work systematically benchmarks RDMA read, write, and latency performance across various queue depths, postlist sizes, and completion moderation settings, in both BF2 and BF3 hardware, using line-rate (200 Gb/s and 400 Gb/s) SmartNICs. To the best of our knowledge, this is the first empirical comparison of BF2 and BF3 using both Perftest and DOCA RDMA APIs at such speeds, under realistic tuning configurations.

6.1 RDMA Theoretical Performance Model

To understand the RDMA performance improvements between BF2 and BF3, we present a theoretical performance model that captures key architectural limits and evaluate it against measured data. Our analysis focuses on both throughput and latency across RDMA modes.

6.1.1 Bandwidth Scaling Model

The effective RDMA throughput T_{RDMA} can be modeled as the minimum of three critical bottlenecks:

$$T_{\text{RDMA}} = \min(BW_{\text{mem}}, BW_{\text{nic}}, BW_{\text{qp}}) \tag{6.1}$$

Where:

- BW_{mem} is the available memory bandwidth,
- BW_{nic} is the NIC wire-speed bandwidth,
- BW_{qp} is the throughput limit imposed by QP queue handling (e.g., QP depth, postlist rate).

We assume a per-core QP mapping to avoid lock contention, memory accesses are within the same NUMA domain, and that the system avoids retransmissions or buffer overruns. We also consider software overhead to be minimal due to kernel bypass in RDMA paths. **Theoretical Limits:**

- **BF2 DDR4-3200:** $2 \times 3200 \times 8 = 51.2$ GB/s
- **BF3 DDR5-5600:** $2 \times 5600 \times 8 = 89.6$ GB/s
- **NIC Link Speeds:**
  - BF2 and BF3 (DPU Mode): 200 Gb/s = 25 GB/s
  - BF3 (Host Mode): 400 Gb/s = 50 GB/s

Under ideal conditions, these values set upper bounds. However, real-world performance may exceed theoretical scaling due to improved microarchitecture. Our observed results suggest that architectural upgrades in BF3, such as PCIe Gen5, upgraded DMA engines, and cache improvements, contribute significantly beyond raw bandwidth increases.

6.1.2 Latency Considerations

Latency in RDMA communications can be modeled as:

$$L = L_{\text{queue}} + L_{\text{copy}} + L_{\text{nic}} + L_{\text{sync}} \tag{6.2}$$

Where:

- L_{queue} is the work queue posting overhead,
- L_{copy} is the DMA read/write latency from DRAM,
- L_{nic} is the link transmission delay,
- L_{sync} is the cost of CQ signaling and user-space polling.

Our measurements indicate significant latency reduction from BF2 to BF3:

- **BF2 DPU (512 B):** 2.24 μs
- **BF3 DPU (512 B):** 1.29 μs
- **BF3 Host Mode (512 B):** 1.12 μs

This $\sim$50% reduction is attributed primarily to reduced L_{sync} via more efficient interrupt moderation and faster memory access.

6.1.3 Model Validation and Gains

To validate our theoretical model, we compare the models predicted performance improvements against the actual observed gains, as summarized in Table 6.1. The results show that the observed message rate and bandwidth in BF3 Host Mode closely

match the theoretical predictions, even slightly surpassing the expected NIC-only improvements. This close alignment between prediction and observation confirms that the model accurately captures the performance gains.

Clarification on theoretical vs. measured latency. It is important to clarify the distinction between theoretical and measured results in this analysis. The latency values reported for BF2 and BF3 are obtained from direct measurements using RDMA microbenchmarks. The term *theoretical* refers exclusively to the expected architectural improvement derived from reduced synchronization overhead, faster memory access, and improved DMA and PCIe subsystems, rather than to an analytically computed latency value. Table 6.1 therefore compares *architectural expectations* against *measured latency reductions*, showing close agreement between predicted trends and observed performance.

Table 6.1: Architectural Expectations vs. Measured RDMA Performance Gains

Metric	**Theoretical Gain**	**Observed Gain**
Memory Bandwidth	1.75×	2.27×
NIC Bandwidth	2×	4.84× (host mode)
Latency	1.12 μs	∼50% lower
Message Rate	512	2.273.18×

6.2 SmartNIC RDMA Performance Results Analysis

We present RDMA performance results collected across BF2 and BF3 SmartNICs, highlighting key metrics, bandwidth, latency, and message rate. Tests were conducted under both DPU and host modes, using DOCA and Perftest tools with controlled parameters.

6.2.1 Hardware Configuration

Our evaluation utilizes both NVIDIA BF2 and BF3 SmartNICs, configured in DPU and host modes depending on the experiment. The BF2 NIC includes 8 Arm Cortex-

A72 cores, dual-channel DDR4-3200 memory, and a 200Gbps Ethernet interface over PCIe Gen4. The BF3 NIC incorporates 16 Arm Cortex-A78 cores, dual-channel DDR5-5600 memory, and supports PCIe Gen5 along with 400Gbps networking. Host-to-host evaluations were conducted by operating BF3 in DPU mode, offloading RDMA functionality to the x86 host processor.

All tests were run on AMD EPYC 7742-based servers, each equipped with 64 physical cores, 512GB DDR4 memory, and running Ubuntu 22.04 with Linux kernel version 5.15.98. The BlueField has a default Ubuntu-based BlueField OS with kernel version 5.10.0. Firmware versions were Mlx 24.38.1002 for BF2 and 28.40.1012 for BF3. CPU frequency governors were set to performance mode, and SMT was enabled with IOMMU disabled for consistency. Process affinity and interrupt pinning were enforced to reduce scheduling variability. RoCE settings were carefully configured, with MTU set to 9000 bytes, ECN enabled, and Priority Flow Control (PFC) active on all test links.

6.2.2 Software Tools and Stack

The evaluation employs two primary tools for RDMA benchmarking: the Perftest suite (v4.5) and NVIDIAs DOCA RDMA API (v2.10). The Perftest utilities `ib_write_bw`, `ib_read_bw`, and `ib_write_lat` were used for latency and throughput characterization across different transport types and queue configurations. DOCA RDMA was used to assess software-level offload efficiencies in both DPU and host modes. The RDMA stack configuration was identical across devices, including support for RoCEv2 transport.

6.2.3 Benchmark Configuration

The experiments were conducted using fixed message sizes ranging from 128 bytes to 1MiB. A consistent queue depth of 512 was used to saturate the link while preventing overruns. Four queue pairs were instantiated to distribute load across available cores and NIC queues. The postlist size was set to 64 to batch work requests, and inline data was enabled with a 64-byte threshold for efficient small message handling.

Completion queue moderation (`cq_mod`) was varied from 1 to 128 to study its impact on interrupt coalescing and performance under load.

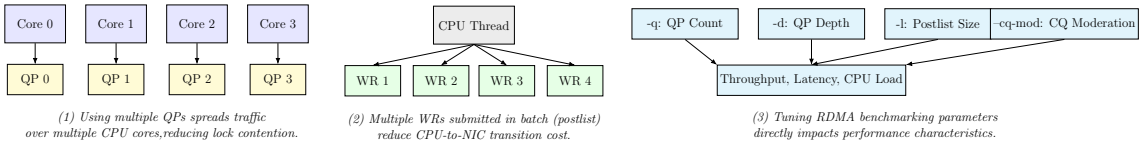


Figure 6.1: RDMA optimization techniques

6.2.3.1 x86 Host vs. Arm DPU Execution

Both host and DPU mode tests used the same RDMA transport and tools, but showed different performance profiles due to architectural differences. Host mode, executed on x86 CPUs, provided higher instruction throughput and memory bandwidth, achieving near-line-rate RDMA at lower jitter. DPU mode, while constrained by lower single-thread performance, showed more deterministic latency under congestion and reduced host-side CPU interference. Table 6.2 summarizes key system and configuration parameters.

Table 6.2: System and configuration parameters Checklist

Parameter	**Setting**
Host CPU	AMD EPYC 7742, 64-core
SmartNICs	BF2 (200G), BF3 (200G/400G)
PCIe Link	Gen4 (BF2), Gen5 (BF3)
Memory	512 GB DDR4 (host); DDR4/DDR5 (DPU)
Kernel Version	5.15.98 (x86), 5.10.0 (BlueField)
DOCA SDK	Versions 2.10/2.5
Firmware	Mlx FW 24.38.1002 (BF2), 28.40.1012 (BF3)
RoCE Settings	MTU = 9000, ECN = on, PFC = enabled
CPU finetuning	IRQ and process affinity used
BIOS Settings	SMT on, IOMMU off, performance mode enabled

6.2.4 System Tuning Parameters

- Use of deeper QP depths and post-listing (e.g., `-q`, `-r`, `-l`) yields performance gains.
- Use 8–16 QPs for host mode, 4–8 QPs for DPU mode to maximize message rates.

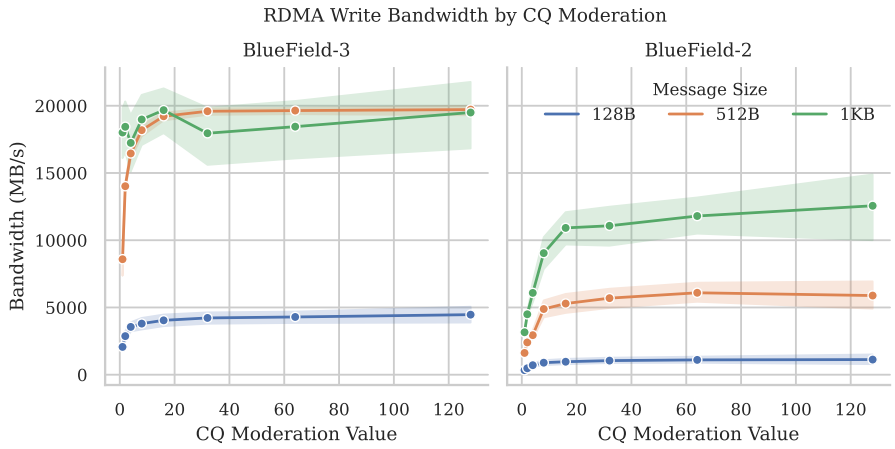


Figure 6.2: Bandwidth variation with increasing completion queue moderation. Moderate values (e.g., 32-64) yield optimal performance.

- BF3 DPU mode for 512-byte workloads offers competitive performance with isolation and security benefits.

6.2.5 Effect of Completion Queue Moderation

To evaluate interrupt moderation effects, we vary the `cq_mod` parameter and record its impact on throughput. Figure 6.1 presents RDMA optimization techniques. In subfigure (1), the distribution of Queue Pairs (QPs) across CPU cores to minimize lock contention, (2) batching of Work Requests (WRs) into postlists to reduce CPU-to-NIC transition overhead, and (3) benchmarking parameters that influence throughput, latency, and CPU load. Figure 6.2 presents the results of the optimization techniques. Proper tuning of `cq_mod` significantly improves performance by reducing interrupt pressure on CPU/DPU cores while retaining responsiveness. This reinforces our observation that BF3's architecture is better suited for high concurrency and sustained queue pressure, particularly in host mode.

6.2.6 Maximum Bandwidth and Latency Performance Comparison

The performance comparison across RDMA metrics between BF3 and BF2 is summarized in Table 6.3, showing up to a 3.8Œ improvement in bandwidth and nearly 50% latency reduction with BF3 in host mode. Table 6.4 summarizes the average

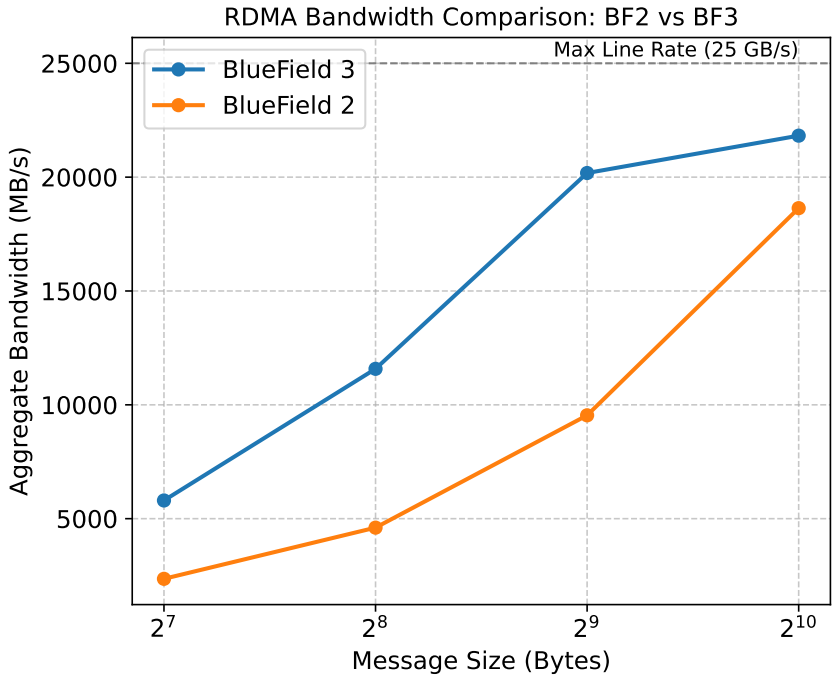


Figure 6.3: Aggregate RDMA write bandwidth across configurations

.

bandwidth achieved using 1 MiB messages across multiple communication configurations. The highest bandwidth of **48,421 MiB/s** was achieved in a server-to-server BF3 configuration at 400 Gbps, indicating near line-rate performance. These results reveal substantial performance gains with BF3: In host mode, BF3 reaches ∼388 GB/s (∼48.4 GB/s) for large messages (1 MiB), achieving near line-rate performance. This reflects a 3.8× improvement over BF2s peak DPU bandwidth. For mid-size packets (512 B), message rates improve by up to 3.18×, with BF3 host mode delivering over 58 million packets per second. BF3 reduces end-to-end RDMA latency by 40%-50% compared to BF2, primarily due to better queue handling, reduced memory contention, and improved interrupt moderation.

Performance scales linearly with link speed when there are no architectural bottlenecks present. BF3 significantly outperforms BF2 across all metrics. Message sizes around 512 B deliver maximum Mpps, while 1 MiB messages maximize bandwidth. As shown in Table 6.6, BF3 significantly outperforms BF2 in bandwidth at all small message sizes. Host mode in particular nearly doubles the throughput for 128-byte and 256-byte messages, confirming the improvements from faster PCIe and memory channels. Average latency values for BF2 and BF3 are summarized

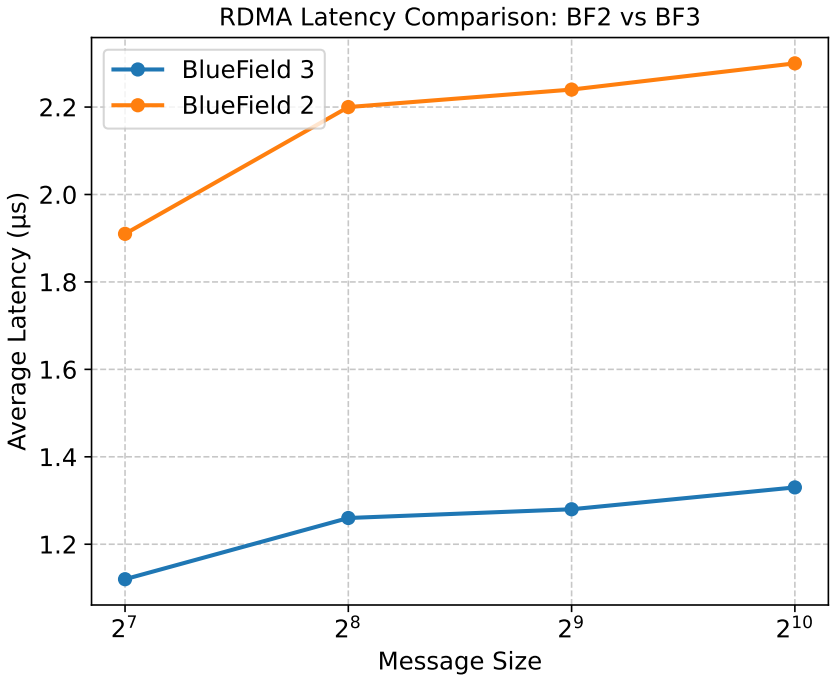


Figure 6.4: End-to-end write latency. BF3 achieves 1.12 μs latency, nearly 50% lower than BF2 for 128B messages.

in Table 6.5. Notably, BF3 achieves sub-1.3 ţs latency for all tested message sizes, with the most dramatic drop observed at 128 bytes (a 41% reduction over BF2).

Table 6.3: Maximum Gains from BF3 (200/400 Gb/s) Compared to BF2 (200 Gb/s)

Metric	**BF3 400G Host Mode**	**BF3 DPU Mode**	**BF2 DPU (Baseline)**	**Gain (BF3 over BF2)**
Bandwidth (1 MiB)	≈ 48,421 MB/s	≈ 22,742 MB/s	≈ 10,004 MB/s	Up to 3.8×
Message Rate (512 B)	58.36 Mpps	44.42 Mpps	19.54 Mpps	2.27×-3.18×
Latency (512 B)	1.12 μs	1.29 μs	2.24 μs	∼40%-50% lower

Figure 6.3 summarizes the peak observed performance for BF3 in both host and DPU modes, relative to the BF2 baseline. Figure 6.4 compares RDMA write latency for 128B and 512B messages across BF2, BF3 DPU, and BF3 host mode.

6.2.7 Maximum Message Rate (Mpps)

Table 6.7 highlights the message rate performance at smaller message sizes (512 B and 1024 B). Server-to-server BF3 achieved the highest throughput of **58.36 Mpps**

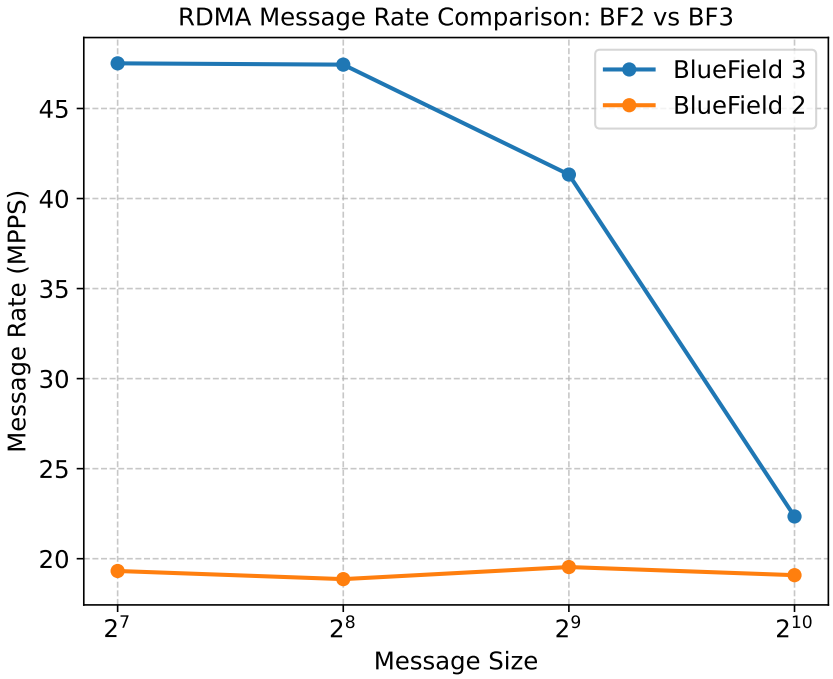


Figure 6.5: Message rate performance across BF2 and BF3 configurations.

Table 6.4: Maximum Bandwidth Performance with different modes (1 MiB Message Size)

Communication Pair	**Speed**	**Avg BW (MiB/s)**	**Notes**
Server → BF3	400 Gbps	48,421.07	High throughput
Server ↔ Server (BF3)	400 Gbps	**48,421.07**	Peak observed bandwidth
Server ↔ Server (BF3)	200 Gbps	23,526.48	Expected scaling
BF3 ↔ BF3	200 Gbps	23,446.45	Balanced bidirectional BW
Server ↔ Server (BF2)	200 Gbps	23,526.46	Comparable to BF3
BF2 ↔ BF2	200 Gbps	11,769.29	Suboptimal

with 512 B message size, demonstrating the efficiency of BF3 for high-frequency, low-latency message exchanges.

Figure 6.5 shows throughput in terms of Mpps for varying packet sizes.

6.3 Toward SmartNIC-Aware Cloud Services and API Recommendations

While traditional RDMA benchmarking provides raw performance metrics, deployment in real-world cloud-native HPC services requires deeper consideration of API support, SmartNIC selection, and orchestration layers. This section maps practical workloads to SmartNIC configurations, enabling optimized infrastructure design.

Table 6.5: Average Latency (ţs) (microseconds)

Size (bytes)	BF2 DPU	BF3 DPU
128	1.91	1.12
256	2.20	1.27
512	2.24	1.29

Table 6.6: Bandwidth Comparison with different message sizes (MB/s)

Size (bytes)	BF2 DPU-DPU	BF3 DPU-DPU	BF3 Host-Host
128	2449.92	6081.28	8101.12
256	4830.72	12144.64	16184.32
512	10004.48	22742.84	22806.20

6.3.1 Service-Driven SmartNIC Recommendation

Different cloud services impose unique demands on bandwidth, latency, and compute offload capabilities. Figure 6.6 summarizes our recommendation matrix linking services, core tasks, DPU models, APIs, and deployment environments. Figure 6.7 illustrates a layered approach to deploying such services. Applications interact with middleware abstractions, which are mapped to container-aware virtualization layers and SmartNIC-accelerated APIs.

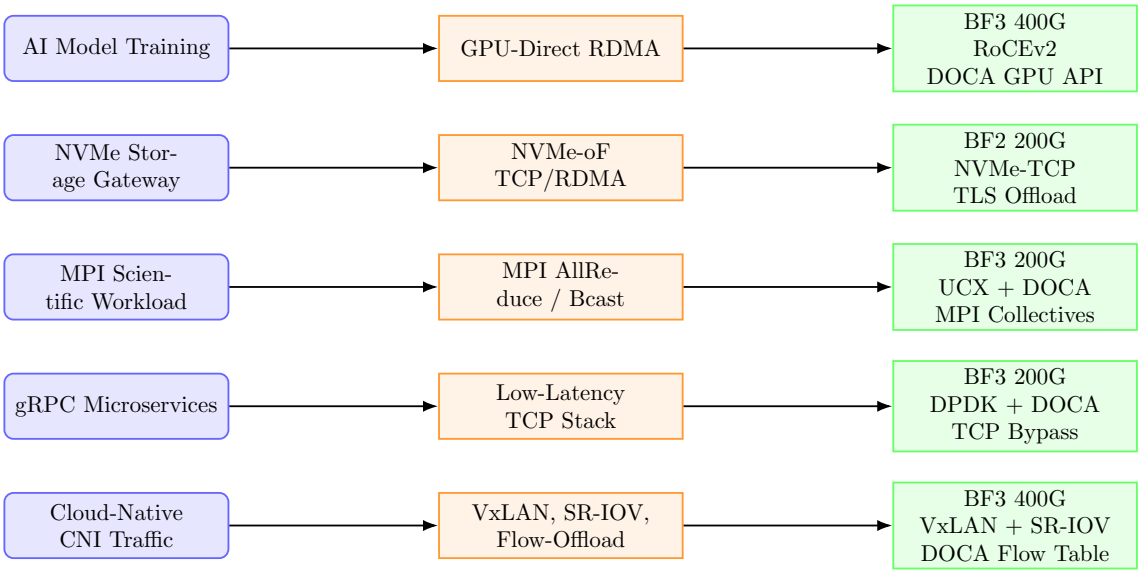


Figure 6.6: Service-Task Mapping to SmartNIC Recommendation

Figure 6.6 illustrates that SmartNICs act as heterogeneous accelerators rather than one-size-fits-all devices. The mapping of workloads to specific BlueField generations and software stacks is guided by four primary design criteria such as, la-

Table 6.7: Maximum Message Rate (Mpps) Performance With Message Size 512 B

Communication	**Speed**	**Mpps**	**BW (MiB/s)**
Server → BF3	400 Gbps	42.63	20,816.32
Server ↔ Server (BF3)	400 Gbps	**58.36**	28,497.96
Server ↔ Server (BF3)	200 Gbps	43.57	21,276.19
BF3 ↔ BF3	200 Gbps	43.38	21,184.49
Server ↔ Server (BF2)	200 Gbps	44.48	21,718.81
BF2 ↔ BF2	200 Gbps	13.65	6,663.71

tency sensitivity, bandwidth and message-rate requirements, data locality and PCIe traversal cost, and availability of hardware/software acceleration primitives on the SmartNIC.

- **AI model training.** GPU-centric workloads benefit most from *GPU-Direct RDMA*, which eliminates host memory copies and minimizes PCIe round-trips. BF3 with 400 Gb/s ConnectX-7 and DOCA GPU APIs is selected to sustain high message rates and overlap communication with computation during distributed training.

- **NVMe storage gateway.** NVMe-oF traffic is throughput-dominated and latency-tolerant compared to control-plane services. BF2 at 200 Gb/s is sufficient to offload NVMe-TCP and TLS, as the bottleneck lies in storage media rather than NIC bandwidth, making BF3 unnecessary for this workload class.

- **MPI scientific workloads.** MPI collectives (e.g., AllReduce, Broadcast) are highly sensitive to message latency and synchronization overhead. BF3 with UCX and DOCA collective offload is selected to exploit hardware-assisted reductions and reduce CPU participation during collective operations.

- **gRPC microservices.** These workloads are dominated by tail latency and connection setup overhead. BF3 with DPDK and DOCA TCP bypass enables low-latency user-space networking and avoids kernel-induced jitter, improving p99 latency.

- **Cloud-native CNI traffic.** Overlay networking (VXLAN, SR-IOV) is data-plane intensive and benefits from hardware flow steering. BF3 at 400 Gb/s

is chosen to offload encapsulation and flow-table management, reducing host CPU load while sustaining high east–west traffic rates.

This mapping highlights that SmartNIC deployment decisions are driven by workload characteristics rather than raw bandwidth alone, reinforcing the role of DPUs as workload-specific accelerators.

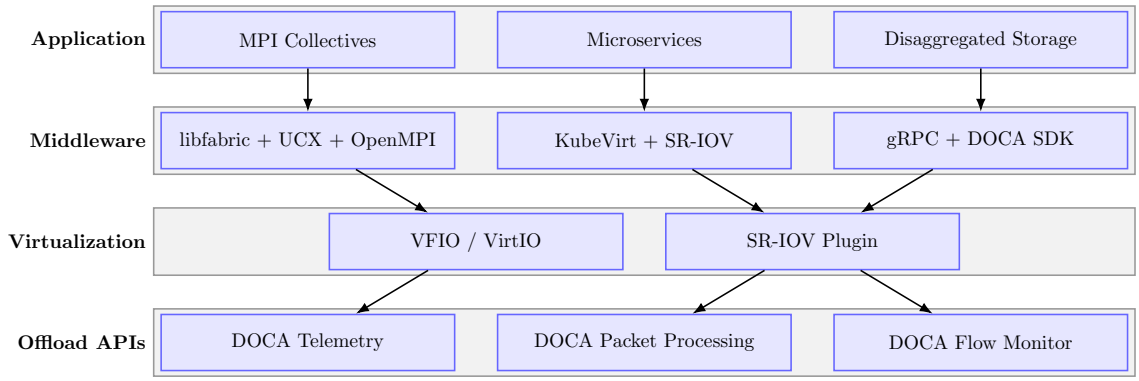


Figure 6.7: API-based architecture leveraging DOCA, SR-IOV, and MPI/UCX layers for SmartNIC orchestration.

6.4 Discussion

6.4.1 Interrupt Moderation and Polling Mode Trade-offs

Optimizing RDMA performance involves tuning system-level parameters beyond raw hardware capabilities. Completion queue (CQ) moderation on BF3 reduces CPU interrupt overheadespecially in host modeenabling higher throughput. However, for small messages (e.g., 128B), increasing CQ moderation beyond a queue depth (QD) of 16 introduces tail latency penalties. We also observe that its benefits diminish when used alongside deep postlist batching, highlighting the need for co-tuning. Additionally, NUMA misalignment on x86 hosts, where NIC interrupts or memory buffers span across NUMA nodes, incurs a 10%-15% throughput drop due to remote memory access latency. Small messages (128B to 512B) are especially sensitive to queue depth, CQ moderation, and PCIe latency. Table 6.8 shows the message rate across configurations, where BF3 host mode achieves up to 66 Mpps at 128Bsignificantly higher than BF2but still below the 195 Mpps theoretical maximum due to software overhead and PCIe signaling delays.

Table 6.8: Message Rate (Mpps) and Bandwidth Comparison with Small Message Sizes with Theoretical max

Size (B)	BF2 DPU-DPU	BF3 DPU-DPU	BF3 Host-Host
128	19.14	47.51	66.17
256	18.87	47.44	65.80
512	19.54	44.41	44.51

6.4.2 Trade-Offs Between DPA and ARM Cores in BF3

While BF3 introduces a Datapath Accelerator (DPA) comprising many-core RISC-V processors to offload packet processing from the primary CPU path, our results, along with prior work, highlight a performance trade-off when compared to the on-path ARM Cortex-A78 cores.

The DPA cores, designed for low-power and parallel execution, typically operate at lower clock frequencies (e.g., 500–1800 MHz) and lack the complex cache hierarchy of the ARM cores. As such, their single-threaded performance is significantly lower. This architectural limitation directly impacts tasks that require fast memory access or involve complex control logic. DPA execution achieves lower IPC and memory throughput, making it less suitable for RDMA queue handling, MPI collectives, or compression-heavy workloads [130]. Moreover, DPA cores run bare-metal firmware, without the support of a full OS stack, which limits their flexibility in scheduling, synchronization, and I/O management. This is in contrast to ARM cores, which run a full Linux environment and are better suited for high-level communication libraries like libfabric or MPI. Programming the DPA also presents practical challenges. Toolchain support is limited to the DOCA DPA SDK, with minimal debugging and profiling capabilities, whereas the ARM cores can leverage the mature Linux/GCC ecosystem for development, optimization, and tracing. Despite these limitations, the DPA has unique advantages. It is well suited for deterministic, off-path tasks such as packet classification, telemetry tagging, flow lookups, and stateless header rewrites. For example, line-rate DPI inspection, flow counting, and traffic shaping functions can be distributed across RISC-V threads for maximum concurrency with minimal host CPU impact. While the ARM cores

in BF3 offer higher flexibility and better overall performance for general-purpose and latency-sensitive workloads, the DPA remains a valuable architectural asset for microservices and off-path compute operations. Optimal usage of BF3 requires task-specific partitioning between these compute domains.

6.5 Chapter Summary

In this chapter, we present a detailed benchmarking study of RDMA performance on NVIDIA BF2 and BF3 DPUs, evaluating both DPU mode and host mode under varying message sizes and tuning parameters. Our results show that BF3 delivers up to 3.18Œ higher message rates and 50% lower latency compared to BF2, achieving peak throughput of 48.4 GB/s and over 63 Mpps. These gains derive from architectural enhancements including DDR5 memory, PCIe Gen5, and increased ARM core counts. Beyond raw performance, we map common workloads, such as MPI collectives, NVMe storage, and AI inference, to appropriate SmartNIC configurations, and recommend API stacks for optimized deployment using DOCA, libfabric, and KubeVirt. We also introduced models for queue depth and postlist scaling, offering practical guidance for SmartNIC-aware orchestration. These findings highlight the growing role of programmable DPUs in high-performance cloud and edge environments. Future work will explore multi-tenant RDMA behavior, deeper integration of DOCA telemetry, and cost-performance analysis using real workload traces.

Chapter 7

SmartNIC-based User Plane Function (UPF)

Traditional 5G user plane function (UPF) architectures are software-based implementations that struggle to maintain performance. To meet the stringent quality of service (QoS) and scalability requirements of 5G under high session and data plane loads, a highly efficient next-generation UPF is required. To address hardware limitations, we propose *6GUPF*, a novel DPU-based programmable UPF that leverages hardware offload, flow-based programmability, and stateful traffic tracking via NVIDIA's DOCA SDK API. Flow management is achieved using Stateful Flow Table (SFT) offload, which enables efficient per-flow tracking in hardware. While the SmartNIC/DPU supports Deep Packet Inspection (DPI) offload, this work leverages SFT for scalable flow state management. We demonstrate 6GUPF's ability to emulate and handle over 500,000 concurrent UE IDs (TEIDs), significantly outperforming existing software-based solutions (DPDK, bmv2-P4) and hardware-based solutions, such as P4-switch (Tofino). In this thesis, we present a next-generation 6GUPF that fully leverages the hardware acceleration of SmartNICs/DPUs. The 6GUPF is developed using DOCA Flow API, which enables fast, programmable packet processing directly in the data path, specifically for the N3 gNB network and N6 interfaces of datanet. The architecture integrates flow-based acceleration for stateful flow tracking and symmetric Receive Side Scaling (RSS) to utilize cores and manage non-blocking states efficiently. This will help minimize latency and

avoid contention in multi-core environments. Our experiments show that hardware-offloaded UPF achieves an aggregate line rate of 400 Gbit/s, supports 1 million concurrent data streams, and scales up to 500,000 active User Equipment (UE) instances while maintaining QoS and session isolation. It also enables 40% lower latency compared to traditional software UPFs. Unlike Tofino-based P4 switch UPF designs, which are limited by SRAM and TCAM constraints when storing large-scale data streams, our DPU-based solution is ideal for high-density Protocol Data Unit (PDU) deployments without sacrificing programmability and performance. It creates a path for cloud-native, 6G UPF deployments that can scale to a wide range of workloads, from ultra-low-latency applications to large-scale IoT backhaul.

In this thesis, we propose a novel architecture for 6GUPF that integrates a DPU to offload computationally intensive tasks such as GTP-U encapsulation, decapsulation, and forwarding by leveraging the hardware acceleration capabilities of NVIDIA's DOCA Flow API. The 6GUPF architecture is depicted in Figure 7.1.

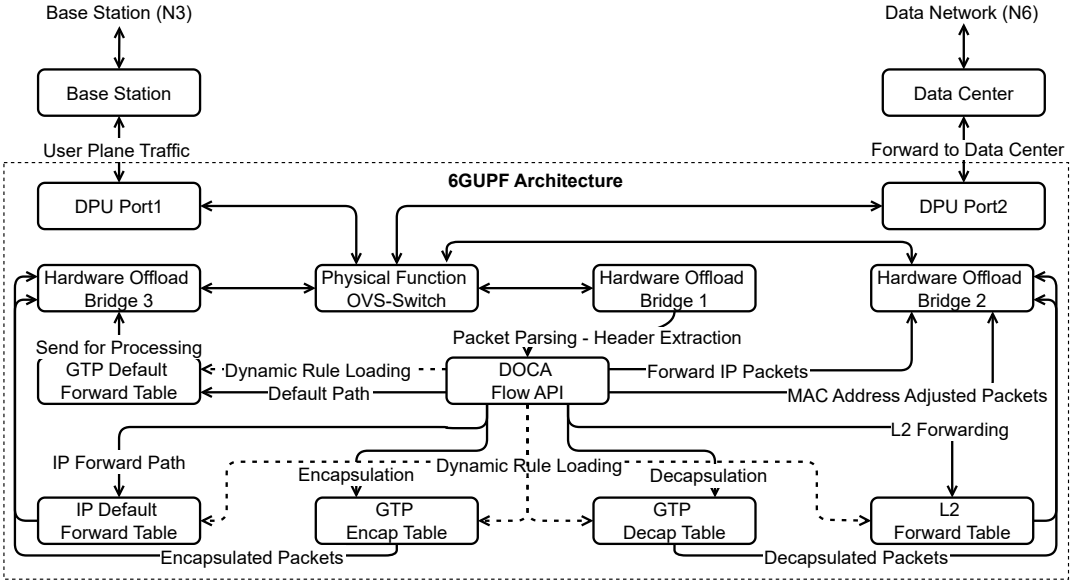


Figure 7.1: Packet Processing and Forwarding in DPU-Based 6GUPF

The ingress pipeline is structured around five primary tables, each performing a distinct function. The *GTP Default Forward Table* routes GTP-U packets for processing, while the *IP Default Forward Table* manages the forwarding of standard IP packets. The *GTP Encap Table* inserts GTP-U headers for outgoing packets, whereas the *GTP Decap Table* removes the GTP-U headers from incoming flows.

Finally, the *L2 Forward Table* ensures correct layer-2 forwarding by modifying the source MAC address as required after encapsulation or decapsulation.

At the initial state, the DOCA Flow pipeline contains only default rules, forwarding all traffic to the GTP/IP Default Forward Tables. During runtime, dynamic rules are installed into the DPU, enabling packets that match specific headers (e.g., TEID, 5-tuple) to be fully processed on the DPU without host intervention. Each table is implemented as a DOCA Flow stage $\mathcal{C}_1 \ldots \mathcal{C}_5$, as summarized in Algorithm 5, which formalizes the $\mathcal{C}_1$ as Ingress/Classify, $\mathcal{C}_2$ as QoS assignment, $\mathcal{C}_3$ as GTP processing, $\mathcal{C}_4$ as L2 Forwarding and $\mathcal{C}_5$ as Egress. If no rule is matched, packets will follow the default path to the host for further processing and potential rule update, ensuring a robust fallback mechanism.

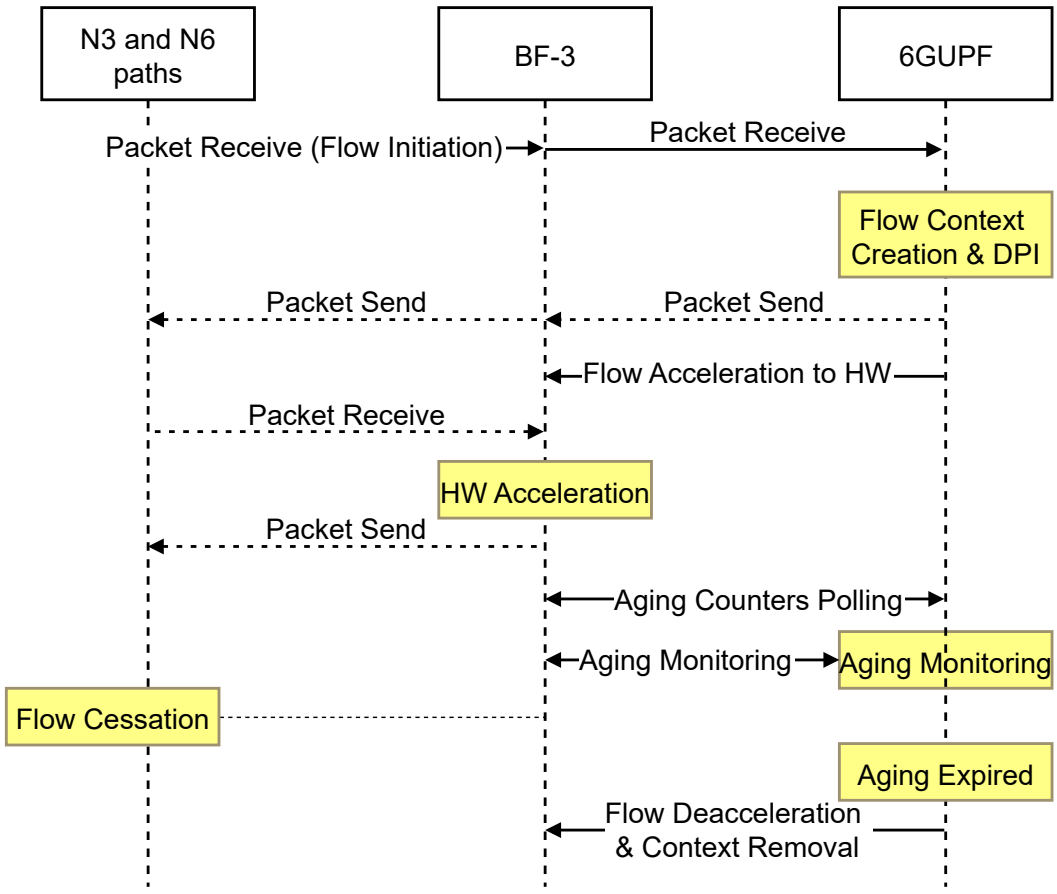


Figure 7.2: Flow Initiation, Acceleration, and Deacceleration Process in a DPU-based 6GUPF

Algorithm 5 DPU-Based GTP-U Flow Handling

Require: DOCA Flow stages $\mathcal{C}_1 \ldots \mathcal{C}_5$: $\mathcal{C}_1$ Ingress/Classify, $\mathcal{C}_2$ QoS assignment, $\mathcal{C}_3$ GTP processing, $\mathcal{C}_4$ L2 Forwarding, $\mathcal{C}_5$ Egress.
Tables $\leftrightarrow$ stages: GTP/IP Default $\rightarrow \mathcal{C}_1$, QoS $\rightarrow \mathcal{C}_2$, Encap/Decap $\rightarrow \mathcal{C}_3$, L2 Fwd $\rightarrow \mathcal{C}_4$.
Load initial rule set $\mathcal{R}$ (e.g., TEID/5-tuple) into DPU.
for each packet p **do**
  $[\mathcal{C}_1]$ Extract headers; lookup $r \in \mathcal{R}$.
  if r found **then**
    Assign flow ID f_i and priority q_p.
    $[\mathcal{C}_2]$ Tag QoS per q_p.
    $[\mathcal{C}_3]$ **if** GTP-U $\Rightarrow$ decap; **elseif** required $\Rightarrow$ encap.
    $[\mathcal{C}_4]$ Resolve next-hop MAC; $[\mathcal{C}_5]$ enqueue to SR-IOV/VF or DN.
  else
    Steer p to host via default rule.
    Control plane update $\mathcal{R}$ (install/learn rule).
  end if
end for

7.0.1 Flow Initialization and Processing in a DPU-based 6GUPF Network

Figure 7.2 describes the sequence of interactions involved in flow initiation, acceleration, aging, and de-acceleration within a 6GUPF network using a DPU. This process involves three primary components: the N3/N6 interfaces, the BF-3/DPU, and the 6GUPF. The sequence flow diagram demonstrates how packets are processed, accelerated, monitored, and eventually deaccelerated based on flow context and aging criteria.

When a new flow is initiated, the N3/N6 interface receives a packet and forwards it to the BF-3 DPU (step 1). The DPU then sends the packet to the 6GUPF (step 2), where a flow context is created and the flow is categorized (step 3). Once the flow context is established, the 6GUPF signals the BF-3 to accelerate the flow processing to hardware, optimizing packet handling for subsequent transmissions (step 4). The packet is then sent back through the N3/N6 interface, leveraging the hardware-accelerated processing (steps 5-6).

Figure 7.3 illustrates the DPU-based 6GUPF QoS architecture, where IP flows are first parsed and processed by the DOCA QoS Engine. This engine performs classification, policy enforcement, load balancing, and scheduling using the DPU

offload pipeline. Each uplink/downlink IP flow is encapsulated into GTP-U PDUs and associated with a QoS Flow Identifier (QFI). These QFI flows are mapped to Data Radio Bearers (DRBs) via the gNodeB's Service Data Adaptation Protocol (SDAP) layer. The DRB mapping ensures that the QoS levels defined in the core are considered at the radio interface. Notably, DOCA handles QoS classification, enforcement, and flow state in hardware, allowing the system to scale with minimal CPU intervention.

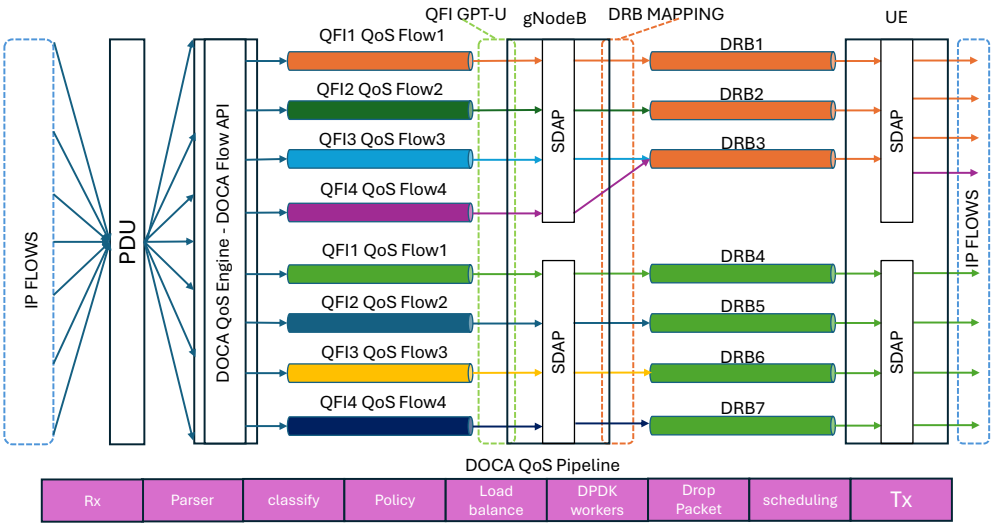


Figure 7.3: Flow-Based QoS at UPF-Downlink, gNodeB, and UE

7.1 Implementation of 6GUPF

To evaluate the proposed 6GUPF architecture, we implemented a realistic testbed on a Dell R760 server with an NVIDIA BF-3 DPU, integrating both control- and data-plane functions representative of 6G core deployments. Figure 7.4 illustrates this setup, which connects a base station emulator to a data center traffic sink. Traffic is generated from the Base Station, which represents the 5G radio access network (RAN), and is forwarded via the N3 interface as GTP-U encapsulated packets. These packets are injected using an emulated VIAVI/TRex traffic generator, which emulates high-throughput and latency-sensitive mobile traffic over L2 links.

At the core of the testbed lies a Dell R760 server equipped with a BF-3 DPU, responsible for the hardware offloading of UPF packet processing workloads. Inside the BF-3 DPU, the DOCA Flow API manages programmable pipelines for flow

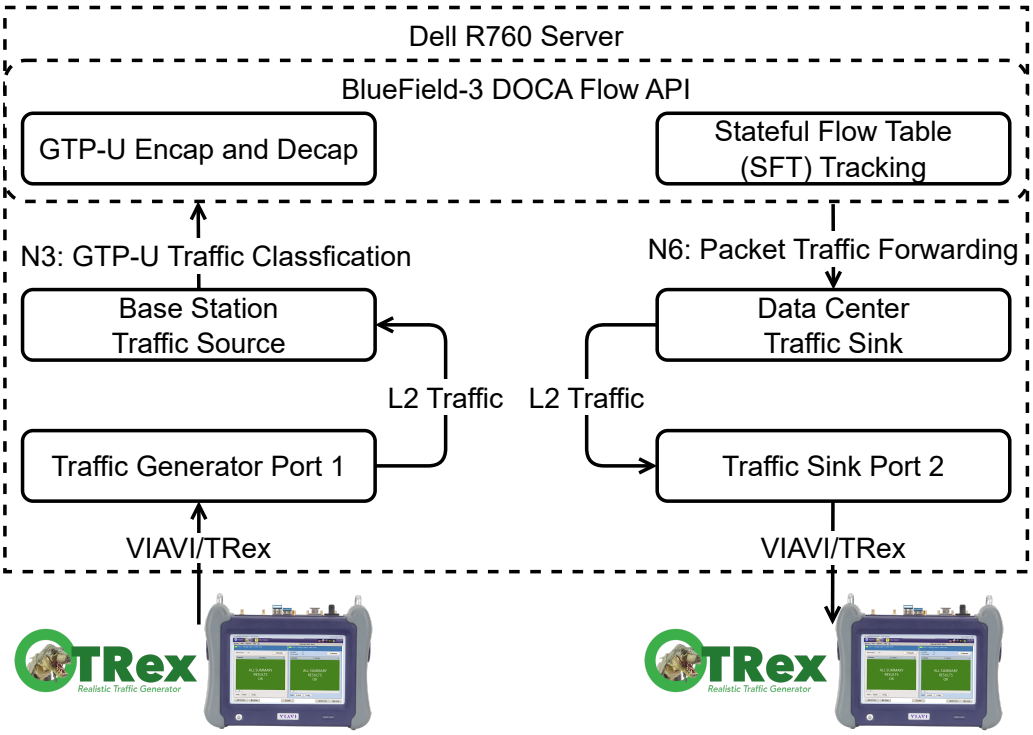


Figure 7.4: DPU-accelerated 6GUPF testbed emulating GTP-U traffic for 6G core evaluation.

parsing, GTP-U header encapsulation/decapsulation, and traffic classification. A key component of the system is the SFT for flow tracking, which maintains per-flow metadata to enable intelligent state-aware forwarding at line rate.

Following classification and processing, the decapsulated packets are forwarded over the N6 interface as standard IP traffic to the Data Center, acting as the traffic sink. Traffic sink validation and replay are facilitated via another port of the DPU, which connects to the VIAVI/TRex traffic generators. Traffic is generated with a Gaussian mixture of 70% elephant flows (long-lived) and 30% burst mice flows, calibrated across MTUs from 128B to 1500B. The traffic generator produces a mix of GTP-U and IP packets to simulate real-world 6G network scenarios.

This architecture encapsulates a production-realistic deployment scenario where the UPF is embedded directly within the data path of the base station. Packet processing is conducted based on QoS policies, which are dynamically assigned to packets according to their classification and network conditions. High Priority is assigned to packets requiring minimal delay for network applications. Medium Pri-

ority is assigned to packets that require high throughput but are not latency-sensitive. Low Priority is assigned to packets that are less sensitive to latency and throughput.

The DOCA Flow API manages the ingress pipeline with five stages (as defined above, namely ingress classification, QoS assignment, GTP processing, L2 forwarding, and final forwarding. After ingress classification, the DOCA flow checks for QoS. The encapsulation process involves defining programmable tables for TEID-based matching and forwarding. As shown in Listing 7.1, a teid_as_key table is configured to match on GTP TEIDs and apply forwarding actions accordingly.

Listing 7.1: TEID-based forwarding table in encapsulation pipeline

```
table teid_as_key {
    key = {
        headers.gtpv1.teid : exact;
    }
    actions = {
        NoAction;
        drop;
        forward;
    }
    size = 128;
    default_action = forward(3);
    const entries = {
        (32w0x01) : forward(1);
        (32w0x02) : forward(2);
    }
}
```

The complete DOCA Flow configuration scripts, including TEID-based match-action pipelines and rule installation routines, are publicly available in our repository [1]. This implementation demonstrates that offloading GTP-U processing into the DPU pipeline significantly reduces CPU overhead, lowers packet-processing la-

[1] 6GUPF implementation: `https://github.com/engranaabubakar/6GUPF`

tency, and enables wire-speed throughput while preserving programmability through a software-managed control plane. This implementation forms the foundation for the performance evaluation that we present in Sec. 7.2.

7.2 6GUPF Results

This section presents the performance evaluation of the proposed 6GUPF architecture under varying traffic loads and packet sizes. Key metrics, including throughput, server CPU utilization, latency, and packet drop rate, were measured to demonstrate the superiority of the DPU-based implementation over DPDK-UPF, P4-UPF, and our 6GUPF.

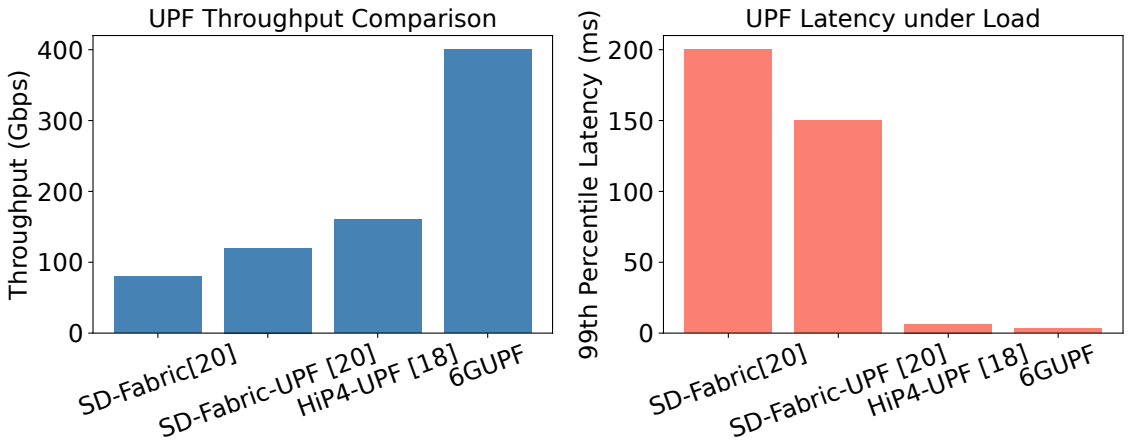


Figure 7.5: Comparative analysis of throughput and latency among software-based and DPU-accelerated UPF implementations.

Figure 7.5 shows the comparative performance of multiple UPF designs, including SD-Fabric [175], HiP4-UPF [153], and our proposed 6GUPF. The DPU-based 6GUPF demonstrates a fourfold increase in throughput, reaching up to 400 Gbps, compared to 80–160 Gbps in software UPF variants. The 6GUPF achieves a drastic reduction in the 99th percentile latency, dropping from 200 ms in conventional platforms to just 0.2 ms.

Figure 7.6 shows the performance benefits of DPU offloading in UPF processing. On the left, CPU utilization rises sharply for the DPDK-based UPF because per-packet operations (parsing, TEID lookup, GTP-U encap/decap) saturate the cores at high PPS. Offloading these primitives to the DPU removes the CPU bottleneck

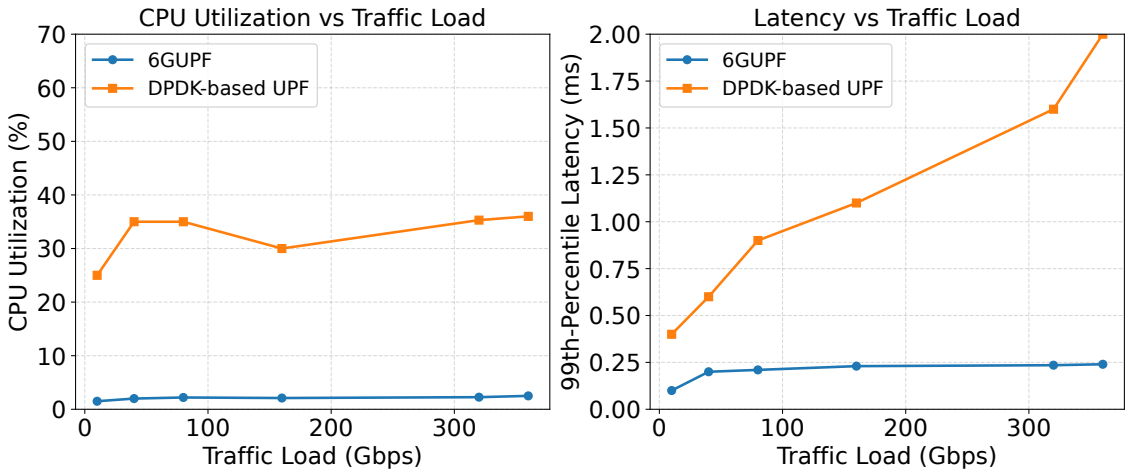


Figure 7.6: CPU utilization and latency analysis of DPU-based UPF versus baseline DPDK-based UPF under different load conditions.

and shortens the processing pipeline, which directly lowers the 99th-percentile latency shown on the right. At a 380 Gbit/s traffic load, CPU utilization is under 5% for the DPU-based design, compared to 50% for the DPDK-based UPF implementation.

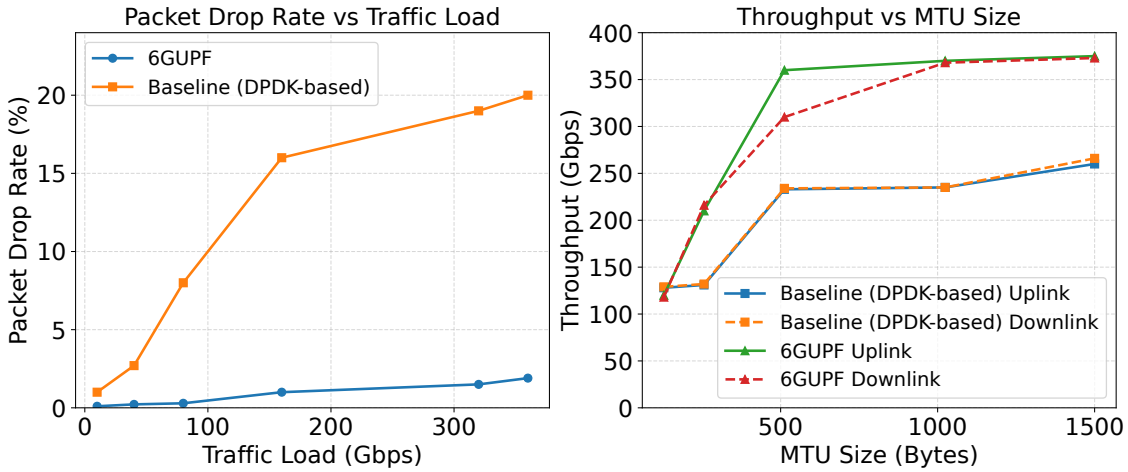


Figure 7.7: Performance comparison of the proposed 6GUPF against a DPDK-based baseline.

Figure 7.7 shows that the DPU-based UPF has a significantly lower packet drop rate than the NIC-based implementation, even under high traffic loads. The packet drop rate was 2.29% with 380 Gbit/s load using the DPU, while the NIC-based system suffered a drop rate of 10%. At the maximum MTU size of 1500 bytes, 6GUPF achieved a throughput of 380 Gbit/s, compared to the baseline DPDK-UPF's throughput of 234 Gbit/s. For an MTU size of 256 bytes, 6GUPF throughput reached 110 Gbit/s, significantly surpassing the baseline DPDK-UPF. For smaller

MTU sizes, such as 512 bytes and 256 bytes, 6GUPF demonstrated sustained performance.

The results clearly illustrate the advantages of the DPU-based UPF architecture in achieving higher throughput, reduced server CPU utilization, lower latency, and improved reliability. These results indicate not only raw performance improvements but also practical scalability for next-generation mobile networks. For instance, reducing CPU utilization by over 50% directly translates into cost savings in cloud-native deployments, as fewer servers are required to handle equivalent traffic loads. Similarly, the packet drop reduction (2.29% vs. 20%) ensures service continuity under congestion, which is crucial for applications such as URLLC in 6G. Compared to P4-switch UPFs that suffer from limited flow table memory, our DPU-based design supports large-scale flow states without sacrificing wire-speed processing. This highlights the viability of DPUs as a future-proof platform for UPF.

7.2.1 Discussion

The results confirm that CPU-based UPFs quickly saturate, explaining the high drop rates of NIC-only (DPDK-based) implementations even at modest loads. However, the DPU sustains near-zero losses by executing encapsulation and forwarding in dedicated hardware pipes.

The observed $\sim$200–300 μs latency of the 6GUPF under high load reflects not only GTP-U processing in hardware but also queuing and flow-handling overheads currently executed on the DPUs ARM cores due to SDK limitations. By comparison, the DPDK-based UPF exhibits $\sim$1 ms delay caused by context switches and cache contention on host CPUs. While DPU latency is higher than bare-metal switching, it remains an order of magnitude lower than CPU-based solutions while sustaining 200–400 Gbps throughput.

Compared with DPDK-UPFs and P4 switches, the DPU provides a unique balance of programmability, hardware acceleration, and scalable stateful flow tracking. Future SDK releases that enable further migration of queuing and flow management into hardware are expected to reduce latency further, making DPUs an increasingly

attractive platform for next-generation UPFs. These insights suggest that although DPUs cannot yet match bare-metal forwarding latency, their combination of programmability, scalability, and line-rate performance makes them strong candidates for 6G UPF deployments. A limitation of our current testbed is that it remains server-attached, and fully edge-integrated UPFs will be explored in future work. We also plan to extend the evaluation to multi-DPU clusters to investigate horizontal scalability and to integrate inline security functions, such as DPI and DDoS mitigation, into the UPF pipeline.

This work presented a DPU-based programmable UPF for 6G networks, designed to overcome the performance limitations of traditional software-based implementations. By offloading computationally intensive tasks such as QoS management, GTP tunnel handling, and packet encapsulation/decapsulation into the hardware pipeline of BlueField-3 using the DOCA Flow API, the proposed 6GUPF achieves substantial improvements in throughput, latency, and scalability. Our evaluation demonstrates that the system can sustain a throughput of up to 400 Gbps with significantly reduced latency, while maintaining programmability and efficient flow-based QoS enforcement. These results highlight the potential of DPUs as a practical foundation for future mobile cores, striking a balance between wire-speed performance and flexibility. Future work will extend this approach to fully edge-integrated UPFs, explore multi-DPU scalability, and integrate AI-assisted QoS enforcement and inline security functions, paving the way for autonomous and intelligent 6G core deployments.

7.3 Chapter Summary

This chapter presented a DPU-based programmable UPF for 6G networks, designed to overcome the performance limitations of traditional software-based implementations. By offloading computationally intensive tasks such as QoS management, GTP tunnel handling, and packet encapsulation/decapsulation into the hardware pipeline of BlueField-3 using the DOCA Flow API, the proposed 6GUPF achieves substantial improvements in throughput, latency, and scalability. Our evaluation demonstrates

that the system can sustain a throughput of up to 400 Gbps with significantly reduced latency, while maintaining programmability and efficient flow-based QoS enforcement. These results highlight the potential of DPUs as a practical foundation for future mobile cores, striking a balance between wire-speed performance and flexibility. Future work will extend this approach to fully edge-integrated UPFs, explore multi-DPU scalability, and integrate AI-assisted QoS enforcement and inline security functions, paving the way for autonomous and intelligent 6G core deployments.

Chapter 8

Conclusion

This dissertation investigated the role of Data Processing Units (DPUs) as a unifying platform for accelerating, securing, and scaling next-generation networked systems. Motivated by the growing mismatch between line-rate network capabilities and CPU-bound processing models, the thesis examined how programmable SmartNICs can shift critical functionality closer to the data path, reducing latency while improving system efficiency and robustness. The work spans three domains5G user-plane offloading, high-performance computing (HPC), and network securitybut is unified by a common architectural perspective: DPUs enable an *observedecideenforce* loop directly in the network data plane.

With respect to **RQ1**, the thesis showed that security and networking functions with high packet-rate requirementssuch as GTP encapsulation/decapsulation, QoS enforcement, flow tracking, and signature-based DDoS mitigationbenefit the most from SmartNIC/DPU offloading. Offloading these functions alleviates host CPU bottlenecks while introducing new design considerations related to memory placement, pipeline depth, and controldata plane coordination.

Addressing **RQ2**, the evaluation of FTG-based and ensemble GNN models demonstrated that graph-based DDoS detection is particularly advantageous in scenarios characterized by multi-flow correlations, fan-in/fan-out patterns, and coordinated low-rate attacks. At the same time, the results clarify the limiting cases where single-flow or time-series detectors achieve comparable performance, especially when datasets are dominated by strong per-flow temporal features.

In response to **RQ3**, the thesis confirmed that programmable DPUs can sustain wire-speed DDoS mitigation using stateful tracking and DPI-based enforcement, even under high attack rates. Experimental results on NVIDIA BlueField platforms showed that mitigation can be executed with bounded latency and without degrading forwarding performance, validating DPUs as viable enforcement points in high-speed networks.

Regarding **RQ4**, the proposed SmartNIC-based authentication framework demonstrated strong resilience against masquerading, desynchronization, and physical attacks, while explicitly showing how SmartNIC-based execution reduces trust dependencies on host software and preserves deployability in heterogeneous swarm and edge environments. By combining PUF-based primitives with DPU-assisted processing, the protocol achieves strong security guaranteesincluding resistance to masquerading, desynchronization, and physical attackswhile preserving device anonymity and untraceability.

The investigation of **RQ5** provided a detailed analysis of BlueField-3 architectural advances. Measurements of RDMA latency, bandwidth, and message rate showed how PCIe Gen5, DDR5 memory, and the DPA contribute to performance gains, while also highlighting the gap between theoretical models and practical measurements under real system constraints.

Finally, **RQ6** explored the offloading of GTP encapsulation and decapsulation on BlueField-3, showing that protocol processing traditionally handled by host CPUs can be efficiently migrated to the SmartNIC, reducing host overhead and enabling scalable 5G and beyond-5G deployments.

Beyond individual results, the thesis highlights important practical challenges. These include inference-time constraints for machine learning models, sensitivity to system-level tuning, and trade-offs between Arm cores and DPA execution. These findings emphasize that effective DPU utilization requires careful co-design across hardware, software, and workload characteristics.

Taken together, the results of this thesis provide clear answers to the research questions posed in Chapter 1. The analysis of SmartNIC and DPU offloading demon-

strates that functions requiring deterministic, high-rate processingsuch as DDoS mitigation, user-plane forwarding, and RDMA data movementbenefit most from hardware acceleration. Graph-based traffic representations and hierarchical GNN models are shown to outperform single-flow detectors in scenarios with strong inter-flow dependencies, while exhibiting identifiable inference-time limits. Wire-speed mitigation using stateful tracking and DPI is validated on programmable DPUs under high attack rates with bounded latency. Furthermore, the DPUAUT protocol confirms that authentication can be accelerated at the SmartNIC level without compromising deployability and isolation. Finally, empirical benchmarking quantifies the architectural gains of BlueField-3 and validates theoretical RDMA models, while UPF offload experiments demonstrate practical benefits for 5G/6G user-plane processing.

In conclusion, this work establishes DPUs as a key architectural element for future 6G-era infrastructures. By integrating networking, computing, and security capabilities within a single programmable platform, DPUs enable systems that are not only faster and more scalable, but also more resilient by design. Future research should extend these results to large-scale deployments, explore SmartNIC-aware cloud abstractions, and investigate tighter integration between DPUs and AI-driven network control frameworks.

References

[1] Internet Crime Complaint Center IC3, *FBI Internet Crime Report 2021*, 2021.

[2] Kaspersky Lab ZAO, *DDoS attacks in Q2 2022*, Aug. 2022.

[3] E. F. Kfoury, S. Choueiri, A. Mazloum, A. AlSabeh, J. Gomez, and J. Crichigno, "A comprehensive survey on smartnics: Architectures, development models, applications, and research directions," *IEEE Access*, vol. 12, pp. 107 297–107 336, 2024. DOI: 10.1109/ACCESS.2024.3437203.

[4] T. Döring, H. Stubbe, and K. Holzinger, "Smartnics: Current trends in research and industry," *Network*, vol. 19, 2021.

[5] F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, and G. Monfardini, "The graph neural network model," *IEEE transactions on neural networks*, vol. 20, no. 1, pp. 61–80, 2008.

[6] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, and S. Y. Philip, "A comprehensive survey on graph neural networks," *IEEE transactions on neural networks and learning systems*, vol. 32, no. 1, pp. 4–24, 2020.

[7] I. R. Ward, J. Joyner, C. Lickfold, Y. Guo, and M. Bennamoun, "A practical tutorial on graph neural networks," *ACM Computing Surveys (CSUR)*, vol. 54, no. 10s, pp. 1–35, 2022.

[8] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," *arXiv preprint arXiv:1609.02907*, 2016.

[9] P. Velikovi, G. Cucurull, A. Casanova, A. Romero, P. Lio, and Y. Bengio, "Graph attention networks," *arXiv preprint arXiv:1710.10903*, 2017.

[10] A. Mohammed and R. Kora, "A comprehensive review on ensemble deep learning: Opportunities and challenges," *Journal of King Saud University-Computer and Information Sciences*, 2023.

[11] L. Von Krannichfeldt, Y. Wang, and G. Hug, "Online ensemble learning for load forecasting," *IEEE Transactions on Power Systems*, vol. 36, no. 1, pp. 545–548, 2020.

[12] I. Ortega-Fernandez, M. Sestelo, J. C. Burguillo, and C. Pinon-Blanco, "Network intrusion detection system for ddos attacks in ics using deep autoencoders," *Wireless Networks*, pp. 1–17, 2023.

[13] K. Rusek, J. Suárez-Varela, A. Mestres, P. Barlet-Ros, and A. Cabellos-Aparicio, "Unveiling the potential of graph neural networks for network modeling and optimization in sdn," in *Proceedings of the 2019 ACM Symposium on SDN Research*, 2019, pp. 140–151.

[14] M. Khan and L. Ghafoor, "Adversarial machine learning in the context of network security: Challenges and solutions," *Journal of Computational Intelligence and Robotics*, vol. 4, no. 1, pp. 51–63, 2024.

[15] Y. A. Abid, J. Wu, G. Xu, S. Fu, and M. Waqas, "Multi-level deep neural network for distributed denial-of-service attack detection and classification in software-defined networking supported internet of things networks," *IEEE Internet of Things Journal*, 2024.

[16] S. Ahmed, Z. A. Khan, S. M. Mohsin, *et al.*, "Effective and efficient ddos attack detection using deep learning algorithm, multi-layer perceptron," *Future Internet*, vol. 15, no. 2, p. 76, 2023.

[17] A. Y.-P. Lee, M. I.-C. Wang, C.-H. Hung, and C. H.-P. Wen, "Ps-ips: Deploying intrusion prevention system with machine learning on programmable switch," *Future Generation Computer Systems*, vol. 152, pp. 333–342, 2024.

[18] C. Zheng, M. Zang, X. Hong, *et al.*, "Automating in-network machine learning," in *arXiv*, 2022.

[19] Q. Gong, P. DeMar, and M. Altunay, "Thundersecure: Deploying real-time intrusion detection for 100g research networks by leveraging stream-based features and one-class classification network," *International Journal of Information Security*, vol. 21, no. 4, pp. 799–812, 2022.

[20] M. Ceka, V. Havlena, L. Holík, *et al.*, "Deep packet inspection in fpgas via approximate nondeterministic automata," in *2019 IEEE 27th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, IEEE, 2019, pp. 109–117.

[21] P. Orosz, T. Tóthfalusi, and P. Varga, "Fpga-assisted dpi systems: 100 gbit/s and beyond," *IEEE Communications Surveys & Tutorials*, vol. 21, no. 2, pp. 2015–2040, 2018.

[22] Z. Zhao, H. Sadok, N. Atre, J. C. Hoe, V. Sekar, and J. Sherry, "Achieving 100gbps intrusion prevention on a single server," in *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*, 2020, pp. 1083–1100.

[23] J. Chen, X. Zhang, T. Wang, *et al.*, "Fidas: Fortifying the cloud via comprehensive fpga-based offloading for intrusion detection: Industrial product," in *Proceedings of the 49th Annual International Symposium on Computer Architecture*, 2022, pp. 1029–1041.

[24] T. Park, J. Nam, S. H. Na, J. Chung, and S. Shin, "Reinhardt: Real-time reconfigurable hardware architecture for regular expression matching in dpi," in *Proceedings of the 37th Annual Computer Security Applications Conference*, 2021, pp. 620–633.

[25] J. Su, S. Chen, B. Han, C. Xu, and X. Wang, "A 60gbps dpi prototype based on memory-centric fpga," in *Proceedings of the 2016 ACM SIGCOMM Conference*, 2016, pp. 627–628.

[26] T. N. Thinh, T. T. Hieu, S. Kittitornkun, *et al.*, "A fpga-based deep packet inspection engine for network intrusion detection system," in *2012 9th Interna-*

tional Conference on Electrical Engineering/Electronics, Computer, Telecommunications and Information Technology, IEEE, 2012, pp. 1–4.

[27] P. Kumari and A. K. Jain, "A comprehensive study of ddos attacks over iot network and their countermeasures," *Computers & Security*, p. 103 096, 2023.

[28] F. Musumeci, A. Fidanci, F. Paolucci, F. Cugini, and M. Tornatore, "Machine-learning-enabled DDoS attacks detection in P4 programmable networks," *Journal of Network and Systems Management*, vol. 30, pp. 1–27, 2022.

[29] A. Sapio, M. Canini, C.-Y. Ho, *et al.*, "Scaling distributed machine learning with {in-network} aggregation," in *18th USENIX Symposium on Networked Systems Design and Implementation (NSDI 21)*, 2021, pp. 785–808.

[30] M. Wu, H. Matsutani, and M. Kondo, "Onlad-ids: Onlad-based intrusion detection system using smartnic," in *2022 IEEE 24th Int Conf on High Performance Computing & Communications;*, 2022, pp. 546–553. DOI: 10.1109/HPCC-DSS-SmartCity-DependSys57074.2022.00100.

[31] S.-Y. Wang and J.-C. Chang, "Design and implementation of an intrusion detection system by using extended bpf in the linux kernel," *Journal of Network and Computer Applications*, vol. 198, p. 103 283, 2022.

[32] T. Kim and W. Pak, "Real-time network intrusion detection using deferred decision and hybrid classifier," *Future Generation Computer Systems*, vol. 132, pp. 51–66, 2022.

[33] L. Barsellotti, F. Alhamed, J. J. V. Olmos, F. Paolucci, P. Castoldi, and F. Cugini, "Introducing data processing units (dpu) at the edge," in *2022 International Conference on Computer Communications and Networks (ICCCN)*, IEEE, 2022, pp. 1–6.

[34] G. Zhang, Z. Li, J. Huang, *et al.*, "Efraudcom: An e-commerce fraud detection system via competitive graph neural networks," *ACM Transactions on Information Systems (TOIS)*, vol. 40, no. 3, pp. 1–29, 2022.

[35] E. Owusu, M. Rahouti, S. K. Jagatheesaperumal, *et al.*, "Online network dos/ddos detection: Sampling, change point detection, and machine learning methods," *IEEE Communications Surveys & Tutorials*, 2024.

[36] M. Almehdhar, A. Albaseer, M. A. Khan, *et al.*, "Deep learning in the fast lane: A survey on advanced intrusion detection systems for intelligent vehicle networks," *IEEE Open Journal of Vehicular Technology*, 2024.

[37] S. A. Abdulkareem, C. H. Foh, M. Shojafar, F. Carrez, and K. Moessner, "Network intrusion detection: An iot and non iot-related survey," *IEEE Access*, 2024.

[38] D. Zhang and S. Wang, "Optimization of traditional snort intrusion detection system," in *IOP Conference Series: Materials Science and Engineering*, IOP Publishing, vol. 569, 2019, p. 042 041.

[39] Q. Hu, M. R. Asghar, and N. Brownlee, "Evaluating network intrusion detection systems for high-speed networks," in *2017 27th International Telecommunication Networks and Applications Conference (ITNAC)*, IEEE, 2017, pp. 1–6.

[40] Q. Hu, S.-Y. Yu, and M. R. Asghar, "Analysing performance issues of open-source intrusion detection systems in high-speed networks," *Journal of Information Security and Applications*, vol. 51, p. 102 426, 2020.

[41] S. Alcock and R. Nelson, "Libprotoident: Traffic classification using lightweight packet inspection," Technical report, University of Waikato, Tech. Rep., 2012.

[42] L. Deri, M. Martinelli, T. Bujlow, and A. Cardigliano, "Ndpi: Open-source high-speed deep packet inspection," in *2014 International Wireless Communications and Mobile Computing Conference (IWCMC)*, IEEE, 2014, pp. 617–622.

[43] T. Bujlow, V. Carela-Español, and P. Barlet-Ros, "Independent comparison of popular dpi tools for traffic classification," *Computer Networks*, vol. 76, pp. 75–89, 2015.

[44] M. Çelebi, A. Özbilen, and U. Yavanolu, "A comprehensive survey on deep packet inspection for advanced network traffic analysis: Issues and challenges," *Nide Ömer Halisdemir Üniversitesi Mühendislik Bilimleri Dergisi*, vol. 12, no. 1, pp. 1–29, 2023.

[45] L. De Marinis, E. Paolini, R. Abu Bakar, F. Cugini, and F. Paolucci, "Cascaded look up table distillation of P4 deep neural network switches," in *Globecom 2023 - 2023 IEEE Global Communications Conference: Next-Generation Networking and Internet*, 2023, pp. 2112–2117.

[46] F. Cugini, D. Scano, A. Giorgetti, *et al.*, "Telemetry and ai-based security p4 applications for optical networks," *Journal of Optical Communications and Networking*, vol. 15, no. 1, A1–A10, 2023.

[47] R. Doriguzzi-Corin, S. Millar, S. Scott-Hayward, J. Martínez-del-Rincón, and D. Siracusa, "Lucid: A practical, lightweight deep learning solution for ddos attack detection," *IEEE Transactions on Network and Service Management*, vol. 17, no. 2, pp. 876–889, 2020. DOI: 10.1109/TNSM.2020.2971776.

[48] H. Ko, I. Praca, and S. G. Choi, "Anomaly detection analysis based on correlation of features in graph neural network," *Multimedia Tools and Applications*, pp. 1–15, 2023.

[49] V.-A. Nguyen, D. Q. Nguyen, V. Nguyen, T. Le, Q. H. Tran, and D. Phung, "Regvd: Revisiting graph neural networks for vulnerability detection," in *Proceedings of the ACM/IEEE 44th International Conference on Software Engineering: Companion Proceedings*, 2022, pp. 178–182.

[50] C. Lin, Y. Xu, Y. Fang, and Z. Liu, "Vuleye: A novel graph neural network vulnerability detection approach for php application," *Applied Sciences*, vol. 13, no. 2, p. 825, 2023.

[51] Y. Zhang, C. Yang, K. Huang, and Y. Li, "Intrusion detection of industrial internet-of-things based on reconstructed graph neural networks," *IEEE Transactions on Network Science and Engineering*, 2022.

[52] C. Liu, B. Li, J. Zhao, Z. Zhen, X. Liu, and Q. Zhang, "Fewm-hgcl: Few-shot malware variants detection via heterogeneous graph contrastive learning," *IEEE Transactions on Dependable and Secure Computing*, 2022.

[53] H. Wang, H. Xu, L. Huang, and Y. Zhai, "Fast and accurate traffic measurement with hierarchical filtering," *IEEE Transactions on Parallel and Distributed Systems*, vol. 31, no. 10, pp. 2360–2374, 2020.

[54] S. Günnemann, "Graph neural networks: Adversarial robustness," *Graph Neural Networks: Foundations, Frontiers, and Applications*, pp. 149–176, 2022.

[55] D. Pujol Perich, J. R. Suárez-Varela Maciá, A. Cabellos Aparicio, and P. Barlet Ros, "Unveiling the potential of graph neural networks for robust intrusion detection," in *3rd International Workshop on AI in Networks and Distributed Systems*, 2021, pp. 1–7.

[56] Y. Li, R. Li, Z. Zhou, *et al.*, "GraphDDoS: Effective DDoS Attack Detection Using Graph Neural Networks," in *2022 IEEE 25th International Conference on Computer Supported Cooperative Work in Design (CSCWD)*, 2022, pp. 1275–1280. DOI: 10.1109/CSCWD54268.2022.9776097.

[57] X. Song, J. Li, Q. Lei, W. Zhao, Y. Chen, and A. Mian, "Bi-clkt: Bi-graph contrastive learning based knowledge tracing," *Knowledge-Based Systems*, vol. 241, p. 108 274, 2022.

[58] W. Guo, H. Qiu, Z. Liu, J. Zhu, and Q. Wang, "GLD-Net: Deep Learning to Detect DDoS Attack via Topological and Traffic Feature Fusion," *Computational Intelligence and Neuroscience*, vol. 2022, 2019.

[59] S. Yang, S. Verma, B. Cai, *et al.*, "Variational co-embedding learning for attributed network clustering," *Knowledge-Based Systems*, vol. 270, p. 110 530, 2023.

[60] W. W. Lo, S. Layeghy, M. Sarhan, M. Gallagher, and M. Portmann, "E-graphsage: A graph neural network based intrusion detection system for iot," in *NOMS 2022-2022 IEEE/IFIP Network Operations and Management Symposium*, IEEE, 2022, pp. 1–9.

[61] W. Hamilton, Z. Ying, and J. Leskovec, "Inductive representation learning on large graphs," *Advances in neural information processing systems*, vol. 30, 2017.

[62] S. Nagaraja, P. Mittal, C.-Y. Hong, M. Caesar, and N. Borisov, "{Botgrep}: Finding {p2p} bots with structured graph analysis," in *19th USENIX Security Symposium (USENIX Security 10)*, 2010.

[63] M. Alshammari, J. Stavrakakis, and M. Takatsuka, "A parameter-free graph reduction for spectral clustering and spectralnet," *Array*, vol. 15, p. 100 192, 2022.

[64] Q. Xiao, J. Liu, Q. Wang, Z. Jiang, X. Wang, and Y. Yao, "Towards network anomaly detection using graph embedding," in *Computational Science–ICCS 2020: 20th International Conference, Amsterdam, The Netherlands, June 3–5, 2020, Proceedings, Part IV 20*, Springer, 2020, pp. 156–169.

[65] W. Hamilton, Z. Ying, and J. Leskovec, "Inductive representation learning on large graphs," *Advances in neural information processing systems*, vol. 30, 2017.

[66] E. Caville, W. W. Lo, S. Layeghy, and M. Portmann, "Anomal-e: A self-supervised network intrusion detection system based on graph neural networks," *Knowledge-Based Systems*, vol. 258, p. 110 030, 2022.

[67] S. Wu, F. Sun, W. Zhang, X. Xie, and B. Cui, "Graph neural networks in recommender systems: A survey," *ACM Computing Surveys*, vol. 55, no. 5, pp. 1–37, 2022.

[68] X. Hou, P. Qi, G. Wang, *et al.*, "Graph ensemble learning over multiple dependency trees for aspect-level sentiment classification," *arXiv preprint arXiv:2103.11794*, 2021.

[69] W. Wei, M. Qiao, and D. Jadav, "Gnn-ensemble: Towards random decision graph neural networks," *arXiv preprint arXiv:2303.11376*, 2023.

[70] S. Barai and Y. Reich, "Ensemble modelling or selecting the best model: Many could be better than one," *Ai Edam*, vol. 13, no. 5, pp. 377–386, 1999.

[71] Y. Wang, J. Li, W. Zhao, *et al.*, "N-stgat: Spatio-temporal graph neural network based network intrusion detection for near-earth remote sensing," *Remote Sensing*, vol. 15, no. 14, 2023, ISSN: 2072-4292. DOI: 10.3390/rs15143611. [Online]. Available: https://www.mdpi.com/2072-4292/15/14/3611.

[72] Q. Qi and P. H. Kwok, "Traffic4cast 2020–graph ensemble net and the importance of feature and loss function design for traffic prediction," *arXiv preprint arXiv:2012.02115*, 2020.

[73] H. Zhu and J. Lu, "Graph-based intrusion detection system using general behavior learning," in *GLOBECOM 2022-2022 IEEE Global Communications Conference*, IEEE, 2022, pp. 2621–2626.

[74] B. Esmaeili, A. Azmoodeh, A. Dehghantanha, G. Srivastava, H. Karimipour, and J. C.-W. Lin, "A gnn-based adversarial internet of things malware detection framework for critical infrastructure: Studying gafgyt, mirai and tsunami campaigns," *IEEE Internet of Things Journal*, 2023.

[75] V. O. Nyangaresi, "Privacy preserving three-factor authentication protocol for secure message forwarding in wireless body area networks," *Ad Hoc Networks*, vol. 142, p. 103 117, 2023.

[76] L. Barsellotti, F. Alhamed, J. J. Vegas Olmos, F. Paolucci, P. Castoldi, and F. Cugini, "Introducing data processing units (dpu) at the edge [invited]," in *2022 International Conference on Computer Communications and Networks (ICCCN)*, 2022, pp. 1–6. DOI: 10.1109/ICCCN54977.2022.9868927.

[77] K. R. Ozyilmaz and A. Yurdakul, "Designing a blockchain-based iot with ethereum, swarm, and lora: The software solution to create high availability with minimal security risks," *IEEE Consumer Electronics Magazine*, vol. 8, no. 2, pp. 28–34, 2019.

[78] A. Mullai and K. Mani, "Enhancing the security in rsa and elliptic curve cryptography based on addition chain using simplified swarm optimization

and particle swarm optimization for mobile devices," *International Journal of Information Technology*, vol. 13, pp. 551–564, 2021.

[79] L. Buttyan and J.-P. Hubaux, *Security and cooperation in wireless networks: thwarting malicious and selfish behavior in the age of ubiquitous computing*. Cambridge University Press, 2007.

[80] L. Agilandeeswari, S. Paliwal, A. Chandrakar, and M. Prabukumar, "A new lightweight conditional privacy preserving authentication and key–agreement protocol in social internet of things for vehicle to smart grid networks," *Multimedia Tools and Applications*, vol. 81, no. 19, pp. 27 683–27 710, 2022.

[81] K. Xue, W. Meng, S. Li, D. S. Wei, H. Zhou, and N. Yu, "A secure and efficient access and handover authentication protocol for internet of things in space information networks," *IEEE Internet of Things Journal*, vol. 6, no. 3, pp. 5485–5499, 2019.

[82] P. Garcia Lopez, A. Montresor, D. Epema, *et al.*, *Edge-centric computing: Vision and challenges*, 2015.

[83] Z. Liu, L. Wan, J. Guo, *et al.*, "Ppru: A privacy-preserving reputation updating scheme for cloud-assisted vehicular networks," *IEEE Transactions on Vehicular Technology*, pp. 1–16, 2023. DOI: 10.1109/TVT.2023.3340723.

[84] R.-F. Liao, H. Wen, J. Wu, *et al.*, "Security enhancement for mobile edge computing through physical layer authentication," *IEEE Access*, vol. 7, pp. 116 390–116 401, 2019.

[85] X. Jia, D. He, N. Kumar, and K.-K. R. Choo, "A provably secure and efficient identity-based anonymous authentication scheme for mobile edge computing," *IEEE Systems Journal*, vol. 14, no. 1, pp. 560–571, 2019.

[86] B. D. Deebak, F. Al-Turjman, and L. Mostarda, "Seamless secure anonymous authentication for cloud-based mobile edge computing," *Computers & Electrical Engineering*, vol. 87, p. 106 782, 2020.

[87] S. Hussain, K. Mahmood, M. K. Khan, C.-M. Chen, B. A. Alzahrani, and S. A. Chaudhry, "Designing secure and lightweight user access to drone for smart city surveillance," *Computer Standards & Interfaces*, vol. 80, p. 103 566, 2022, ISSN: 0920-5489. DOI: 10.1016/j.csi.2021.103566.

[88] Y. Li, Q. Cheng, X. Liu, and X. Li, "A secure anonymous identity-based scheme in new authentication architecture for mobile edge computing," *IEEE Systems Journal*, vol. 15, no. 1, pp. 935–946, 2020.

[89] K. Kaur, S. Garg, G. Kaddoum, M. Guizani, and D. N. K. Jayakody, "A lightweight and privacy-preserving authentication protocol for mobile edge computing," in *2019 IEEE Global Communications Conference (GLOBECOM)*, IEEE, 2019, pp. 1–6.

[90] C. Wang, Y. Zhang, X. Chen, K. Liang, and Z. Wang, "Sdn-based handover authentication scheme for mobile edge computing in cyber-physical systems," *IEEE Internet of Things Journal*, vol. 6, no. 5, pp. 8692–8701, 2019.

[91] Z. Wang, Y. Zhou, Z. Qiao, *et al.*, "An anonymous and revocable authentication protocol for vehicle-to-vehicle communications," *IEEE Internet of Things Journal*, vol. 10, no. 6, pp. 5114–5127, 2022.

[92] P. Gope, Y. Gheraibia, S. Kabir, and B. Sikdar, "A secure iot-based modern healthcare system with fault-tolerant decision making process," *IEEE Journal of Biomedical and Health Informatics*, vol. 25, no. 3, pp. 862–873, 2020.

[93] P. Gope and B. Sikdar, "An efficient privacy-preserving authenticated key agreement scheme for edge-assisted internet of drones," *IEEE Transactions on Vehicular Technology*, vol. 69, no. 11, pp. 13 621–13 630, 2020.

[94] S. Roy, D. Das, A. Mondal, M. H. Mahalat, B. Sen, and B. Sikdar, "Plake: Puf-based secure lightweight authentication and key exchange protocol for iot," *IEEE Internet of Things Journal*, vol. 10, no. 10, pp. 8547–8559, 2023. DOI: 10.1109/JIOT.2022.3202265.

[95] S.-W. Lee, M. Safkhani, Q. Le, *et al.*, "Designing secure puf-based authentication protocols for constrained environments," *Scientific Reports*, vol. 13, no. 1, p. 21 702, 2023.

[96] S. A. Chaudhry, K. Yahya, S. Garg, G. Kaddoum, M. M. Hassan, and Y. B. Zikria, "Las-sg: An elliptic curve-based lightweight authentication scheme for smart grid environments," *IEEE Transactions on Industrial Informatics*, vol. 19, no. 2, pp. 1504–1511, 2022.

[97] S. Zhang, Y. Liu, Y. Xiao, and R. He, "A trust based adaptive privacy preserving authentication scheme for vanets," *Vehicular Communications*, vol. 37, p. 100 516, 2022, ISSN: 2214-2096. DOI: https://doi.org/10.1016/j.vehcom.2022.100516. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2214209622000638.

[98] Y. Ma, X. Li, W. Shi, and Q. Cheng, "Stcla: An efficient certificateless authenticated key agreement scheme for the internet of vehicles," *IEEE Transactions on Vehicular Technology*, pp. 1–12, 2023. DOI: 10.1109/TVT.2023.3334034.

[99] Y. Liang, E. Luo, and Y. Liu, "Physically secure and conditional-privacy authenticated key agreement for vanets," *IEEE Transactions on Vehicular Technology*, 2023.

[100] F. Zerrouki, S. Ouchani, and H. Bouarfa, "Puf-based mutual authentication and session key establishment protocol for iot devices," *Journal of Ambient Intelligence and Humanized Computing*, vol. 14, no. 9, pp. 12 575–12 593, 2023.

[101] A. Irshad, M. Sher, H. F. Ahmad, B. A. Alzahrani, S. A. Chaudhry, and R. Kumar, "An improved multi-server authentication scheme for distributed mobile cloud computing services," *KSII Transactions on Internet and Information Systems (TIIS)*, vol. 10, no. 12, pp. 5529–5552, 2016.

[102] L. Xiong, D. Peng, T. Peng, and H. Liang, "An enhanced privacy-aware authentication scheme for distributed mobile cloud computing services.," *KSII Transactions on Internet & Information Systems*, vol. 11, no. 12, 2017.

[103] X. Yang, X. Huang, and J. K. Liu, "Efficient handover authentication with user anonymity and untraceability for mobile cloud computing," *Future Generation Computer Systems*, vol. 62, pp. 190–195, 2016.

[104] X. Li, T. Chen, Q. Cheng, and J. Ma, "An efficient and authenticated key establishment scheme based on fog computing for healthcare system," *Frontiers of Computer Science*, vol. 16, pp. 1–12, 2022.

[105] J. Li, W. Zhang, V. Dabra, K.-K. R. Choo, S. Kumari, and D. Hogrefe, "Aeppa: An anonymous, efficient and provably-secure privacy-preserving authentication protocol for mobile services in smart cities," *Journal of Network and Computer Applications*, vol. 134, pp. 52–61, 2019.

[106] E. Castelló Ferrer, "The blockchain: A new framework for robotic swarm systems," in *Proceedings of the Future Technologies Conference (FTC) 2018: Volume 2*, Springer, 2019, pp. 1037–1058.

[107] S. O. Ogundoyin, "An autonomous lightweight conditional privacy-preserving authentication scheme with provable security for vehicular ad-hoc networks," *International Journal of Computers and Applications*, vol. 42, no. 2, pp. 196–211, 2020.

[108] T. Nandy, M. Y. I. Idris, R. M. Noor, *et al.*, "An enhanced lightweight and secured authentication protocol for vehicular ad-hoc network," *Computer Communications*, vol. 177, pp. 57–76, 2021.

[109] C. Lin, D. He, X. Huang, N. Kumar, and K.-K. R. Choo, "Bcppa: A blockchain-based conditional privacy-preserving authentication protocol for vehicular ad hoc networks," *IEEE Transactions on Intelligent Transportation Systems*, vol. 22, no. 12, pp. 7408–7420, 2020.

[110] J. Miao, Z. Wang, X. Ning, N. Xiao, W. Cai, and R. Liu, "Practical and secure multifactor authentication protocol for autonomous vehicles in 5g," *Software: Practice and Experience*, 2022.

[111] E. P. Kumar and S. Priyanka, "A comprehensive survey on hardware-assisted malware analysis and primitive techniques," *Computer Networks*, vol. 235, p. 109 967, 2023.

[112] T. Alladi, S. Chakravarty, V. Chamola, and M. Guizani, "A lightweight authentication and attestation scheme for in-transit vehicles in iov scenario," *IEEE Transactions on Vehicular Technology*, vol. 69, no. 12, pp. 14 188–14 197, 2020.

[113] S. A. Chaudhry, M. S. Farash, N. Kumar, and M. H. Alsharif, "Pflua-diot: A pairing free lightweight and unlinkable user access control scheme for distributed iot environments," *IEEE Systems Journal*, vol. 16, no. 1, pp. 309–316, 2022. DOI: 10.1109/JSYST.2020.3036425.

[114] Z. Liu, L. Wan, J. Guo, *et al.*, "Ppru: A privacy-preserving reputation updating scheme for cloud-assisted vehicular networks," *IEEE Transactions on Vehicular Technology*, 2023.

[115] Z. Liu, J. Guo, F. Huang, *et al.*, "Lightweight trustworthy message exchange in unmanned aerial vehicle networks," *IEEE Transactions on Intelligent Transportation Systems*, vol. 24, no. 2, pp. 2144–2157, 2023. DOI: 10.1109/TITS.2021.3136304.

[116] Z. Liu, J. Weng, J. Guo, *et al.*, "Pptm: A privacy-preserving trust management scheme for emergency message dissemination in spaceairground-integrated vehicular networks," *IEEE Internet of Things Journal*, vol. 9, no. 8, pp. 5943–5956, 2022. DOI: 10.1109/JIOT.2021.3060751.

[117] M. A. Abdel-Malek, K. Akkaya, A. Bhuyan, and A. S. Ibrahim, "A proxy signature-based swarm drone authentication with leader selection in 5g networks," *IEEE Access*, vol. 10, pp. 57 485–57 498, 2022.

[118] N. Corporation, *Nvidia supernic*, https://www.nvidia.com/en-us/networking/products/ethernet/supernic/, Accessed: 2025-07-26, 2025.

[119] E. F. Kfoury, S. Choueiri, A. Mazloum, A. AlSabeh, J. Gomez, and J. Crichigno, "A comprehensive survey on smartnics: Architectures, development models, applications, and research directions," *IEEE Access*, 2024.

[120] S. Levy, W. Schonbein, and C. Ulmer, "Leveraging high-performance data transfer to offload data management tasks to smartnics," in *2024 IEEE International Conference on Cluster Computing (CLUSTER)*, IEEE, 2024, pp. 346–356.

[121] S. Karamati, C. Hughes, K. S. Hemmert, *et al.*, "smarter nics for faster molecular dynamics: A case study," in *2022 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, IEEE, 2022, pp. 583–594.

[122] S. Basu and D. Nadig, "Offloading nvme over fabrics (nvme-of) to smartnics on an at-scale distributed testbed," in *2024 IEEE 10th International Conference on Network Softwarization (NetSoft)*, IEEE, 2024, pp. 316–318.

[123] S. Hinic, R. A. Bakar, A. Marotta, and F. Paolucci, "Wire-speed ddos attack mitigation using hardware acceleration of programmable dpus," in *GLOBECOM 2024-2024 IEEE Global Communications Conference*, IEEE, 2024, pp. 1197–1202.

[124] M. Usman, S. Iserte, R. Ferrer, and A. J. Peña, "Dpu offloading programming with the openmp api," in *Proceedings of the SC'23 Workshops of the International Conference on High Performance Computing, Network, Storage, and Analysis*, 2023, pp. 884–891.

[125] M. Usman, M. Benito, S. Iserte, and A. Peña, "HPC-Friendly SmartNIC Offloading of Computation/Communication Kernels," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage, and Analysis (SC'25)*, To appear, Nov. 2025.

[126] M. Beebe, B. Michalowicz, D. K. Panda, Y. Chen, W. Poole, and S. Poole, "Openshmem performance on bluefield-3 data processing units (dpus)," in *Practice and Experience in Advanced Research Computing 2025: The Power of Collaboration*, 2025, pp. 1–6.

[127] N. Namashivayam, "Gpu-centric communication schemes for hpc and ml applications," *arXiv preprint arXiv:2503.24230*, 2025.

[128] J. Liu, C. Maltzahn, C. Ulmer, and M. L. Curry, "Performance characteristics of the bluefield-2 smartnic," *arXiv preprint arXiv:2105.06619*, 2021.

[129] WekaIO, *Nvidia bluefield: The smartnic for modern data centers*, `https://www.weka.io/learn/glossary/ai-ml/nvidia-bluefield/`, Accessed: July 26, 2025, 2023.

[130] A. Kashyap, Y. Li, D. Ng, and X. Lu, "Understanding the idiosyncrasies of emerging bluefield dpus," 2025.

[131] B. Michalowicz, K. K. Suresh, H. Subramoni, D. K. Panda, and S. Poole, "DPU-Bench: A MicroBenchmark Suite to Measure Offload Efficiency Of SmartNICs," in *PEARC '23: Practice and Experience in Advanced Research Computing*, ACM, 2023, pp. 94–101. DOI: `10.1145/3569951.3593595`.

[132] B. Michalowicz, K. K. Suresh, H. Subramoni, D. K. D. Panda, and S. Poole, "Battle of the bluefields: An in-depth comparison of the bluefield-2 and bluefield-3 smartnics," in *2023 IEEE Symposium on High-Performance Interconnects (HOTI)*, IEEE, 2023, pp. 41–48.

[133] X. Wei, R. Cheng, Y. Yang, R. Chen, and H. Chen, "Characterizing off-path smartnic for accelerating distributed systems," in *USENIX Annual Technical Conference (USENIX ATC)*, 2023.

[134] H. Tajbakhsh, T. Xing, I. Haque, M. Honda, and A. Barbalace, "Towards portable end-to-end network performance characterization of smartnics," in *Proceedings of the 13th ACM SIGOPS Asia-Pacific Workshop on Systems (APSys '22)*, ACM, 2022.

[135] M. K. Banafaa, Ö. Pepeolu, I. Shayea, *et al.*, "A comprehensive survey on 5g-and-beyond networks with uavs: Applications, emerging technologies, regulatory aspects, research trends and challenges," *IEEE access*, vol. 12, pp. 7786–7826, 2024.

[136] S. M. Mohammed, A. Al-Barrak, and N. T. Mahmood, "Enabling technologies for ultra-low latency and high-reliability communication in 6g networks.," *Ingénierie des Systèmes d'Information*, vol. 29, no. 3, 2024.

[137] A. A. Barakabitze, A. Ahmad, R. Mijumbi, and A. Hines, "5g network slicing using sdn and nfv: A survey of taxonomy, architectures and future challenges," *Computer Networks*, vol. 167, p. 106 984, 2020.

[138] L. Bonati, M. Polese, S. DOro, S. Basagni, and T. Melodia, "Open, programmable, and virtualized 5g networks: State-of-the-art and the road ahead," *Computer Networks*, vol. 182, p. 107 516, 2020.

[139] F. Z. Yousaf, M. Bredel, S. Schaller, and F. Schneider, "Nfv and sdnkey technology enablers for 5g networks," *IEEE Journal on Selected Areas in Communications*, vol. 35, no. 11, pp. 2468–2478, 2017.

[140] A. K. Alnaim, "Securing 5g virtual networks: A critical analysis of sdn, nfv, and network slicing security," *International Journal of Information Security*, pp. 1–21, 2024.

[141] D. Lake, N. Wang, R. Tafazolli, and L. Samuel, "Softwarization of 5g networks–implications to open platforms and standardizations," *IEEE access*, vol. 9, pp. 88 902–88 930, 2021.

[142] S. Malik and S. Bera, "Security-as-a-function in 5g network: Implementation and performance evaluation," in *2024 International Conference on Signal Processing and Communications (SPCOM)*, IEEE, 2024, pp. 1–5.

[143] L. Wernet, L.-M. Spang, F. Siegmund, and T. Meuser, "Resilient user plane traffic redirection in cellular networks," in *2024 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN)*, IEEE, 2024, pp. 1–6.

[144] U. K. Dayalan, Z. Wu, G. Gautam, F. Tian, and Z.-L. Zhang, "Towards an ebpf+ xdp based framework for open, programmable and scalable nextg rans," in *2023 IEEE Future Networks World Forum (FNWF)*, IEEE, 2023, pp. 1–6.

[145] L. Dong, N. Hu, W. Deng, X. Xu, H. Ding, and Y. Huang, "Quality of service evolution and enhancement for holographic video communications toward 6g," *IEEE Wireless Communications*, vol. 32, no. 2, pp. 140–146, 2025.

[146] K. T. Selvi and R. Thamilselvan, "An intelligent traffic prediction framework for 5g network using sdn and fusion learning," *Peer-to-Peer Networking and Applications*, vol. 15, no. 1, pp. 751–767, 2022.

[147] A. Bhattacharyya, S. Ramanathan, A. Fumagalli, and K. Kondepu, "An end-to-end dpdk-inteerated open-source 5g standalone radio access network: A proof of concept," *Computer Networks*, vol. 250, p. 110 533, 2024.

[148] P. Salva-Garcia, R. Ricart-Sanchez, E. Chirivella-Perez, Q. Wang, and J. M. Alcaraz-Calero, "Xdp-based smartnic hardware performance acceleration for next-generation networks," *Journal of Network and Systems Management*, vol. 30, no. 4, p. 75, 2022.

[149] Y. Moon, Y. Han, S. Kim, M. S. Sai, A. Deshmukh, and D. Kim, "Data plane acceleration using heterogeneous programmable network devices towards 6g," in *ICC 2024-IEEE International Conference on Communications*, IEEE, 2024, pp. 421–426.

[150] R. A. Bakar, F. Alhamed, P. Castoldi, *et al.*, "5gdad: A deep learning approach for ddos attack detection in 5g p4-based upf," in *2024 IEEE 25th International Conference on High Performance Switching and Routing (HPSR)*, IEEE, 2024, pp. 185–190.

[151] S. K. Singh, C. E. Rothenberg, J. Langlet, *et al.*, "Hybrid p4 programmable pipelines for 5g gnodeb and user plane functions," *IEEE Transactions on Mobile Computing*, vol. 22, no. 12, pp. 6921–6937, 2022.

[152] T. Schneider, P. Xu, and T. Hoefler, "Fpspin: An fpga-based open-hardware research platform for processing in the network," *arXiv preprint arXiv:2405.16378*, 2024.

[153] Z. Wen and G. Yan, "{Hip4-upf}: Towards {high-performance} comprehensive 5g user plane function on p4 programmable switches," in *2024 USENIX Annual Technical Conference (USENIX ATC 24)*, 2024, pp. 303–320.

[154] F. Paolucci, D. Scano, F. Cugini, *et al.*, "User plane function offloading in p4 switches for enhanced 5g mobile edge computing," in *2021 17th International Conference on the Design of Reliable Communication Networks (DRCN)*, IEEE, 2021, pp. 1–3.

[155] L. Barsellotti, L. De Marinis, F. Cugini, and F. Paolucci, "Ftg-net: Hierarchical flow-to-traffic graph neural network for ddos attack detection," in *2023 IEEE 24th International Conference on High Performance Switching and Routing (HPSR)*, IEEE, 2023, pp. 173–178.

[156] M. Wang, Y. Cui, X. Wang, S. Xiao, and J. Jiang, "Machine learning for networking: Workflow, advances and opportunities," *Ieee Network*, vol. 32, no. 2, pp. 92–99, 2017.

[157] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016, `http://www.deeplearningbook.org`.

[158] Y. Cao, H. Jiang, Y. Deng, J. Wu, P. Zhou, and W. Luo, "Detecting and mitigating ddos attacks in sdn using spatial-temporal graph convolutional network," *IEEE Transactions on Dependable and Secure Computing*, vol. 19, no. 6, pp. 3855–3872, 2021.

[159] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," *arXiv preprint arXiv:1609.02907*, 2016.

[160] A. Gharib, I. Sharafaldin, A. H. Lashkari, and A. A. Ghorbani, "An evaluation framework for intrusion detection dataset," in *2016 International Conference on Information Science and Security (ICISS)*, IEEE, 2016, pp. 1–6.

[161] I. Sharafaldin, A. H. Lashkari, and A. A. Ghorbani, "Toward generating a new intrusion detection dataset and intrusion traffic characterization," *4th International Conference on Information Systems Security and Privacy (ICISSP)*, vol. 1, pp. 108–116, 2018.

[162] M. Sarhan, S. Layeghy, and M. Portmann, "Evaluating standard feature sets towards increased generalisability and explainability of ml-based network intrusion detection," *Big Data Research*, vol. 30, p. 100 359, 2022.

[163] R. Morro, E. Riccardi, D. Scano, *et al.*, "First demonstration of mantra ipowdm convergent sdn architecture using sonic white box and 400zr/zr+ pluggables," in *2023 International Conference on Optical Network Design and Modeling (ONDM)*, IEEE, 2023, pp. 1–3.

[164] K. Goutsos, "Physical unclonability framework for the internet of things," Ph.D. dissertation, Newcastle University, 2020.

[165] C. Aitchison, R. Buckle, A. Chng, C. Clarke, J. Malley, and B. Halak, "On the integration of physically unclonable functions into arm trustzone security technology," in *2020 European Conference on Circuit Theory and Design (ECCTD)*, IEEE, 2020, pp. 1–4.

[166] P. Castoldi, A. Sgambelluri, L. Ismail, F. Paolucci, F. Cugini, and D. Bowden, "Network programmability for smart factory mobile robotics: The smartedge project approach," in *2023 25th International Conference on Transparent Optical Networks (ICTON)*, 2023, p. 0000.

[167] M. Wazid, A. K. Das, N. Kumar, and A. V. Vasilakos, "Design of secure key management and user authentication scheme for fog computing services," *Future Generation Computer Systems*, vol. 91, pp. 475–492, 2019.

[168] AVISPA Project, *AVISPA: Automated Validation of Internet Security Protocols and Applications*, Available online: `http://www.avispa-project.org/`, Accessed in June 2023, 2017.

[169] M. Abdalla, E. Bresson, O. Chevassut, B. Möller, and D. Pointcheval, "Provably secure password-based authentication in tls," in *Proceedings of the 2006 ACM Symposium on Information, computer and communications security*, 2006, pp. 35–45.

[170] C.-C. Chang and H.-D. Le, "A provably secure, efficient, and flexible authentication scheme for ad hoc wireless sensor networks," *IEEE Transactions on wireless communications*, vol. 15, no. 1, pp. 357–366, 2015.

[171] J. Srinivas, A. K. Das, N. Kumar, and J. J. Rodrigues, "Cloud centric authentication for wearable healthcare monitoring system," *IEEE Transactions on Dependable and Secure Computing*, vol. 17, no. 5, pp. 942–956, 2018.

[172] M. Wazid, A. K. Das, V. Bhat, and A. V. Vasilakos, "Lam-ciot: Lightweight authentication mechanism in cloud-based iot environment," *Journal of Network and Computer Applications*, vol. 150, p. 102 496, 2020.

[173] M. Kaveh and M. R. Mosavi, "A lightweight mutual authentication for smart grid neighborhood area network communications based on physically unclonable function," *IEEE Systems Journal*, vol. 14, no. 3, pp. 4535–4544, 2020.

[174] T. Limbasiya, M. Soni, and S. K. Mishra, "Advanced formal authentication protocol using smart cards for network applicants," *Computers & Electrical Engineering*, vol. 66, pp. 50–63, 2018.

[175] Open Networking Foundation (ONF), "SD-Fabric: Open source full-stack programmable leaf-spine network fabric," Open Networking Foundation (ONF), Tech. Rep., Jun. 2021, White paper, June 2021. [Online]. Available: `https://opennetworking.org/sd-fabric/`.